Eine Klasse beweglicher Figuren für interaktive Lernbausteine zur Geometrie

Dissertation zur Erlangung des Doktorgrads
am Institut für Mathematik und ihre Didaktik
der Universität Flensburg

vorgelegt von
Timo Ehmke

Gutachter:
Prof. Dr. Alfred Schreiber
Prof. Dr. Hartmut Wellstein
Prof. Dr. Heinz Schumann

Tag der Disputation:
13. März 2001

Vorwort

In der vorliegenden Arbeit geht es um interaktive Lernbausteine, in denen geometrische Sachverhalte durch bewegliche Figuren auf Web-Seiten dargeboten werden. Diese kurze und knappe Umschreibung deutet bereits die Interdisziplinarität dieses Themas an. Inhalte und Erkenntnisse aus Mathematik, Informatik und Pädagogik sind in diese Arbeit mit eingeflossen. Mein Interesse an den mathematischen Inhalten und die Freude am Entwickeln eigener Software und an der Wissensvermittlung waren meine Motivation, für die Anfertigung dieser Arbeit. Von Anfang an war es für mich eine Herausforderung, im Bereich der dynamischen Geometrie eine eigene Konzeption zu entwerfen und zu realisieren.

Vom Beginn bis zur Fertigstellung dieser Arbeit wurde ich dabei betreut von Herrn Prof. Dr. Alfred Schreiber, dessen Seminare über Bildungstechnologien den Anstoß für diese Arbeit lieferten. Er stand mir bei allen fachlichen und inhaltlichen Fragen zur Seite. Dafür danke ich ihm recht herzlich. Ebenfalls danken möchte ich Herrn Prof. Dr. Hartmut Wellstein und Herrn Prof. Dr. Heinz Schumann, die mir zahlreiche Anregungen und Hinweise zu dieser Arbeit gegeben haben. Herrn Prof. Dr. Heinz Schumann und Herrn Dr. Volker Hole sei insbesondere für die Unterstützung bei der Evaluation des Online-Skripts zur Elementargeometrie gedankt.

Ohne den Rückhalt von familiärer Seite hätte ich diese Arbeit nicht fertigstellen können. Hier danke ich vor allem meiner Frau Katja Ehmke-Janell, daß sie zu mir gestanden hat und mir den Rücken freihielt. Auch meinen Eltern und Schwiegereltern sei für die wohlwollende Unterstützung gedankt.

Zuletzt noch eine Anmerkung: Grundsätzlich habe ich beim grammatischen Geschlecht von Substantiven der flüssigeren Lesbarkeit halber die männliche Form verwendet. Alle weiblichen grammatischen Formen sind dabei jedoch ausdrücklich mit eingeschlossen.

Rotenburg (Wümme), im Juli 2000

Timo Ehmke

Inhaltsverzeichnis

Kapitel 1

Einleitung

In den letzten Jahrzehnten hat sich die Computertechnik rasant entwickelt. In dem Bereich der Geometrie waren vor allem Fortschritte in der Computergrafik bedeutend und manifestierten sich in dem Aufkommen sogenannter Dynamischer Geometrie-Systeme (kurz: DG-Systeme). Diese Programme modellieren geometrische Konstruktionen durch interaktiv bewegliche Figuren und erreichen eine neue Qualität bei der Visualisierung geometrischer Inhalte. Der Begriff Zugmodus spielt dabei eine wichtige Rolle. Im Zugmodus kann man Teile einer Figur mit der Computermaus bewegen. Dabei bleiben alle konstruktiv festgelegten geometrischen Relationen erhalten und Invarianten werden sichtbar. Diese Interaktivität eröffnet neue Wege beim Geometrielernen vor allem bei der induktiven Satz- und Beweisfindung sowie bei der Begriffsbildung.

Beim Einsatz von DG-Systemen im Unterricht lag der Schwerpunkt bislang aber auf dem Konstruieren geometrischer Figuren und auf dem Lösen von Konstruktionsaufgaben. Dabei besteht die Aufgabe des Schülers darin, eine bewegliche Figur zu erzeugen, die bestimmte Eigenschaften erfüllt. Aus didaktischer Sicht ist das Lösen von Konstruktionsaufgaben mit DG-Systemen jedoch nicht unproblematisch. Um eine bewegliche Figur konstruieren zu können, muß der Schüler nämlich den Umgang mit dem Konstruktionswerkzeug beherrschen und das Geometriemodell der Software verstehen. Im Unterschied zu einer Zeichenblattkonstruktion ist beim Entwickeln einer Figur mit einem Konstruktionswerkzeug der Aspekt der Beweglichkeit zu berücksichtigen. Damit kommt im Konstruktionsprozeß eine neue Komplexität hinzu.

Der Ansatz dieser Arbeit besteht in einer Akzentverlagerung weg von der Konstruktion hin zur Benutzung fertiger Figuren. Dazu habe ich ein Konzept für interaktive Lernbausteine zur Geometrie entwickelt. Im wesentlichen besteht ein Lernbaustein aus einer im Zugmodus beweglichen Figur und einer damit verbundenen Aufgabenstellung. Der Schüler soll und kann keine eigenen Konstruktionen mehr erstellen, stattdessen interagiert er mit fertigen Lernbausteinen und setzt sich mit den darin umgesetzten geometrischen Sachverhalten auseinander. Besondere instrumentelle Voraussetzungen werden dafür nicht benötigt. Das Entwickeln eines Lernbausteins wird einem Figurenautor[1] übertragen. Er ist für den geometrischen Lerninhalt und die didaktische Aufbereitung verantwortlich.

[1] Begriffserläuterung auf Seite 9.

Anwendungsfelder für Lernbausteine sind etwa die von Schumann[2] entwickelten interaktiven Arbeitsblätter. Dabei handelt es sich um vorbereitete Figuren, die mit *Cabri-Géomètre* konstruiert wurden. Sie stellen abgeschlossene geometrische Experimente dar, mit denen der Schüler bestimmte Sachverhalte interaktiv untersuchen kann. Besonders gut lassen sich damit funktionale Abhängigkeiten simulieren, z. B., wie das Volumen einer Schachtel von der Form ihres Faltbilds abhängt oder wie der Oberflächeninhalt eines Zylinders mit konstantem Volumen von seiner Form beeinflußt wird.

Das Lernbaustein-Konzept berücksichtigt den Entwicklungsstand aktueller DG-Systeme, setzt aber in den folgenden Punkten neu an:

Publikation im Internet Ziel ist es, Lernbausteine auf Web-Seiten im Internet zu veröffentlichen. Dieses erlaubt einen Zugriff unabhängig von Zeit und Ort. Sind einmal zu einem speziellen Thema Lernbausteine entwickelt und im Internet publiziert worden, so stehen sie zu Lehr- und Lernzwecken allen Interessierten zur Verfügung. Es entstehen keine Kompabilitätsprobleme mit dem Betriebssystem, das auf dem Rechner des Anwenders installiert ist, da bei der Realisation die plattformunabhängige Programmiersprache Java verwendet wurde. Ein weiterer Vorteil ist, daß Lernbausteine ohne zusätzliche Erwerbs- oder Lizenzkosten verwendet werden können.

Skriptsprache *GeoScript* Den Inhalt eines Lernbausteins beschreibt man in der von mir entwickelten Skriptsprache *GeoScript*. Diese läßt sich als eine Art HTML-Erweiterung auffassen, die speziell für die Darstellung geometrischer Inhalte ausgelegt ist. Verglichen mit dem Konstruktionseditor eines DG-Systems ist eine Skriptsprache flexibler. Beispielsweise lassen sich dadurch auch externe Funktionsbibliotheken einbinden. Zur Interpretation von *GeoScript* ist ein als Betrachter bezeichnetes Laufzeitmodul vorgesehen, durch das Lernbausteine auf dem Bildschirm dargestellt werden.

Erweiterte Geometrieauffassung Durch *GeoScript* wird eine erweiterte Geometrieauffassung berücksichtigt. Während aktuelle DG-Systeme in erster Linie zur Konstruktion von euklidischen und abbildungsgeometrischen Figuren ausgelegt sind, können in einem Lernbaustein auch Inhalte aus der Analysis oder der Differentialgeometrie wiedergegeben werden. Es lassen sich außerdem Lernbausteine erstellen, die nicht aus dem Bereich der Geometrie stammen. Beispielsweise kann das aus der Informatik bekannte Problem der acht Damen in einem Lernbaustein umgesetzt werden. Dazu wird ein Schachbrett angezeigt, auf dem sich acht Damen verschieben lassen (Seite 174).

Antwortanalyse Mit Hilfe einer Antwortanalyse ist es möglich, den vom Schüler hergestellten Figurenzustand zu analysieren und als Antwort auf eine Aufgabe zu bewerten. Dazu werden mit *GeoScript* verschiedene Figurenzustände definiert, denen jeweils bestimmte Antwortkommentare zugewiesen werden. Auf diese Weise kann der Figurenautor die richtige Lösung, aber auch unvollständig richtige, teilweise richtige oder falsche Antworten kommentieren. Beispielsweise läßt sich durch eine Antwortanalyse prüfen, ob zwei Dreiecke

[2] Schumann 1997

durch eine bestimmte Abbildungsvorschrift aufeinander abgebildet werden oder ob acht Damen auf einem Schachbrett sich gegenseitig bedrohen.

Die Möglichkeiten, die Lernbausteine beim Lehren und Lernen bieten, werde ich in einer didaktischen Analyse untersuchen. Dabei unterscheide ich zwischen Lernbausteinen im Kontext der Instruktion, Lernbausteinen zum Explorativen Lernen und Lernbausteinen zur Selbstkontrolle. Neben der Analyse dieser didaktischen Funktionen zeige ich in einer nach Themenbereichen gegliederten Beispielsammlung, welche Inhalte sich besonders gut darstellen lassen. Vertiefend stelle ich den Bereich der Elementargeometrie dar. Ein vorhandenes Vorlesungsskript habe ich in Web-Seiten übertragen und mit Lernbausteinen angereichert. Dieses Online-Skript wurde anschließend im praktischen Einsatz evaluiert.

Ausgangsbasis für die Realisation des Lernbaustein-Konzepts war das von David Joyce entwickelte *Geometry-Applet*. Dieser hatte das klassische Werk "Die Elemente" von Euklid im Internet publiziert und mit interaktiven Figuren angereichert. Darüberhinaus veröffentlichte er auch den Quellcode für das Java-Applet. Seine Klassenstruktur diente mir als Grundlage, die ich systematisch um neue Klassen für geometrische Objekte und Funktionale sowie um einen Parser für *GeoScript* erweitert habe.

Als Zukunftsperspektive sind folgende Weiterentwicklungen denkbar: Um die Beschreibung einer Figur mit *GeoScript* zu erleichtern, wäre es möglich, einen interaktiven Editor zu entwickeln, mit dem man Figuren konstruieren und in der Skriptsprache speichern kann. Die Definition einer Antwortanalyse und anderer Funktionen könnte dabei durch eine Art Wizard unterstützt werden. Eine andere Alternative wäre, einen Konverter zu entwickeln, der Dateien, in denen Figuren gespeichert sind, nach *GeoScript* konvertiert. Für Entwickler von DG-Systemen würde es sich außerdem anbieten, eine Funktion zum Speichern einer Figur in *GeoScript* zu realisieren. Auf diese Weise würden sie einen Web-Export erreichen, ohne selbst mit Java ein Laufzeitmodul zum Betrachten von Figuren entwickeln zu müssen.

Im einzelnen gliedert sich diese Arbeit wie folgt:

1. Kapitel Das erste Kapitel enthält die Einleitung, in der ich den Ansatz der Arbeit beschreibe.

2. Kapitel Zur Einführung gebe ich einen kurzen Überblick über die wichtigsten drei Entwicklungsstufen der DG-Systeme. Ich diskutiere, für welche Aufgaben-Typen diese Programme im Unterricht eingesetzt werden können und nenne Vor- und Nachteile. Vor diesem Hintergrund skizziere ich die Konzeption von *Geometria*, dem Laufzeitmodul, mit dem sich Lernbausteine darstellen lassen. Eine Vergleichsübersicht, die Auskunft über den Funktionsumfang aktueller DG-Systeme gibt, bildet den Abschluß dieses Kapitels.

3. Kapitel Das drittel Kapitel umfaßt den softwaretechnischen Teil der Arbeit. Dazu erkläre ich den Aufbau und die Bestandteile eines Lernbausteins. Anschließend zeige ich, wie das Softwaremodell von *Geometria* realisiert ist und beschreibe die Klassenhierarchie der geometrischen Objekte. Dieses Kapitel wird durch einen Vergleich zwischen *Geometria* und anderen Geometrie-Systemen

abgeschlossen, bei dem der Arbeitszyklus der Figurenerstellung im Mittelpunkt steht.

4. Kapitel Das vierte Kapitel bildet den didaktischen Teil der Arbeit. Ich untersuche die didaktischen Möglichkeiten beim Einsatz von Lernbausteinen und unterscheide dabei zwischen den Kontexten Instruktion, Exploratives Lernen und Selbstkontrolle. Die didaktische Analyse wird abgerundet durch eine Beispielsammlung, in der ausgewählte Lernbausteine mit ihren Besonderheiten nach Themenbereichen gegliedert vorgestellt werden.

5. Kapitel Im fünften Kapitel wird der praktische Anwendungsteil dieser Arbeit dargelegt. Ein vorhandenes Vorlesungsskript zur Elementargeometrie habe ich in Web-Seiten umgesetzt und mit interaktiven Lernbausteinen angereichert. Dieses Online-Skript zur Elementargeometrie wurde im praktischen Einsatz erprobt und evaluiert. Dazu erläutere und interpretiere ich die gewonnenen Ergebnisse.

Kapitel 2

Dynamische Geometrie-Systeme in der Diskussion

Im Abschnitt 2.1 möchte ich vorab einige Fachausdrücke erläutern. Um einen ersten Überblick zu geben, werde ich dann in Abschnitt 2.2 die wichtigsten Entwicklungsstufen der DG-Systeme beschreiben und ihre besonderen Eigenschaften nennen. Der vorwiegende Einsatz solcher Programme im Unterricht wird in Abschnitt 2.3 untersucht. Darin wird zwischen Konstruktionsaufgaben und Variationsaufgaben unterschieden. Ihre spezifischen Schwierigkeiten und Probleme beim Unterrichtseinsatz zeige ich in Abschnitt 2.4 auf. Vor diesem Hintergrund wird dann in Abschnitt 2.5 eine alternative Konzeption für ein Geometrie-System vorgestellt. Dieses Kapitel wird durch einen tabellarischen Vergleich von aktuellen Geometrie-Systemen abgeschlossen (Abschnitt 2.6).

2.1 Terminologie

In diesem Abschnitt werde ich die Verwendung einiger Fachbegriffe erläutern.

Schüler Den Begriff Schüler verwende ich in dieser Arbeit als einen Fachausdruck, der synonym für den Lernenden oder den Anwender einer Lernsoftware steht. Ähnlich wie bei der Bezeichnung 'student' im Bereich des Computer-Based Training wird dabei nichts darüber ausgesagt, ob es sich ausschließlich um einen Schüler an einer Schule, einen Studenten an einer Universität oder um sonst eine interessierte Person handelt. Gemeint ist lediglich der Anwender eines Lernprogramms.

Figurenautor Als Figurenautor bezeichne ich die Person, die für die Entwicklung eines Lernbausteins verantwortlich ist. Der Figurenautor ist sozusagen der Experte für die didaktische Aufbereitung des darzustellenden Lerninhalts und für die Definition der Figur in *GeoScript*.

Lehrender Als Lehrenden bezeichne ich die Person, die mit Hilfe eines Lernbausteins einen geometrischen Sachverhalt einer Lerngruppe vermittelt. Der Lernbaustein wird dabei als Medium im Kontext der Instruktion eingesetzt (Abschnitt 4.1). Der Lehrende kann mit dem Figurenautor identisch sein, muß es aber nicht.

Betrachter Als Betrachter wird das Laufzeitmodul bezeichnet, mit dem der Schüler einen Lernbaustein "betrachten" kann. Der Begriff bezieht sich ausschließlich auf die softwaretechnische Komponente und darf nicht mit der Person des Schülers verwechselt werden.

2.2 Entwicklungsstufen Dynamischer Geometrie-Systeme

In diesem Abschnitt beschreibe ich die wichtigsten Entwicklungsstufen der DG-Systeme. Hierbei lassen sich folgende drei Stufen unterscheiden. Die erste Stufe wird durch das Programm *Geometric Supposer* dargestellt, das als direkter Vorläufer der DG-Systeme gesehen werden kann (Abschnitt 2.2.1). Die zweite Stufe wurde durch die Entwicklung von *Cabri-Géomètre* eingeleitet und umfaßt die erste Generation von DG-Systemen (Abschnitt 2.2.2). Die dritte Stufe besteht aus den aktuellen Geometrie-Systemen. Ihre besonderen Merkmale werden im Abschnitt 2.2.3 aufgeführt. Insgesamt soll die Beschreibung der Entwicklungsstufen einen Überblick über die softwaretechnischen und didaktischen Eigenschaften vermitteln.

2.2.1 *Geometric Supposer*

Betrachtet man die Entwicklung der DG-Systeme, so ist als erster Vorläufer das Programm *Geometric Supposer* zu nennen. Dieses wurde von J. L. Schwarz und M. Yerushalmy 1985 konzipiert und kann folgendermaßen gekennzeichnet werden:[1]

Inhaltlich geht es bei diesem Computerprogramm um das Experimentieren mit Dreiecken, Vierecken und Kreisen im Rahmen der Schulgeometrie. Für das induktive Arbeiten mit Dreiecken gab es das Programm *Geometric Supposer Triangle*, für das Arbeiten mit Vierecken und Kreisen existierte jeweils ein eigenes Programm. Der Aufbau dieser drei Systeme war jedoch im wesentlich identisch. Aus diesem Grunde beziehen sich die folgenden Ausführungen speziell auf den *Geometric Supposer Triangle*. Für die beiden weiteren Programme gelten sie entsprechend. Die Kernidee der Software setzt sich aus drei Schritten zusammen:

1. Von dem Programm wird ein Dreieck mit zufälliger Lage, Position und Größe auf dem Bildschirm erzeugt. Ausgehend von einem solchen sogenannten Basisdreieck kann eine geometrische Konstruktion schrittweise ausgeführt werden. Zum Konstruieren stehen folgende Menübefehle zur Verfügung: Strecke, Kreis, Seitenhalbierende, Mittelsenkrechte, Streckenmittelpunkt und Verlängern. Außerdem können an einer Konstruktion

[1] vgl. Biehler 1992, S. 121

Längen, Abstände und Winkel gemessen und Flächeninhalte berechnet werden.

2. Der *Geometric Supposer* speichert die durchgeführten Konstruktionsschritte für den Schüler unbemerkt als eine Prozedur.

3. Der Schüler kann den *Geometric Supposer* veranlassen, ein neues Basisdreieck zu generieren und die gespeicherte Konstruktion wiederholt auszuführen. Hierzu ist der Menübefehl "Neues Dreieck" anzuwählen. Als spezielle Basisdreiecke können rechtwinklige, stumpfwinklige, spitzwinklige, gleichschenklige oder gleichseitige Dreiecke zufällig erzeugt werden.

Der besondere didaktische Vorteil besteht darin, daß der Schüler eine gespeicherte Konstruktion an verschiedenen, zufällig erzeugten Basisdreiecken wiederholt ausführen kann. Dadurch lassen sich Invarianten entdecken, Vermutungen aufstellen und Sätze über geometrische Zusammenhänge formulieren. Problematisch ist beim *Geometric Supposer* die Trennung zwischen Dreiecks-, Vierecks- und Kreiskonstruktionen. Auch läßt sich eine Konstruktionsvorschrift nicht dauerhaft speichern. Die zentrale Erfindung ist aber das – durch die Wiederholungsfunktion realisierte – Prinzip: "Variiere das Basisobjekt, aber nicht die durchgeführte Konstruktion." Dieses Prinzip ist der entscheidende Schritt, der den Zugmodus vorbereitet.

2.2.2 *Cabri-Géomètre*

Das von Jean-Marie Laborde 1988 entwickelte Programm *Cabri-Géomètre* greift die Kernidee des *Geometric Supposers* auf und führt sie weiter. *Cabri-Géomètre* ermöglicht, die Basisobjekte einer Konstruktion direkt mit der Computermaus unter Beibehaltung der Konstruktionsvorschrift zu verschieben. Dadurch entsteht der Eindruck einer stetigen Bewegung der geometrischen Figur. Diese qualitativ neue Interaktionsform wird Zugmodus genannt. Das zufällige Erzeugen eines Basisobjekts in Form eines Dreiecks, Vierecks oder Kreises weicht dem interaktiven menügesteuerten Konstruieren euklidischer Figuren in der Zeichenfläche. Zu diesem Zweck verfügt *Cabri-Géomètre* über ein Konstruktionsmenü. Es unterstützt jedoch noch weitergehende Funktionen:

- **Makrotechnik** Eine Sequenz von Konstruktionsschritten kann zu einem neuen Konstruktionsbefehl zusammengefaßt werden. Dieser läßt sich speichern und beliebig in weiteren Konstruktionen ausführen.

- **Ortslinientechnik** Die Ortsspuren abhängiger Punkte (oder anderer Objekte) können aufgezeichnet werden.

- **Lehrerschnittstelle** Zur individuellen Anpassung des Programms an eine Lerngruppe ist eine Schnittstelle vorgesehen, über die das Repertoire der verfügbaren Konstruktionsbefehle eingeschränkt oder erweitert werden kann.

In Anlehnung an *Cabri-Géomètre* wurden eine Reihe weiterer Konstruktionswerkzeuge veröffentlicht: *Felix* (später *Thales*) von Kadunz & Kautschitsch (1992), *GEOLOG* von Holland (1993), *The Geometer's Sketchpad* von Jackiw

(1992) oder *Euklid* von Mechling (1994). Diese Programme benutzen den Zugmodus, haben Konstruktionsmenüs und unterstützen Makro- und Ortslinientechnik. Man faßt sie unter der Bezeichnung Dynamische Geometrie-Systeme zusammen.

2.2.3 Aktueller Entwicklungsstand

Die DG-Systeme der ersten Generation liegen heute in teilweise mehrfach überarbeiteten Versionen vor. Die oben aufgeführten Merkmale sind zwar weiterhin vorhanden, aber vor allem in drei Bereichen wurden die Programme optimiert und erweitert:

1. **Verbesserung der Benutzerschnittstelle** Die Konstruktionsmenüs wurden in ergonomischer Hinsicht vereinfacht.

2. **Erweiterung der Menge der konstruierbaren Objekte** Die Menge konstruierbarer Objekte wurde vergrößert. Während die DG-Systeme der ersten Generation nur die euklidische Geometrie abdeckten, können in den aktuellen Systemen (Stand: Juli 2000) auch Figuren zur analytischen Geometrie erzeugt werden. Einige aktuelle Programmversionen erlauben sogar die Verwendung von Kegelschnitten in Konstruktionen. Die Ortslinientechnik wurde so erweitert, daß sich Ortslinien als eigenständige, dynamische Objekte in Konstruktionen referenzieren lassen.

3. **Zusatzfunktionen** Zu den Zusatzfunktionen zählen etwa: das automatische Animieren von konstruierten Figuren (Animationstechnik), das Bereitstellen von Werkzeugen zum Messen von geometrischen Größen und zum Prüfen von speziellen Eigenschaften sowie das Anzeigen einer Konstruktionsbeschreibung.

Nähere Auskunft über den Funktionsumfang und die besonderen Eigenschaften aktueller DG-Systeme gibt die tabellarische Übersicht in Abschnitt 2.6.

2.3 Dynamische Geometrie-Systeme im Unterricht

Nachdem die wichtigsten technischen Möglichkeiten der DG-Systeme beschrieben worden sind, soll nun die didaktische Seite betrachtet werden. Im wesentlichen kann man bei den in der Literatur[2] genannten Aufgabenstellungen zum Unterricht mit DG-Systemen zwei Typen unterscheiden:

1. Konstruieren einer geometrischen Figur und

2. Variieren einer geometrischen Figur.

Aufgaben, die sich auf den Konstruktionsprozeß einer geometrischen Figur beziehen, werden hier als Konstruktionsaufgaben bezeichnet. Aufgaben, die sich auf Erkenntnisgewinnung durch Variieren einer gegebenen Figur beziehen, werden Variationsaufgaben genannt.

[2] z. B. Schumann 1991, Elschenbroich 1996, Henn & Jock 1992

In der Unterrichtspraxis der DG-Systeme ist es nicht immer möglich, jede Aufgabenstellung genau einem der beiden Aufgabentypen zuzuordnen. Zuweilen können sich das Konstruieren und das Variieren nacheinander ablösen. Beispielsweise beschreibt Schumann[3] Konstruktionsaufgaben, bei denen das experimentelle Variieren im Zugmodus einen wichtigen Schritt zur Lösung darstellt. Dennoch ist die obige Unterscheidung zweckmäßig: Sie ermöglicht im folgenden eine strukturierte Beschreibung der didaktischen Besonderheiten von DG-Systemen.

2.3.1 Lösen von Konstruktionsaufgaben

Das Ziel einer Konstruktionsaufgabe besteht darin, eine Figur zu konstruieren, die bestimmte geometrische Eigenschaften erfüllt. Mit DG-Systemen lassen sich dabei einige Defizite der traditionellen Instrumente aufheben[4]:

- Geometrische Konstruktionen können mit DG-Systemen schneller und genauer durchgeführt werden.
- Konstruktionsfehler können leichter korrigiert werden (einzelne Konstruktionsschritte lassen sich zurücknehmen).
- Eine Figur kann in der Zeichenfläche anders positioniert werden und läßt sich (auch in der Größe) variieren.
- Eine einmal durchgeführte Konstruktion kann jederzeit wiederholt werden.

Beschreibt man den Lösungsvorgang einer Konstruktionsaufgabe durch einen Drei-Phasen-Prozeß (heuristische Phase, algorithmische Phase, analytische Phase), so entfalten DG-Systeme vor allem in der heuristischen Phase ihre Leistungsfähigkeit. Schumann kennzeichnet diese Phase durch folgende Aktivitäten:

- "Konfiguration zeichnen, die den gegebenen Bedingungen genügt,
- Konfiguration variieren und dabei auf (vollständige) Fallunterscheidung achten,
- erste Überlegungen zur Determination anstellen,
- Konfiguration durch Einzeichnen von Hilfslinien ergänzen,
- Beziehungen von der Konfiguration ablesen,
- heuristische Strategien anwenden."[5]

Besonders die letztgenannte Aktivität stellt einen Vorteil der DG-Systeme im Vergleich zur Zeichenblattgeometrie dar. "Lasse zunächst eine Bedingung an die fertige Lösung fallen und variiere die Daten."[6] Diese von Polya beschriebene Methode läßt sich gewinnbringend zum Lösen von Konstruktionsaufgaben einsetzen. Beim Anwenden der genannten Strategie zur Problemlösung ist die Ortslinientechnik ein wichtiges Hilfsmittel. Ein ausführlich kommentiertes

[3] Schumann 1991, S. 183f
[4] Schumann 1991, S. 35
[5] Schumann 1991, S. 33f
[6] Polya 1949

Beispiel beschreibt Hölzl[7]. Dabei geht es darum, ein gleichseitiges Dreieck mit maximalen Inhalt in ein gegebenes Quadrat einzubeschreiben.

Die algorithmische Phase definiert Schumann als "Ausführung des gefundenen Lösungswegs mit Hilfe von Grundkonstruktionen". Sie wird durch das Erstellen einer Makrokonstruktion abgeschlossen. Die Makrotechnik ermöglicht den Schülern, ein eigenes Konstruktionsrepertoire aufzubauen. Sie unterstützt auf diese Weise die von Kaput[8] formulierte Forderung nach "Ko-Evolution von Werkzeug und geistigem Wachstum der Lernenden"[9]. Die analytische Phase bezeichnet im wesentlichen die Tätigkeiten, die die Variation der Figur betreffen (Abschnitt 2.3.2).

Die Funktion, die DG-Systeme beim Lösen von Konstruktionsaufgaben übernehmen, fassen Graumann et al.[10] als "Verstärker-Funktion" zusammen: Die Werkzeuge ermöglichen es, eine geometrische Figur schneller und genauer herzustellen. Sie verstärken die "zeichnerischen Möglichkeiten, die mit Zirkel und Lineal allein rasch an enge Grenzen stoßen."

2.3.2 Lösen von Variationsaufgaben

Visualisiert der Lehrende in der Einstiegs- oder Problematisierungsphase einen geometrischen Sachverhalt an einer Figur oder demonstriert er in der Übungs- und Anwendungsphase einen funktionalen Zusammenhang, dann steht jedesmal das Variieren im Zugmodus im Vordergrund. Wird einer Lerngruppe eine anregende Situation in Form einer geometrischen Figur angeboten und eine zugehörige Variationsaufgabe gestellt, so setzen sich die Schüler in erster Linie mit dem geometrischen Sachverhalt auseinander. Das Lernziel "Konstruieren geometrischer Figuren" tritt in allen diesen Zusammenhängen in den Hintergrund. Die didaktische Intention besteht hingegen darin, daß die Schüler die Figur im Zugmodus variieren und dabei experimentieren, Eigenschaften erkunden und Besonderheiten entdecken. Von Schumann[11] werden Variationsaufgaben, bei denen eine Figur vorgegeben ist, auch als "interaktive Arbeitsblätter" bezeichnet. Er differenziert folgende Tätigkeiten beim Variieren:

- "Aus einer Konfiguration (als Satz - oder Begriffsrealisat) in großer Variationsbreite viele weitere isomorphe Konfigurationen (mit stetigen Übergängen, d. h. in Realzeitverarbeitung) erzeugen.
- Stetige Übergänge zwischen Sonderfällen derselben Konfiguration erzeugen.
- Aus einem allgemeinen Fall spezielle Fälle einer Konfiguration durch stetige Übergänge erzeugen.
- Aus einem speziellen Fall allgemeinere Fälle einer Konfiguration durch stetige Übergänge erzeugen.
- Grenzfälle einer Konfiguration durch stetige Übergänge erzeugen."[12]

[7] Hölzl 1994, S. 30f
[8] Kaput 1988
[9] Biehler 1992, S. 122
[10] Graumann et al. 1996, S. 198f
[11] Schumann 1997
[12] Schumann 1991, S. 258f

Besonders geeignet sind Variationsaufgaben für die induktive Satzfindung. Der Schüler untersucht hierbei eine Figur durch einen "stetigen, individuell durchgeführten Änderungsvorgang" und prüft, welche Eigenschaften dabei invariant bleiben. "Elementargeometrische Sätze ergeben sich so als Invarianzaussagen bei stetigem Verändern von geometrischen Konfigurationen."[13] Durch das Variieren und Untersuchen von Grenzlagen einer Figur lassen sich Beweisideen anregen oder Begriffe bilden und festigen. In Kapitel 4 werde ich diese Einsatzmöglichkeiten ausführlich untersuchen. Inhaltlich können Variationsaufgaben etwa zu den folgenden Themen gestellt werden:

1. Die Ortslinientechnik kann zur formenkundlichen Untersuchung der Bahnbewegung spezieller Punkte, zur Erforschung von Lage und Art von Bildmengen bei Abbildungen und zur interaktiven Erzeugung von höheren Kurven angewendet werden.[14]

2. Mit den DG-Systemen lassen sich neben den elementar- und abbildungsgeometrischen Figuren auch Inhalte aus der Funktionentheorie experimentell untersuchen. Diese ermöglichen ein dynamisches In-Beziehung-Setzen von geometrischen Größen, wie es ohne diese Systeme nur schwer oder kaum möglich ist.[15] Schüler können dadurch Zusammenhänge zwischen Funktionsgleichung, einzelnen Funktionsparametern und dem Schaubild einer Funktion erforschen.

3. Ein weiterer Einsatzbereich für Variationsaufgaben eröffnet sich durch geometrische Extremwertaufgaben. Der Schüler variiert bei einem solchen Aufgabentyp die Figur und sucht eine Lage, in der bestimmte geometrische Größen extremal werden. Die DG-Systeme unterstützen das Experimentieren durch die Darstellung von Zustandsparametern der Figur. Bestimmte geometrische Extremwertaufgaben, etwa isoperimetrische Probleme, können so graphisch-experimentell gelöst werden.[16]

In Abschnitt 4.4 werde ich eine Reihe von Themen für Variationsaufgaben an konkreten Beispielen vorstellen. Insgesamt kann die Funktion der DG-Systeme beim Lösen von Variationsaufgaben als "Reorganisationsfunktion" beschrieben werden. Graumann et al. schreiben dazu: "Die Software führt so zur Reorganisation unserer bisherigen Tätigkeiten, indem nicht mehr der Vorgang des Konstruierens, das Herstellen, sondern die Manipulation und Untersuchung des Ergebnisses in den Mittelpunkt rückt."[17]

2.4 Kritik an Dynamischen Geometrie-Systemen

Die Vorzüge von DG-Systemen beim Lösen von Konstruktions- und Variationsaufgaben wurden im vorangehenden Abschnitt verdeutlicht. Jedoch belegen Untersuchungen zum Unterrichtseinsatz, daß auch Probleme auftreten können. In diesem Abschnitt sollen daher einige kritische Punkte zum Einsatz der beiden Aufgabentypen mit DG-Systemen diskutiert werden.

[13] Schumann 1991, S. 259
[14] vgl. Schumann 1991, S. 99
[15] vgl. Schumann 1991, S. 258
[16] Schumann 1991, S. 224f
[17] Graumann et al. 1996, S. 199

2.4.1 Probleme beim Lösen von Konstruktionsaufgaben

Um Konstruktionsaufgaben erfolgreich mit einem DG-System lösen zu können, muß der Schüler ein Konstruktionsmenü verstehen und bedienen können. Hole schreibt zu diesen instrumentellen Anforderungen: Das Anwenden bestimmter Konstruktionsbefehle erfordert ein "höheres Maß an Vorüberlegungen [als] beim Konstruieren"[18] mit Zirkel und Lineal. Die Ursache hierfür liegt darin, daß bestimmte Konstruktionsbefehle die Angabe von mehreren Objekten in einer genau festgelegten Reihenfolge erfordern. Hole beschreibt dies konkret an einem Beispiel zur Dreieckskonstruktion mit dem Geometrie-System *GEOLOG*. Hier soll u. a. eine Halbgerade in einem bestimmten Winkel an eine Strecke angetragen werden. Er kommt dabei zu folgendem Schluß: "Das Vorgehen mit *GEOLOG* ist also etwas anspruchsvoller, weil die Schenkel von vornherein als geordnetes Paar (Anfangsschenkel, Endschenkel) zu betrachten sind."[19]

Zu dieser Problematik noch ein zweites Beispiel mit *GEOLOG*: Zu zwei Kreisen k_1 und k_2 mit den Mittelpunkten M_1 und M_2 sollen die gemeinsamen Schnittpunkte erzeugt werden. Der Schüler konstruiert die Schnittpunkte, indem er zuerst den Befehl "Schnittpunkt erzeugen" auswählt und dann nacheinander die beiden Kreisobjekte anklickt. Die Reihenfolge ist dabei wichtig: Wählt er zuerst den Kreis k_1 und als zweites den Kreis k_2 aus, erzeugt *GEOLOG* den Schnittpunkt S_1. Und umgekehrt: Wählt der Schüler zuerst k_2 und dann k_1 aus, wird der Schnittpunkt S_2 bestimmt (Abbildung 2.1). Diese Asymmetrie der Schnittpunktfunktion ist meines Erachtens eine unschöne Eigenschaft des Programms, die den Konstruktionsprozeß erschwert und dem Schüler unnötige Syntaxkenntnisse abverlangt. Neben der Anforderung, einzelne Konstruktions-

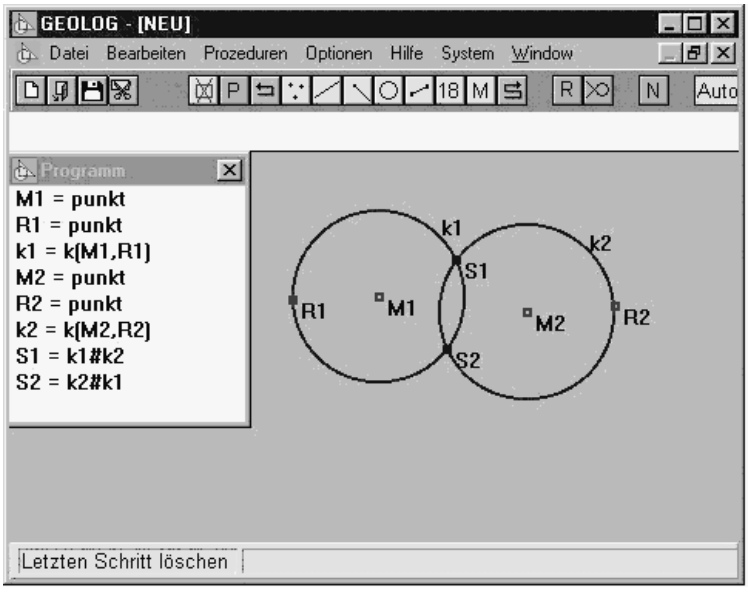

Abbildung 2.1: Kreisschnittpunkte mit *GEOLOG*.

[18] Hole 1998, S. 196
[19] Hole 1998, S. 196

schritte programmspezifisch zu formulieren, zeigen sich noch an anderer Stelle Schwierigkeiten beim Lösen von Konstruktionsaufgaben mit DG-Systemen. Zwischen die subjektiven Vorstellungen des Schülers und das zu konstruierende geometrische Wissen schiebt sich die "Geometrieauffassung" der Software.[20] Dies zeigt sich daran, daß die Lösung einer Konstruktionsaufgabe nicht nur darin besteht, eine statische Zeichnung zu konstruieren. Das Ziel des Schülers ist es vielmehr, eine Figur zu konstruieren, die alle geforderten Eigenschaften besitzt und bei der diese Eigenschaften auch im Zugmodus erhalten bleiben.

Der Schüler ist demnach gezwungen zu wissen, wie die bewegliche Geometrie "funktioniert". Beim Konstruktionsvorgang muß er zwischen dem Erzeugen und Konstruieren diverser Punktobjekte unterscheiden. Bei Hölzl heißt es: "Basis-, Schnitt- und Objektpunkte - drei Sorten von Punkten also, die diejenigen unterscheiden müssen, die mit *Cabri-Géomètre* vernünftig arbeiten wollen. Für diese Unterscheidung besitzen Lernende, die Geometrie vom Zeichenblatt her kennen, keinerlei Vorerfahrung. Demzufolge verwundert es nicht, wenn gerade hier spezifische Schwierigkeiten im Umgang mit dem Programm auftreten."[21] Es stellt sich in diesem Zusammenhang die Frage, ob so etwas wie Vorerfahrung zu dieser Eigenart überhaupt Sinn macht. Letztlich liegt die Unterscheidung von Punktarten in dem Geometriemodell der DG-Systeme begründet, das die Variation im Zugmodus ermöglicht. Zwangsläufig kommt durch diese Beweglichkeit eine neue Dimension beim Konstruktionsprozeß hinzu, die kein Pendant in der Zeichenblatt-Geometrie hat. Die Vorzüge des Zugmodus haben ihren Preis auf der Seite der Konstruktion.

Die Differenzierung zwischen Basis-, Objekt- und Schnittpunkten ist zudem unvollständig, denn sie umfaßt nicht alle Punktobjekte aktueller DG-Systeme. Beispielsweise lassen sich die Bildpunkte einer Drehung nicht in diese Unterteilung einordnen. Deshalb soll vorzugsweise zwischen ziehbaren und nichtziehbaren Punktobjekten differenziert werden.

Die nicht-ziehbaren Punktobjekte kennt der Schüler aus der Geometrie mit Papier und Bleistift. Dies sind alle Punkte, die durch geometrische Konstruktionsschritte erzeugt oder durch analytische Berechnungen bestimmt werden können. Der einzige Unterschied liegt darin, daß mit einem DG-System jedes Punktobjekt in einem separaten Konstruktionsschritt erzeugt wird. In der Zeichenblatt-Geometrie ist dies nicht notwendig. Der Schnittpunkt zweier Geraden ist offensichtlich, vorausgesetzt er liegt nicht außerhalb des Zeichenblatts.

Ziehbare Punktobjekte kennt der Schüler aus der Zeichenblatt-Geometrie nicht. Sie entsprechen am ehesten den Punkten, die man zu Beginn einer Konstruktion beliebig auf einem Zeichenblatt einzeichnet und auf denen die nachfolgenden Konstruktionen aufbauen. DG-Systeme ermöglichen das "Ziehen" dieser Punkte im Zugmodus. Neben den ziehbaren Punkten, die innerhalb der gesamten Zeichenfläche bewegt werden können, gibt es auch Punktobjekte, deren Bewegungsfreiheit auf bestimmte Punktmengen (etwa auf einer Kreislinie oder einem Polygon-Kantenzug) eingeschränkt ist. In Abschnitt 3.3.2 werde ich alle unterschiedlichen Punktobjekte in einem DG-System systematisch differenzieren.

Ob ein Schüler den Unterschied zwischen den verschiedenen Arten von Punkten vollständig verstanden hat, zeigt sich am deutlichsten beim Umgang mit

[20] vgl. Graumann et al. 1996, S. 203
[21] Hölzl 1994, S. 74

den Konstruktionsbefehlen "Objektbindung eines Punkts" und "Auflösen einer Objektbindung". Mit der ersten Funktion können ziehbare Punkte an ein Objekt (z. B. eine Strecke oder einen Kreis) gebunden werden. Die Bewegungsfreiheit des Punkts wird so eingeschränkt. Mit der zweiten Funktion lassen sich solche Objektbindungen wieder auflösen, und die Bewegungsfreiheit des ziehbaren Punkts wird erweitert. An den nicht-ziehbaren Punkten (beispielsweise an Schnittpunkten) können diese Operationen schon per definitionem nicht durchgeführt werden. Einigen Zehntklässlern der Untersuchungsgruppe von Hölzl ist dies auch nach einem halben Jahr nicht vollständig klar geworden. Er schreibt hierzu: "Als softwaretechnische Angelegenheit wird die 'Objektbindung eines Schnittpunktes' gesehen, weil man ja den konstruierten Punkt, den man an ein bestimmtes Objekt binden möchte, meistens ohne Schwierigkeiten auf dem Objekt plazieren kann. Hierbei handelt es sich aber ausschließlich um das Erreichen visueller Inzidenz – dieser Sachverhalt war den meisten Projektteilnehmern bis zuletzt nicht klar. Ihrer Meinung nach müßte *Cabri-Géomètre* nur noch 'nebenbei binden'."[22]

Das wenig gefestigte Verständnis der "Geometrieauffassung" der Software zeigt sich noch an einem weiteren Punkt. Die Schüler versuchen, von dem eigentlichen Ziel einer Konstruktionsaufgabe, dem Finden eines Konstruktionsalgorithmus, abzuweichen. Schumann beobachtete in einer 8. Realschulklasse, daß die Schüler beim Lösen von Konstruktionsaufgaben dazu neigen, sich mit der experimentellen Lösung zufrieden zu geben, die sie mit Verwendung des Zugmodus erhalten.[23] Ähnliches Schülerverhalten analysiert Hölzl in einer Fallstudie über das Lösen einer Einschubaufgabe.[24] In dieser Interaktionsanalyse arbeitet er vier grobe Denk- und Verhaltensmuster heraus:

1. "Vermeiden mathematischer Analyse: Die Schüler sind auf das praktische Herstellen einer Lösung aus und nicht auf die theoretische Durchdringung des Problems. In der Interaktion mit der jeweiligen Microworld dominieren daher visuelle Lösungsstrategien gegenüber analytischen - auch dann, wenn der gegebene Problemdruck analytische Strategien scheinbar unausweichlich macht [...]

2. Umgehen von Werkzeugen [...]

3. Unreflektiertes Benutzen von Werkzeugen [...]

4. Abweichen vom Ziel: Oft läßt sich beobachten, daß die Software in den Lernenden ein eigenständiges Erkundungsinteresse weckt und damit die Problemsicht verändert. Fragen der Lehrenden, die auf deduktive Durchdringung des gestellten Problems zielen, greifen dann nicht mehr."[25]

Insgesamt zeigt sich also: Durch die Möglichkeit, Figuren im Zugmodus zu bewegen, kommt insbesondere beim Lösen von Konstruktionsaufgaben eine neue Komplexität ins Spiel. Die in diesem Abschnitt aufgeführten Schwierigkeiten legen nahe, daß die Schüler oftmals stärker mit der Bedienung des Programms im allgemeinen beschäftigt sind, als mit dem Finden eines adäquaten Konstruktionsalgorithmus.

[22] Hölzl 1994, S. 170
[23] Schumann 1991, S. 35
[24] Hölzl 1994, S. 80f
[25] Hölzl 1995, S. 93f

2.4.2 Defizite bei der Darstellung von Variationsaufgaben

Die Realisierung des Zugmodus eröffnet einen qualitativ neuen Zugang zum Lernen von Geometrie. Für den Lehrenden bietet es sich an, Variationsaufgaben vorzubereiten, um ein "Lernen durch gelenktes Entdecken" (Freudenthal) zu inszenieren.Weil DG-Systeme jedoch in erster Linie als Konstruktionswerkzeuge verstanden werden wollen, können mit ihnen geometrische Lernumgebungen nur in dem Umfang vorbereitet werden, wie sich diese mit dem Konstruktionseditor erstellen lassen. Diese Unterstützung kann jedoch nicht als optimal bezeichnet werden. Bei der didaktischen Aufbereitung eines geometrischen Inhalts in Form einer Variationsaufgabe zeigen sich Defizite der DG-Systeme in den folgenden drei Bereichen:

1. **Die Menge der konstruierbaren Objekte** Die Menge der geometrischen Objekte, die mit aktuellen DG-Systemen erzeugt werden können, orientiert sich im wesentlichen an den Inhalten der euklidischen Geometrie und der Abbildungsgeometrie. Aus diesem Grund sind DG-Systeme wenig flexibel beim Erzeugen von analytisch definierten Objekten. Beispielsweise läßt sich der Graph einer Funktion nicht durch Eingabe einer Funktionsgleichung erzeugen, sondern muß – umständlich – durch ein Ortslinienobjekt realisiert werden. Eine Gegenüberstellung der konstruierbaren Objekte bei aktuellen DG-Systemen findet sich in der tabellarischen Vergleichsübersicht in Abschnitt 2.6.

2. **Rückmeldung des Geometrie-Systems an den Schüler** Bei einer Variationsaufgabe sollte die Figur vom Figurenautor so vorbereitet werden, daß der Lernprozeß des Schülers durch verschiedene Formen der Rückmeldung unterstützt wird. Aus diesem Grund sollte es nicht nur die visuelle Rückmeldung durch die Form und Lage der Figur geben, sondern auch Rückmeldungen durch dauerhaft angezeigte numerische Zustandsparameter oder durch Einblendungen von Textinformationen bei speziellen Figurenzuständen. Der Schüler wird durch solche Zusatzinformationen angeregt, über sein Lernen zu reflektieren. Der Lernprozeß wird so insgesamt positiv unterstützt. Die aktuellen DG-Systeme bieten jedoch hierfür nur begrenzte Funktionen an. Schumann[26] stellt fest: "Der Nachteil interaktiver Arbeitsblätter besteht in den beschränkten Möglichkeiten der Selbstkontrolle des Lernens." Über den Umfang von Rückmeldungen bei aktuellen Geometrie-Systemen gibt die Vergleichsübersicht in Abschnitt 2.6 detailliert Auskunft.

3. **Darstellen von Variationsaufgaben in fremden Softwareumgebungen** Wünschenswert ist es, daß sich Figuren und zugehörige Aufgaben flexibel in fremde Umgebungen einbinden lassen und nicht nur mit dem verwendeten Konstruktionswerkzeug betrachtet werden können. Auf diese Weise könnte eine Variationsaufgabe, die einmal mit besonderer Sorgfalt erstellt wurde, in verschiedene Lernkontexte eingebunden werden, etwa auf Web-Seiten oder in Lernprogrammen. Um dies softwaretechnisch zu realisieren, ist ein Betrachter oder ein Laufzeitmodul erforderlich, das speziell für die Darstellung geometrischer Figuren und Aufgaben ausgelegt

[26] Schumann 1997, S. 2

ist. Eine solche Programmkonzeption besitzen bislang jedoch erst wenige DG-Systeme (s. Vergleichsübersicht in Abschnitt 2.6). Schreiber[27] hat diesen Aspekt allgemein für Aufgaben-Frames in Lernprogrammen unter dem Begriff Selbständigkeit behandelt.

2.5 Das Geometrie-System *Geometria*

In den vorangehenden Abschnitten habe ich die wichtigsten Entwicklungsstufen von DG-Systemen genannt und ihre Eigenschaften dargestellt. Es sind die didaktischen Möglichkeiten dieser Werkzeuge skizziert worden, und es wurde auf bekannte Probleme und Defizite beim Unterrichtseinsatz hingewiesen. Vor diesem Hintergrund will ich das Geometrie-System *Geometria* vorstellen, das ich im Rahmen dieser Arbeit entwickelt habe. Die Konzeption orientiert sich an den Möglichkeiten von dynamischer Geometrie und versucht, die beschriebenen Probleme und Defizite durch einen neuen Ansatz aufzuheben.

Konkret bedeutet dies: Kernidee von *Geometria* ist nach wie vor, daß der Schüler eine geometrische Figur im Zugmodus variiert. Der besondere Ansatz der Konzeption besteht jedoch darin, daß die Konstruktion einer Figur einem Figurenautor übertragen wird, der den Lerninhalt mit der Skriptsprache *GeoScript* beschreibt. Der Schüler kann hingegen keine eigene Konstruktionen mehr durchführen.

Auf den ersten Blick erscheint dies, als würde dadurch lediglich die Interaktivität eines DG-Systems beschnitten. Aber diese Arbeit soll zeigen: Wenn man bei der Konzeption eines Geometrie-Systems auf eine interaktive Konstruktion verzichtet und sich darauf konzentriert, Variationsaufgaben und interaktive Arbeitsblätter mit Hilfe einer Skriptsprache zu realisieren, so eröffnet dieses eine Reihe von neuen Möglichkeiten.

Niedrige instrumentelle Anforderungen Das direkte Nutzen vorbereiteter und unmittelbar verständlicher Figuren entlastet den Schüler von instrumentellen Anforderungen. Er muß nicht die Bedienung eines Konstruktionswerkzeugs erlernen, und er braucht nicht zwischen verschiedenen Punktobjekten zu differenzieren. Insgesamt ist für den Schüler ein weniger tiefes Verständnis der "Geometrieauffassung" der DG-Systeme erforderlich. Der Schüler konzentriert sich auf den geometrischen Inhalt und interagiert mit einer Figur anstatt mit einem Konstruktionseditor.

Definition der geometrischen Figur Der Figurenautor definiert eine geometrische Figur, indem er alle ihre Bestandteile und Relationen zueinander mit *GeoScript* beschreibt. Auf diese Weise können im wesentlichen alle diejenigen Objekttypen definiert werden, die sich auch mit einem DG-System durch einen interaktiven Konstruktionseditor erzeugen lassen. Jedoch geht die Skriptsprache noch einen Schritt weiter. Es lassen sich mehr Objekttypen realisieren und damit auch mehr geometrische Sachverhalte darstellen, die die enge euklidische Geometrieauffassung überwinden.

GeoScript bietet zudem einige strukturelle Vorteile: Der Figurenautor kann sich bei der Definition geometrischer Objekte nicht nur auf zuvor erzeugte Ob-

[27] vgl. Schreiber 1998, S. 142f

jekte als Ganzes beziehen, sondern auch auf alle Objekt- und Systemeigenschaften zugreifen. Über eine Schnittstelle kann er außerdem externe Funktionsbibliotheken einbinden.

Zusatzinformationen und Rückmeldungen Mit *Geometria* können nicht nur geometrische Figuren, Grafiken und Textinformationen angezeigt werden, es lassen sich mit *GeoScript* auch Figurenzustände definieren, um alle diese Bestandteile ein- und auszublenden. Auf diese Weise werden zustandsabhängige Zusatzinformationen gegeben und spezielle Rückmeldungen an den Schüler realisiert.

Antwortanalyse für Variationsaufgaben Das Verwenden einer Skriptsprache zum Definieren von Variationsaufgaben eröffnet die Möglichkeit, eine Antwortanalyse zu realisieren, durch die der vom Schüler hergestellte Zustand einer Figur bewertet und kommentiert werden kann. Durch diese Funktion wird dem Wunsch nach Möglichkeiten zur Selbstkontrolle nachgekommen. Eine detaillierte Beschreibung der Antwortanalyse gebe ich im Abschnitt 3.1.

Publikation auf Web-Seiten Das Geometrie-System *Geometria* wurde mit der Programmiersprache Java als ein sogenanntes Applet entwickelt. Dieses läßt sich flexibel in jedes WWW-Dokument einbinden. Der in Abschnitt 2.4.2 aufgestellten Forderung nach dem Einbinden von Aufgaben und Figuren in fremde Softwareumgebungen wird dadurch Rechnung getragen.

Es zeigt sich also, daß eine Skriptsprache einige Chancen bietet. In den nachfolgenden Kapiteln wird die skizzierte Konzeption im Detail beschrieben.

2.6 Vergleichende Übersicht aktueller Geometrie-Systeme

Mit der folgenden tabellarischen Vergleichsübersicht möchte ich einen Überblick über den Funktionsumfang aktueller DG-Systeme geben. Wie jeder Vergleich von Software unterliegt auch dieser der Gefahr, daß bestimmte Defizite eines Programms bereits mit der Entwicklung der Folgeversion behoben sind. Daher sind die genauen Versionen der untersuchten DG-Systeme im Programmverzeichnis auf Seite 413 beschrieben.

Gegliedert ist die Vergleichsübersicht wie folgt: In Tabelle 2.1 wird gezeigt, welche Typen von ziehbaren und nicht-ziehbaren Punktobjekten bereitgestellt werden. Tabelle 2.2 stellt die Menge aller weiteren Objekte, die erzeugt werden können, dar. Ich unterscheide zwischen eindimensionalen Objekten, Kurven im weitesten Sinne, und zweidimensionalen Objekten. Tabelle 2.3 zeigt, mit welchem Programm man welche Objekteigenschaften messen kann. Diese Programmfunktionen sind notwendig, um dem Schüler Zustandsinformationen der Figur anzuzeigen und um Funktionale zu berechnen. Die Menge der eingebauten Funktionale und die Flexibilität im Umgang mit Termen ist in Tabelle 2.4 aufgeführt. Abschließend werden in Tabelle 2.5 einige weitere Programmoptionen gegenübergestellt.

In dieser letzten Tabelle befindet sich unter der Überschrift "Autorenschnittstelle" der Punkt "Skriptsprache" sowie unter der Bezeichnung "Zusatzinformationen und Feedback" der Punkt "Tutorielle Komponenten". Wegen der besonderen Bedeutung dieser beiden Aspekte für diese Arbeit, möchte ich hierzu einige zusätzliche Anmerkungen geben:

Skriptsprache Neben *Geometria* wird eine Skriptsprache bei vier weiteren Geometrie-Systemen eingesetzt.

In *GEOLOG* ist neben einem interaktiven Konstruktionseditor eine Skriptsprache implementiert, mit der eine Figur durch direkte Eingabe von Befehlen sukzessive konstruiert werden kann. Ein Beispiel habe ich im Abschnitt 2.1 auf Seite 16 gegeben.

Beim *Geometry-Applet* wird eine Figur durch eine Liste von Objektbeschreibungen innerhalb der Applet-Parameter in einem HTML-Dokument definiert.

Das gleiche Verfahren wird bei *Geonet* angewendet. Dieses DG-System ist eine Weiterentwicklung des *Geometry-Applets*. Es realisiert einen erweiterten Befehlsumfang und bietet für die Skripterstellung einen interaktiven Konstruktionseditor. Nachteilig ist bei beiden Skriptsprachen, daß eine Figur nur direkt innerhalb eines HTML-Dokuments und nicht in einer separaten Datei gespeichert werden kann.

Das DG-System *Sketchpad* bietet einen Konverter an, mit dem konstruierte und gespeicherte Figuren nach HTML konvertiert werden können. Ähnlich wie beim *Geometry-Applet* und bei *Geonet* werden die Daten als Applet-Parameter übergeben. Theoretisch könnte ein Figurenautor auch direkt eine entsprechende Web-Seite entwickeln, aber die verwendete Skriptsprache ist dafür wenig geeignet. Alle Objekte werden darin fortlaufend numeriert und durch diese numerischen Bezeichner referenziert. Diese Vorgehensweise ist ungünstig. Sollen etwa in eine bestehende Liste neue Objekte eingefügt werden, muß man sämtliche Referenzen neu setzen. Außerdem sind alle Terme in der polnischen Notation anzugeben. Das ist für den Figurenautor umständlich und unnötig kompliziert.

Tutorielle Komponenten Neben *Geometria* besitzen von den untersuchten Programmen noch *GEOLOG* und *Cinderella* eine tutorielle Komponente.

In dem DG-System *GEOLOG* gibt es zwei tutorielle Programmkomponenten: *GEOKON* und *GEOBEWEIS*. Die Komponente *GEOKON* realisiert das Lösen von Konstruktionsaufgaben in Form von Interpolationsproblemen. Dabei wird das Konstruktionsziel durch eine Planfigur und eine Liste von Zieleigenschaften vorgegeben. Als Konstruktionswerkzeuge stehen die aus *GEOLOG* bekannten Befehle zur Verfügung. Jeder Konstruktionsschritt des Schülers wird nach der Durchführung analysiert und entweder akzeptiert oder vom System mit einem Kommentar zurückgewiesen. Zu jeder Zeit kann sich der Schüler eine mehrstufige Hilfe abrufen. Nach durchgeführter Konstruktion liefert *GEOKON* Rückmeldung über die Anzahl richtiger Lösungsschritte, falscher Lösungsschritte und abgerufener Hilfetexte. Die Abbildung 2.2 zeigt eine Konstruktionsaufgabe mit *GEOKON*, bei der eine Figur zu konstruieren ist, die die beiden Zieleigenschaften $AM = BM$ und $BM = CM$ besitzt. Das Fenster links unten in der Abbildung zeigt die zur Aufgabe gehörige Planfigur. Das Hauptfenster enthält die Lösungsfigur.

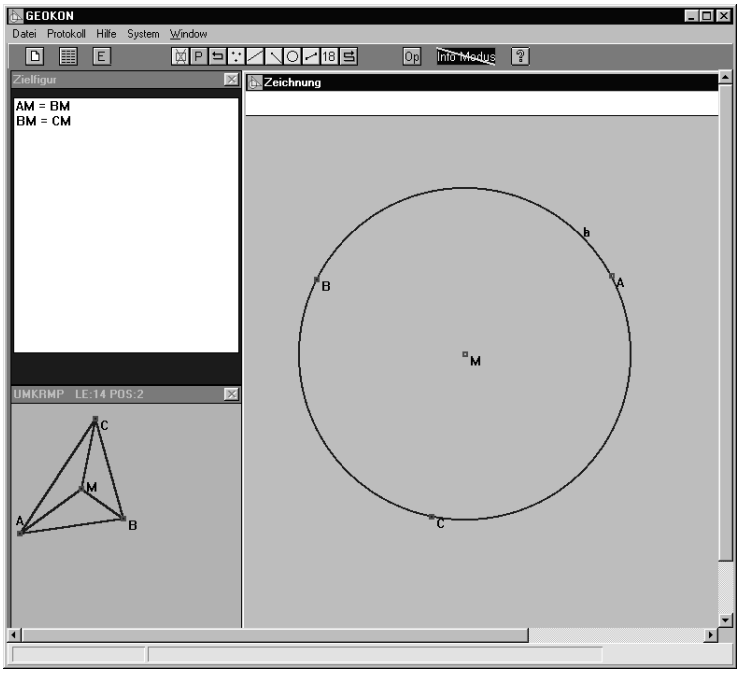

Abbildung 2.2: *GEOKON*: Eine tutorielle Komponente für Konstruktionsaufgaben.

Die zweite tutorielle Programmkomponente *GEOBEWEIS* ist ähnlich aufgebaut. Mit ihr werden Beweis- und Berechnungsaufgaben als Interpolationsprobleme realisiert. Eine solche Aufgabe besteht aus einer Menge von Voraussetzungen und der zu beweisenden Behauptung. Diese Bestandteile werden in einem Beweisfenster durch sogenannte Knoten symbolisiert. Ferner ist eine Planfigur und eine Liste mit Sätzen gegeben. Die Satzliste enthält alle für die Interpolation zugelassenen Operatoren. Wiederum steht dem Schüler eine mehrstufige Hilfe zur Verfügung, und alle vom Schüler durchgeführten Beweisschritte werden von *GEOBEWEIS* analysiert und entsprechend kommentiert.

Mit dem DG-System *Cinderella* kann ein Figurenautor interaktive Konstruktionsaufgaben für Web-Seiten vorbereiten, bei denen die tutorielle Komponente die Konstruktion des Schülers automatisch kontrolliert und bewertet. Abhängig von den einzelnen Konstruktionsschritten lassen sich Lösungshilfen anzeigen. Diese Form von interaktiven Konstruktionsaufgaben eignet sich insbesondere für den Einsatz in webbasierten tutoriellen Lernumgebungen. Die Abbildung 2.3 zeigt ein Beispiel für eine Konstruktionsaufgabe mit *Cinderella*, bei der der Mittelpunkt zweier gegebener Punkte A und B konstruiert werden soll. Als Konstruktionswerkzeuge stehen mehrere Buttons etwa zum Erzeugen von Punkten, Geraden und Kreisen zur Verfügung. Als Hilfen werden Lösungshinweise in das Textfenster ausgegeben sowie ggf. einzelne Konstruktionsschritte an der Figur vorgeführt. *Cinderella* überwacht bei einer Aufgabe den Konstruktionsprozeß des Schülers und kommentiert einzelne richtige Schritte. Überflüssige Konstruktionsschritte werden allerdings nicht berücksichtigt und bleiben unkommentiert.

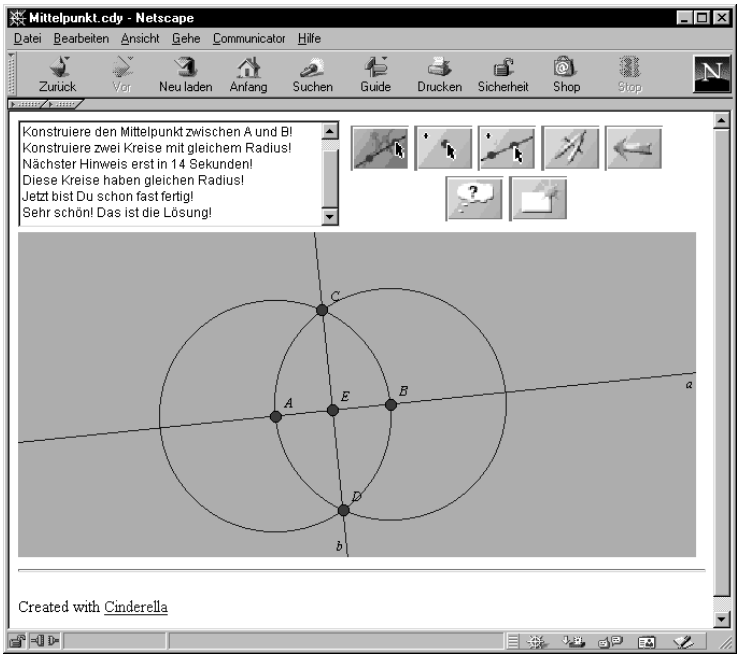

Abbildung 2.3: *Cinderella*: Interaktive Konstruktionsaufgabe mit automatischer Lösungskontrolle.

Verglichen mit den beiden DG-Systemen *GEOLOG* und *Cinderella* besitzt *Geometria* eine tutorielle Komponente, die nicht für das Analysieren von Konstruktionsaufgaben ausgelegt ist. Stattdessen geht es bei der Antwortanalyse in *Geometria* um das Bewerten und Kommentieren von Variationsaufgaben. Wie die Antwortanalyse softwaretechnisch realisiert ist, beschreibe ich im Abschnitt 3.1. Die didaktischen Möglichkeiten von Lernbausteinen mit Antwortanalyse erläutere ich in Abschnitt 4.3.

Tabelle 2.1: Punktobjekte.

	Cabri II	Cinderella	Dr. Geo	Euklid	Geolog	Geometry Applet	Geonet	Sketchpad	Thales	Zirkel und Lineal	Geometria
Punktobjekte (ziehbar in / auf ...)											
– Punktmenge											•
– Zeichenfläche	•	•	•	•	•	•	•	•	•	•	•
– Polygonfläche											•
– Kreisfläche											•
– Ortslinie	•		•					•			•
– Kegelschnitt	•	•									
– Kurve											•
– Polygon	•							•			•
– Kreislinie	•	•	•	•	•	•	•	•	•	•	•
– Gerade	•	•	•	•	•	•	•	•	•	•	•
– Strahl	•		•	•	•			•	•	•	•
– Strecke	•		•	•	•	•		•	•	•	•
Punktobjekte (nicht ziehbar)											
Fixpunkt	•	•		•		•	•	•		•	•
Schnittpunkt	•	•	•	•	•	•	•	•	•	•	•
Bildpunkt einer ...											
– Drehung	•		•					•	•		•
– Verschiebung	•		•					•			•
– Spiegelung	•	•	•	•				•	•		•
– Streckung	•							•			•
– Kreisspiegelung	•	•				•	•				•
– benutzerdef. Abbildung								•			•

Tabelle 2.2: Ein- und zweidimensionale geometrische Objekte.

	Cabri II	Cinderella	Dr. Geo	Euklid	Geolog	Geometry Applet	Geonet	Sketchpad	Thales	Zirkel und Lineal	Geometria
Objekte (eindimensional)											
Ortslinie	•	•	•	•	(•)		(•)	•	(•)	•	•
Kegelschnitt	•	•									
Kurve											•
Polygon	•	•		•	•	•	•	•			•
Kreislinie	•	•	•	•	•	•	•	•	•	•	•
Kreisbogen	•			•		•	•	•			•
Gerade	•	•	•	•	•		•	•	•	•	•
Strahl	•		•	•	•			•	•	•	•
Strecke	•	(•)	•	•	•	•	•	•	•	•	•
Stange				•						•	
Objekte (zweidimensional)											
Polygonfläche	•	•		•		•	•	•			•
Kreisfläche	•					•	•	•	•		•
Kreissegmentfläche						•	•	•			•
Punktmenge											•

Tabelle 2.3: Eigenschaften einzelner geometrischer Objekte.

	Cabri II	Cinderella	Dr. Geo	Euklid	Geolog	Geometry Applet	Geonet	Sketchpad	Thales	Zirkel und Lineal	Geometria
Messen von Objekteigenschaften											
Punktobjekte:											
- Koordinaten	•	•	•	(•)			•	•		•	•
Geradenobjekte:											
- Geradengleichung	•							•			•
- Gleichungsparameter								•			•
- Steigung	•							•			•
Kurvenobjekte:											
- Kurvengleichung											•
Kegelschnittobjekte:											
- Kegelschnittgleichung	•										
Kreisobjekte:											
- Kreisgleichung	•							•			•
- Gleichungsparameter								•			•
- Radius							•	•	•	•	•
- Umfang	•							•	•		•
- Flächeninhalt	•	•						•	•		•
Polygonobjekte:											
- Umfang	•							•			•
- Flächeninhalt	•	•						•	•		•
- Symmetrien											•
- Konvexität											•
- Regularität											•
- Schwerpunkt											•
- autom. Dreieckssklassifikation											•
- autom. Viereckssklassifikation											•

Tabelle 2.4: Funktionale und Terme.

	Cabri II	Cinderella	Dr. Geo	Euklid	Geolog	Geometry Applet	Geonet	Sketchpad	Thales	Zirkel und Lineal	Geometria
Funktionale											
Abstand	•	•	•	•	•		•	•	•	•	•
Winkel	•	•	•	•	•		•	•	•		•
Streckenverhältnisse								•			•
Ähnlichkeit											•
Kollinearität	•										•
Kongruenz											•
Inzidenz	•										•
Inklusion											•
Parallelität	•										•
Orthogonalität	•										•
Berechnen von Termen											
Termevaluation	•			•	•		•	•	•		•
Meßwerte einbinden	•			•			•	•	•		•
Fallunterscheidungen											•
benutzerdef. Funktionen							•				•
externe Funktionsbibliotheken											•

Tabelle 2.5: Verschiedene Programmoptionen.

	Cabri II	Cinderella	Dr. Geo	Euklid	Geolog	Geometry Applet	Geonet	Sketchpad	Thales	Zirkel und Lineal	Geometria
Zusatzinformationen und Feedback											
Textfenster	•	•		•	•			•		•	•
zustandsabh. Textfenster											•
Hilfetexte		•			•						•
Tutorielle Komponenten		•			•						•
Autorenschnittstelle											
Konstruktionsmenü	•	•	•	•	•		•	•	•	•	
benutzerdef. Menüs	•	•		•				•			
Skriptsprache					•	•	•	•			•
Darstellung auf Web-Seiten	•	•		•		•	•	•			•
Sonderfunktionen											
Makrotechnik	•		•	•	•		•	•	•	•	
Animationstechnik	•	•					•	•			•
Zustandsraum begrenzen											•

Kapitel 3

Konzeption und softwaretechnische Realisation von Lernbausteinen zur Geometrie

In diesem Kapitel wird die Konzeption und softwaretechnische Realisation von Lernbausteinen zur Geometrie vorgestellt. Ich beschreibe, aus welchen definitorischen Bestandteilen ein Lernbaustein besteht (Abschnitt 3.1) und wie das Konzept softwaretechnisch umgesetzt worden ist (Abschnitt 3.2). In Abschnitt 3.3 erläutere ich die einzelnen Klassen, aus denen sich das Softwaremodell zusammensetzt. Abschließend wird das vorgestellte Geometrie-System mit anderen DG-Systemen verglichen und die Unterschiede werden diskutiert (Abschnitt 3.4).

3.1 Grundlegende Definitionen

Ein Lernbaustein LB besteht aus einer Zeichenfläche Z, einer Figur F, einer Menge T von Textfenstern, einer Menge B von Bildern, einer Menge H von Hilfen und einer Antwortanalyse α, kurz $LB = (Z, F, T, B, H, \alpha)$.

Zeichenfläche Die Zeichenfläche Z ist die Fläche des Fensters, das auf dem Bildschirm dargestellt wird und in dem die Figur, alle Textfenster und alle Bilder angezeigt werden. Die Größe der Zeichenfläche wird durch die Angabe der Breite und Höhe festgelegt. Innerhalb der Zeichenfläche ist ein Weltkoordinatensystem definiert, das zur Positionierung der Figur dient.

Figur Eine Figur F ist das Datenmodell einer Konstruktion von geometrischen Objekten. Jedes geometrische Objekt ist eine Instanz einer der folgenden

Klassen: `PointElement` (Punkte), `LineElement` (Strecken, Strahlen und Geraden), `CurveElement` (Kurven), `LocusElement` (Ortslinien), `CircleElement` (Kreise), `SectorElement` (Kreissegmente), `PolygonElement` (n-Ecke), `PointSetElement` (Punktmengen) und `Measure` (Funktionale).

Jede dieser Klassen ist ein spezieller Datentyp, durch den ein geometrisches Objekt mit idealen Eigenschaften softwaretechnisch abgebildet wird. Beispielsweise wird das geometrische Objekt Kreis durch die Klasse `CircleElement` realisiert. In der obigen Aufzählung von Klassen sind die Objektbezeichner jeweils in Klammern hinter den Klassenbezeichnern angegeben. Die einzelnen Klassen werden in Abschnitt 3.3 beschrieben.

Das Datenmodell wird durch Darstellung einiger Objekte in der Zeichenfläche visualisiert. Welche Objekte der Figur dabei angezeigt werden, hängt von der Initialisierung der einzelnen Objekte und vom Zustand des Datenmodells ab. Der Schüler kann den Zustand des Datenmodells gezielt verändern, indem er bestimmte (ziehbare) Punktobjekte in der Zeichenfläche verschiebt. Dabei bleiben alle konstruktiv festgelegten geometrischen Relationen erhalten. Der Zustand des Datenmodells zu einem bestimmten Zeitpunkt wird als Figurenzustand s bezeichnet. Die Menge aller Figurenzustände heißt Zustandsraum S. Der Zustandsraum einer Figur kann gezielt verkleinert werden, indem der Figurenautor Figurenzustände definiert, die die Figur nicht annehmen soll. Weiterhin kann man spezielle Figurenzustände definieren, in die die Figur beim Eintritt von bestimmten Ereignissen (z. B. Drücken eines Buttons) versetzt wird.

Textfenster Ein Textfenster T_i ist eine Textinformation, die in einem umrahmten Bereich der Zeichenfläche dauerhaft oder abhängig vom Figurenzustand angezeigt wird. In einem Textfenster lassen sich Aufgabenstellungen oder Zusatzinformationen formulieren.

Bilder Ein Bild B_i ist eine Grafik-Datei, die in der Zeichenfläche dauerhaft oder abhängig vom Figurenzustand angezeigt wird. Dies kann illustrativen Zwecken dienen oder dazu, Formeln typographisch korrekt darzustellen. Außerdem lassen sich Bilder an die Position von ziehbaren Punktobjekten binden. Sie werden so zu quasi-geometrischen Objekten. Aus softwaretechnischen Gründen müssen die Grafik-Dateien im GIF- oder JPG-Format angegeben werden.

Hilfen Eine Hilfe H_i ist ein Text, der dem Schüler Hilfestellung beim Lösen einer Aufgabe geben soll. Alle vorbereiteten Hilfen können nacheinander abgerufen werden. Jede Hilfe wird nur einmal, jeweils in einem separaten Fenster angezeigt.

Antwortanalyse Im Zusammenhang mit einer Variationsaufgabe kann man den Figurenzustand s als die vom Schüler erstellte Antwort oder Lösung auffassen. Mit Hilfe einer Antwortanalyse α ist es dann möglich, den Figurenzustand zu bewerten und zu kommentieren. Dazu wird jeder Antwort s genau ein Antwortwert v_i aus einer endlichen Menge V von Antwortwerten zugeordnet, d. h. α ist eine Abbildung $\alpha\colon S \to V$.

Mögliche Antwortwerte sind: R = richtig, U = unvollständig richtig, T = teilweise richtig/falsch, F = falsch und N = nicht-identifizierbar. Im primitivsten Fall ist $V = \{\mathrm{R}, \mathrm{F}\}$. Eine Antwort läßt sich aber auch differenzierter bewerten,

indem man mehrere unvollständig richtige $U_1, \ldots, U_k$, mehrere teilweise richtige $T_1, \ldots, T_l$ und mehrere falsche $F_1, \ldots, F_m$ Antwortwerte definiert.

Die Definition eines Antwortwerts v_i erfordert vom Figurenautor die Angabe eines Prüfschlüssels und einer Liste $Kt_i[1, \ldots, h_i]$ von Kommentaren.

Ein Prüfschlüssel[1] ist eine charakteristische Eigenschaft des Figurenzustands und wird durch eine booleschwertige Prüffunktion $\texttt{Ps}(s)$ realisiert. Softwaretechnisch werden Prüffunktionen mit Hilfe der Klasse `MeasureCondition` umgesetzt (Abschnitt 3.3.10). Zur Definition einer Prüffunktion mit *GeoScript* sind die gängigen mathematischen Operationen und Funktionen verfügbar. Ferner können außerdem eingebaute oder vom Figurenautor selbst definierte Funktionale verwendet werden. Den genauen Befehlsumfang habe ich in der Konstruktionsreferenz (Anhang B, Seite 238) erläutert. Ein Beispielskript diskutiere ich im Abschnitt 3.2.3.

Die Liste $Kt_i[1, \ldots, h_i]$ von Kommentaren besteht aus Antworttexten, die der Schüler als Rückmeldung angezeigt bekommt. Für jeden Antwortwert kann eine beliebig lange Liste von Antworttexten in *GeoScript* definiert werden. Die Variable h_i wird als spezifische Höchstzahl bezeichnet und gibt an, wie oft der Antwortwert v_i zurückgegeben werden muß, bevor der Zyklus der Antwortanalyse beendet wird.

Die genaue Abwicklung der Antwortanalyse in einem Lernbaustein zeigt das Struktogramm nach Eckel[2] in der Abbildung 3.1. Nach dem Starten eines Lernbausteins werden die beiden Zähler z und z_i eingeführt und auf 0 gesetzt. Dem Schüler wird eine Variationsaufgabe gestellt, bei der die Figur in einen bestimmten Zustand versetzt werden soll. Wenn der Schüler die Figur variiert hat und den Auswertungsbutton betätigt, beginnt die Antwortanalyse. Der Gesamtzähler z wird um Eins erhöht. Die sich nun anschließende Unterschleife berechnet den Index i vom Antwortwert v_i, für den $\texttt{Ps}_i(s) = 1$ ist. Sobald der erste Index i mit $\texttt{Ps}_i(s) = 1$ ermittelt wurde, wird der Zähler z_i inkrementiert und mit der spezifischen Höchstzahl h_i verglichen. Ist h_i noch nicht erreicht, folgt der Vergleich von z mit der Gesamthöchstzahl h. Sobald eine der beiden Höchstzahlen erreicht wurde, wird der Zyklus mit der Ausgabe des Kommentars $Kt_i[h_i]$ verlassen. Andernfalls wird dem Schüler der Kommentar $Kt_i[z_i]$ dargeboten und er erhält einen weiteren Antwortversuch.

Je nachdem, in welchen Lernkontext eine Aufgabe eingebunden ist, kann es sinnvoll sein, die Anzahl von Antwortversuchen, die durch die Gesamthöchstzahl h festgelegt wird, nicht zu begrenzen. Daher gilt folgende Vereinbarung: Wird $h = 0$ gesetzt, dann hat der Schüler für eine Aufgabe unbegrenzt viele Lösungsversuche. Das Ablaufschema in Abbildung 3.1 ändert sich dadurch geringfügig.

Die vorgestellte Antwortanalyse habe ich realisiert, indem ich das von Schreiber[3] beschriebene allgemeine Verfahren der Antwortanalyse in einem Aufgaben-Frame speziell auf Lernbausteine zur Geometrie angewendet habe.

[1] vgl. Schreiber 1998, S. 166f
[2] Eckel 1989, S. 17f
[3] Schreiber 1998, S. 147f

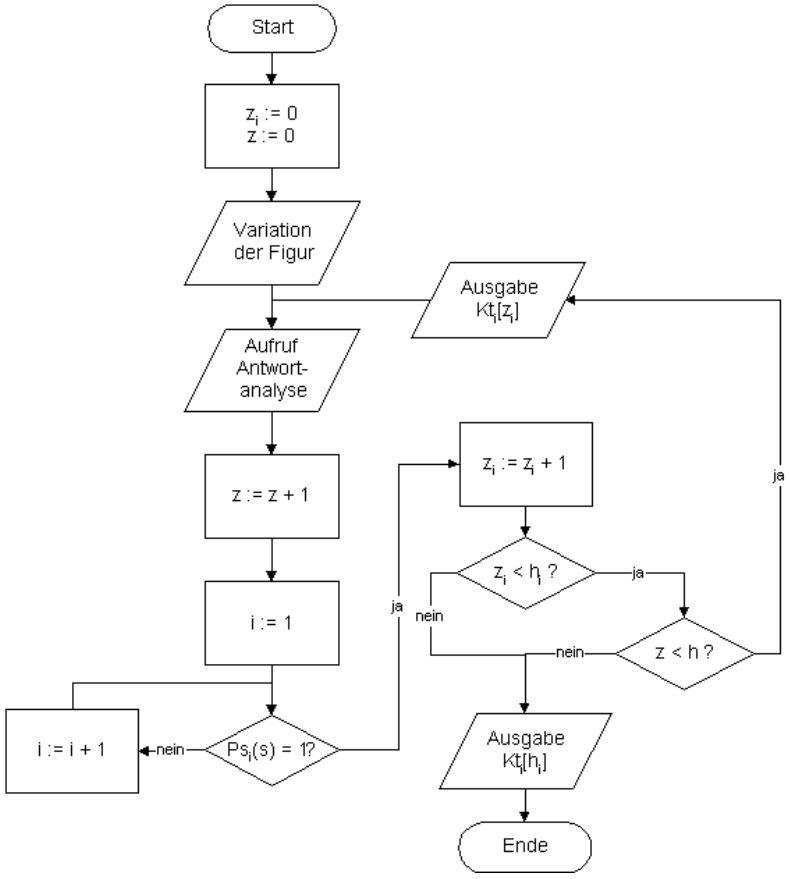

Abbildung 3.1: Struktogramm der Antwortanalyse in einem Lernbaustein.

3.2 Das Softwaremodell für Lernbausteine

In diesem Abschnitt beschreibe ich das Softwaremodell von *Geometria*, durch das Lernbausteine technisch realisiert werden. Dazu skizziere ich zuerst den allgemeinen Aufbau (Abschnitt 3.2.2). Anschließend werden die Aufgaben und die Funktionsweise der einzelnen Komponenten erläutert (Abschnitte 3.2.3-3.2.5). Vorab möchte ich jedoch auf die Arbeit von David Joyce eingehen, die die Ausgangsbasis für die Entwicklung von *Geometria* darstellte.

3.2.1 Die Vorarbeit von David Joyce

Bei der Entwicklung von *Geometria* habe ich auf der Vorarbeit von David Joyce aufgebaut. Joyce hatte 1996 das klassische Werk von Euklid "Die Elemente" im World Wide Web veröffentlicht und mit interaktiven, im Zugmodus beweglichen Figuren angereichert.[4] Zu diesem Zweck entwickelte er ein Java-Programm mit dem Titel *Geometry-Applet* und veröffentlichte den Quellcode im Internet. Dieses Java-Applet bildete den Ausgangspunkt für die Entwicklung von *Geometria*. Von der vorhandenen Klassenstruktur ausgehend, habe ich die Konzeption überarbeitet, einige Algorithmen durch neue ersetzt und zahlreiche Klassen und Funktionen hinzugefügt.

Das *Geometry-Applet* besteht im Kern aus den drei Klassen `Geometry`, `Slate`[5] und `Element`. Die wichtigsten Änderungen und Erweiterungen, die an diesen Klassen vorgenommen wurden, sollen kurz angeführt werden.

Änderungen in `Geometry`

- Die Klasse `Geometry` von Joyce diente als Ausgangsbasis für die neu erstellte Klasse `Geometria`. Die Definition einer Figur erfolgt darin nicht mehr wie beim *Geometry-Applet* durch die Übergabe von Parameterwerten innerhalb der HTML-Seite. Stattdessen wurde ein anderer Weg beschritten: Die Bestandteile einer Figur und die Parameter zur Darstellung werden mittels *GeoScript* in zwei separaten Dateien beschrieben.
- Der Befehls- und Funktionsumfang zur Definition des geometrischen Inhalts und des Layouts wurde erheblich vergrößert. Entsprechend wurden auch die Funktionen des Parsers erweitert, der diese Daten einliest und interpretiert.

Änderungen in `Slate`

- Die Ereignisbehandlung wurde dem Standard der Java-Version 1.1.5 entsprechend aktualisiert.
- Die Darstellung der geometrischen Objekte in der Zeichenfläche erfolgt im Unterschied zum *Geometry-Applet* unter Verwendung von Welt- und Fensterkoordinaten.
- Alle erzeugten geometrischen Objekte werden, anstatt in einer linearen Liste, in einer Eltern-Kind-Hierarchie verwaltet, wodurch sich die Geschwindigkeit des Zugmodus verbessern ließ.

[4] Joyce 1996, http://aleph0.clarku.edu/~ djoyce/java/elements/toc.html
[5] Slate = engl. Schiefertafel

- Neue Funktionen bezüglich der Zeichenfläche wurden geschaffen: Ein Rasterfangmodus kann verwendet werden, und es lassen sich ein Gitternetz oder ein Koordinatensystem darstellen.
- Um die geometrischen Objekte einer Figur auf der Zeichenfläche darzustellen, wurde das Verfahren der Doppelpufferung (engl. Double Buffering) beibehalten, nachdem Versuche ergeben haben, daß andere Methoden weniger effektiv sind.

Änderungen in `Element`

- Das Applet von Joyce wurde entwickelt, um die euklidische Geometrie zu visualisieren. Aus diesem Grund besteht es auch nur aus Klassen, welche die wichtigsten geometrischen Objekte realisieren, die sich mit Zirkel und Lineal konstruieren lassen. So kommt das *Geometry-Applet* sogar ohne Klassen für die Objekte Gerade und Strahl aus. Dennoch hat Joyce es nicht versäumt, die Konzeption der Klasse `Element` so anzulegen, daß eine Erweiterung der Klassenhierarchie um neue Objekte leicht möglich ist. An diesem Punkt setzen die Erweiterungen der Klasse `Element` an, indem zahlreiche weitere geometrische Objekte durch neue Klassen (z. B. funktionsabhängige Punktobjekte, dynamische Ortslinien, parametrisierte Kurven, beliebige Punktmengen, usw.) realisiert wurden.
- Die wichtigste Erweiterung besteht in der Klasse `Measure`, mit der Funktionale erzeugt werden können. Funktionale sind wichtig für die Definition geometrischer Objekte wie etwa funktionsabhängige Punkte oder parametrisierte Kurven. Sie werden außerdem bei der Antwortanalyse eingesetzt, und sie ermöglichen die Realisation einer Schnittstelle für externe Funktionsbibliotheken.

Das *Geometry-Applet* (Version 2.0) ist für die Darstellung von dreidimensionalen geometrischen Objekten ausgelegt, in diese Richtung ist *Geometria* aber nicht weiter ausgebaut worden.

3.2.2 Das Softwaremodell von *Geometria*

Das Softwaremodell von *Geometria* besteht aus drei Komponenten: einem Skript, einer Layout-Vorlage und einem Betrachter (Abbildung 3.2). In einem Skript

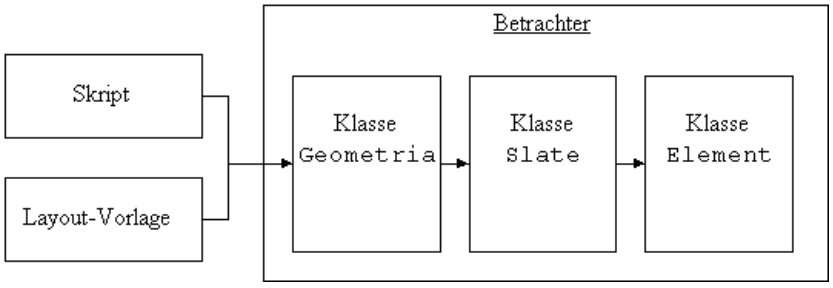

Abbildung 3.2: Das Softwaremodell von *Geometria*.

definiert der Figurenautor die Bestandteile eines Lernbausteins, in der Layout-Vorlage legt er den Layout-Stil fest. Der Betrachter ist das eigentliche Laufzeitmodul: Er stellt die sichtbaren Bestandteile eines Lernbausteins auf dem Bildschirm dar, verwaltet alle zugehörigen Daten und realisiert die Interaktion mit dem Schüler. Softwaretechnisch besteht der Betrachter aus den drei Klassen `Geometria`, `Slate` und `Element`.

3.2.3 Das Skript

Ein Skript ist eine Textdatei, in der der Figurenautor durch *GeoScript*-Ausdrücke die inhaltlichen Bestandteile eines Lernbausteins beschreibt. Dazu gehören: Figur, Textfenster, Bilder, Hilfen sowie Antwortwerte mit Prüffunktionen und Kommentaren. Die Syntax von *GeoScript* wird im folgenden anhand eines Beispiels erläutert. Der Übersichtlichkeit halber zeige ich zuerst das zugehörige Skript als Ganzes. Bei der Beschreibung der Syntax werde ich mich dann auf das Beispiel beziehen.

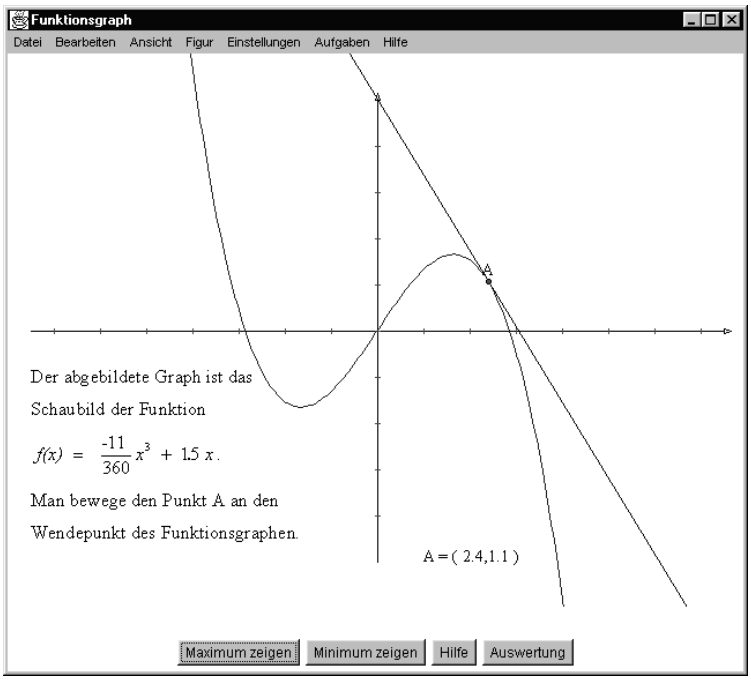

Abbildung 3.3: Beispiel-Lernbaustein.

Lernbaustein-Beispiel

Die Abbildung 3.3 zeigt einen Lernbaustein in dem der Graph der Funktion

$$f(x) = \frac{-11}{360}x^3 + \frac{3}{2}x$$

dargestellt ist. Auf dem Funktionsgraphen kann ein Punkt A gezogen werden. Durch A wird zusätzlich die Tangente an den Graphen angezeigt. Das folgende

Skript zeigt die Definition des Lernbausteins mit *GeoScript*. Die Zeilennummern wurden eingefügt, um das Skript anschließend leichter kommentieren zu können.

```
   //
   // Datei: Funktionsgraph.script
   //

 5 // Systemvariablen
   // ===============

   WORLD_X_MAX   = +8.0
   WORLD_X_MIN   = -8.0
10 WORLD_Y_MAX   = +6.0
   WORLD_Y_MIN   = -6.0

   // Definition der Figur
   // ====================
15
   e[1]  = O;  point;   fixed;          0.0,0.0; "hidden"
   e[2]  = c;  point;   coordSystem;    O,300,300,200,200; 0;red;black;0
   e[3]  = k;  line;    curve;          "t","-11*t^3/60+1.5*t",-8.0,8.0,300;
   e[4]  = A;  point;   dragable;       0.5,0.5,k;
20 e[5]  = m0; measure; coordinates;    A,1.0,-5.0,"A = ","";
   e[6]  = m1; measure; calculate;      "-11*coordinateX(A)^3/60+1.5*coordinateX(A)";
   e[7]  = m2; measure; calculate;      "-33*coordinateX(A)^2/60+1.5";
   e[8]  = A'; point;   functionDepend; "coordinateX(A)+1.0","coordinateY(A)+calculate(m2)"; "hidden"
   e[9]  = t;  line;    straightLine;   A,A'; "hideLabel"
25 e[10] = b1; measure; button;         "Maximum zeigen","action";
   e[11] = b2; measure; button;         "Minimum zeigen","action";
   e[12] = b3; measure; button;         "Hilfe","help";
   e[13] = b4; measure; button;         "Auswertung","evaluate";

30 // Beschränken des Zustandsraums der Figur
   // ========================================

   limit[1] = "coordinateX(A) < 3"
   limit[2] = "coordinateX(A) > -3"
35
   // Ein- und Ausblenden von Objekten
   // ================================

   hidden[1] = "if (abs(coordinateX(A)+1.65144) > 0.01) hide (Textbox_1)"
40 hidden[2] = "if (abs(coordinateX(A)-1.65144) > 0.01) hide (Textbox_2)"

   // Erzeugen von speziellen Figurenzuständen
   // ========================================

45 move[1] = "if (calculate(b1)) move (A, 1.65144, 1.65144)"
   move[2] = "if (calculate(b2)) move (A, -1.65144, -1.65144)"

   // Definition von Textfenstern
   // ===========================
50
   <TextBox>
    Position = 340;60;-1;-1
    An dieser Stelle des Graphen
    liegt ein lokales Minimum vor.
55 </TextBox>

   <TextBox>
    Position = 340;60;-1;-1
    An dieser Stelle des Graphen
```

```
 liegt ein lokales Maximum vor.
</TextBox>

// Einbinden von Hilfen
// ====================

<Help>
 In dem Wendepunkt einer Kurve geht
 eine Linkskurve in eine Rechtskurve
 über (oder umgekehrt).
</Help>

// Einbinden von Bildern
// =====================

image[1] = "\pics\Funktionsgraph01.gif", 20, 270

// Definition einer Antwortanalyse
// ===============================

<Problem>
 MAX_ANSWER = 0
 condition[1] = "abs(coordinateX(A)) < 0.01"
 condition[2] = "abs(coordinateX(A)+1.65144) < 0.01"
 condition[3] = "abs(coordinateX(A)-1.65144) < 0.01"
</Problem>

<Answer 1>
 key = "condition[1]"
 comment[1] = "Richtig.
               Der Wendepunkt des Funktionsgraphen
               liegt im Koordinatenursprung."
</Answer 1>

<Answer 2>
 key = "condition[2]"
 comment[1] = "Ihre Antwort ist nicht richtig.
               Der Wendepunkt darf nicht mit dem lokalen Minimum verwechselt
               werden. Versuchen Sie es noch einmal."
</Answer 2>

<Answer 3>
 key = "condition[3]"
 comment[1] = "Ihre Antwort ist nicht richtig.
               Der Wendepunkt darf nicht mit dem lokalen Maximum verwechselt
               werden. Versuchen Sie es noch einmal."
</Answer 3>

<Answer 4>
 key = "NOT(condition[1] OR condition[2] OR condition[3])"
 comment[1] = "Ihre Antwort ist nicht richtig. Versuchen Sie es noch einmal."
</Answer 4>
```

Systemvariablen

Sämtliche Systemvariablen von *Geometria* werden in der Layout-Vorlage (Abschnitt 3.2.4) mit Werten belegt. Sollen jedoch bestimmte Systemvariablen aus der Layout-Vorlage überschrieben werden und nur für den aktuellen Lernbaustein gültig sein, so können sie auch innerhalb eines Skripts definiert werden. Die Abbildung 3.4 zeigt, durch welche Syntax den Systemvariablen Werte zuge-

wiesen werden.

Im Beispiel wird in den Zeilen 8-11 das Weltkoordinatensystem innerhalb der Zeichenfläche neu definiert.

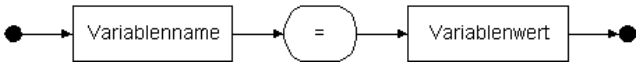

Abbildung 3.4: Syntaxdiagramm zur Definition von Systemvariablen.

Definition der Figur

Die Definition der Figur in einem Lernbaustein orientiert sich an einer – aus der Zeichenblatt-Geometrie bekannten – Konstruktionsbeschreibung (bei der nacheinander sämtliche Konstruktionsschritte aufgeführt werden). Mit *GeoScript* werden ebenfalls nacheinander alle geometrischen Objekte definiert, aus denen die Figur besteht. Jedes Objekt ist dabei ein Element der Liste `e[1..n]`. Das folgende Diagramm (Abbildung 3.5) zeigt die Syntax, durch die ein geometrisches Objekt definiert wird. Die ovalen Kästen enthalten vordefinierte Befehlswörter oder reservierte Bezeichnungen. Die Kreise umfassen die Trennzeichen *,* und *;*. In den Rechtecken stehen verschiedene Syntaxelemente, deren jeweilige Bedeutung im folgenden erläutert wird.

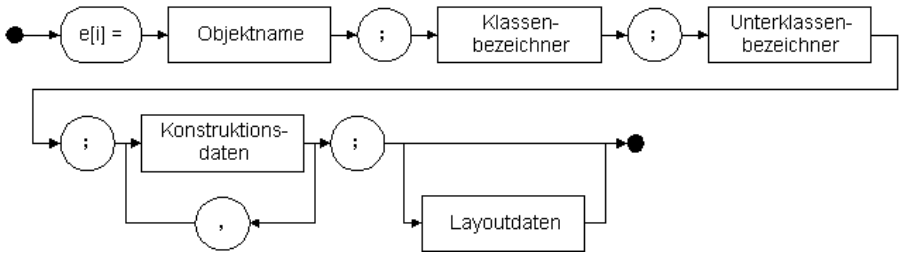

Abbildung 3.5: Syntaxdiagramm zur Definition eines geometrischen Objekts.

In dem Syntaxdiagramm ist die Indizierung der Liste `e[i]` mit der Variablen `i` so zu verstehen, daß für den ersten Listeneintrag `i = 1` gesetzt wird, für den zweiten `i = 2`, usw. Im folgenden wird diese Schreibweise mehrfach verwendet.

Der Objektname ist die Beschriftung des zu erzeugenden Objekts, in dem Beispiel (Zeile 16-28) sind dies: `O`, `c`, `k`, `A` usw. Der Klassenbezeichner ist die Bezeichnung für die Klasse des zu erzeugenden Objekts, im Beispiel kommen `point`, `line` und `measure` vor.[6] Der Unterklassenbezeichner ist die Bezeichnung für die Unterklasse des zu erzeugenden Objekts. Die Unterklasse[7] `functionDepend` (Zeile 23) umfaßt beispielsweise alle funktionsabhängigen Punktobjekte (Abschnitt 3.3.2). Die Konstruktionsdaten sind die Daten, die zur Konstruktion des zu erzeugenden Objekts erforderlich sind. Im Beispiel erfordert die

[6] Alle weiteren in *GeoScript* verfügbaren Befehlswörter sind in der Konstruktionsreferenz beschrieben (Anhang B, Seite 238).

[7] Eine Unterklasse ist eine abgeleitete Klasse, die eine Spezialisierung derjenigen Klasse darstellt, die in der Klassenhierarchie eine Stufe höher steht.

Konstruktion der Geraden `t` die Angabe zweier Punkte `A` und `A'` (Zeile 24). Die Layoutdaten bestimmen, ob und wie die Darstellung des zu erzeugenden Objekts von den Vorgaben aus der Layout-Vorlage abweichen soll. Die Angabe von Layoutdaten ist optional. In dem Beispiel wird der Objektname der Geraden `t` durch den Befehl `hideLabel` nicht auf der Zeichenfläche angezeigt (Zeile 24).

Beschränken des Zustandsraums der Figur

Um den Zustandsraum der Figur zu beschränken, ist die Angabe einer Liste `limit[1..n]` erforderlich. Jeder Listeneintrag legt mit Hilfe einer Prüffunktion einen Figurenzustand fest, den die Figur nicht einnehmen soll. Die entsprechende Syntax ist in Abbildung 3.6 dargestellt. Als Prüffunktion ist dabei eine

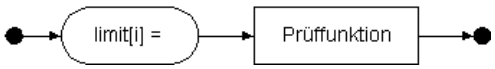

Abbildung 3.6: Syntaxdiagramm zur Beschränkung des Zustandsraums.

booleschwertige Funktion anzugeben. Realisiert wird diese durch eine Instanz der Klasse `MeasureCondition` (Abschnitt 3.3.10). Eine Figur kann im Zugmodus variiert werden, solange keine der definierten Prüffunktionen den Wert `0` annimmt. In diesem Fall wird der letzte gültige Figurenzustand wieder hergestellt.

In dem Beispiel wird in den Zeilen 33 und 34 der Zustandsraum so verkleinert, daß der Punkt A nur so auf dem Graphen verschoben werden kann, daß seine x-Koordinate nicht kleiner als -3 und nicht größer als 3 wird.[8]

Ein- und Ausblenden von Objekten

Die Menge der Objekte, welche abhängig vom Figurenzustand ein- und ausgeblendet werden sollen, wird durch die Einträge der Liste `hidden[1..n]` definiert. Jeder Listeneintrag muß die Syntax aus der Abbildung 3.7 erfüllen. Neben ei-

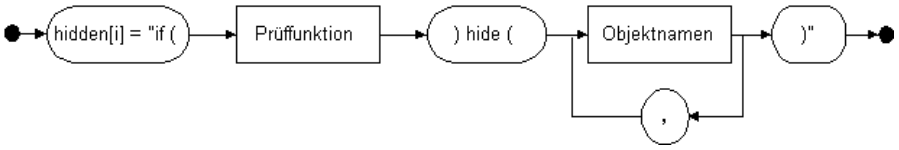

Abbildung 3.7: Syntaxdiagramm zum Ein- und Ausblenden von Teilfiguren.

ner Prüffunktion müssen die Objektnamen angegeben werden, die abhängig von dem Wert der Prüffunktion angezeigt oder nicht angezeigt werden. Im Beispiel wird das erste definierte Textfenster angezeigt, wenn die x-Koordinate von A den Wert $1,65144 \pm 0,01$ besitzt (Zeile 40). Die Abbildung 3.8 verdeutlicht diesen

[8] Inhaltlich macht diese Beschränkung in dem Beispiel wenig Sinn. Sie wurde lediglich zur Demonstration mit in das Skript aufgenommen. Ein besseres Beispiel wäre folgendes: Der Zustandsraum eines Vierecks bestehend aus vier frei ziehbaren Eckpunkten läßt sich so beschränken, daß man nur noch konvexe Vierecke herstellen kann.

Sachverhalt. Das zweite Textfenster wird eingeblendet, wenn die x-Koordinate von A den Wert $-1,65144 \pm 0,01$ annimmt (Zeile 39).

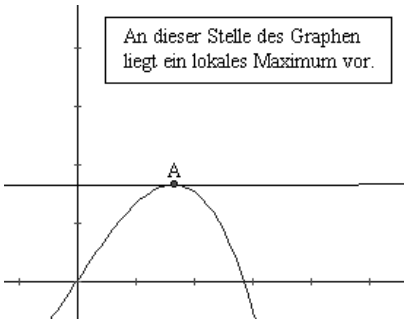

Abbildung 3.8: Einblenden eines Textfensters.

Erzeugen von speziellen Figurenzuständen

Um die Figur durch Auslösen eines Ereignisses in einen speziellen Zustand (Zielzustand) zu versetzen, muß eine Prüffunktion angegeben werden. Verschiedene Zielzustände werden durch die Einträge in der Liste `move[1..n]` festgelegt. Die Figur wird genau dann in einen solchen Zielzustand versetzt, wenn die zugehörige Prüffunktion den Wert 1 annimmt. Die entsprechende Syntax ist in der Abbildung 3.9 dargestellt. Der Punktobjektname ist der Bezeichner eines

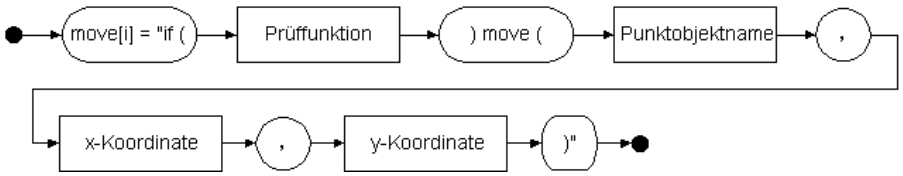

Abbildung 3.9: Syntaxdiagramm zum Erzeugen von speziellen Figurenzuständen.

Punktobjekts. Diesem Punktobjekt werden die angegebenen Koordinatenwerte (x-Koordinate, y-Koordinate) zugewiesen, sobald die Prüffunktion den Wert 1 annimmt. Alle vom Punktobjekt abhängigen Objekte werden daraufhin neu berechnet.

In dem Beispiel werden durch den Befehl in Zeile 45 dem Punkt A die Koordinaten $(1,65144, 1,65144)$ zugewiesen, sobald der Button "Maximum zeigen" gedrückt wird. Die Abbildung 3.8 zeigt diesen Figurenzustand. Die Anweisung in Zeile 46 bewirkt, daß A die Koordinaten $(-1,65144, -1,65144)$ zugewiesen werden, sobald der Button "Minimum zeigen" betätigt wird.

Definition von Textfenstern

Textfenster werden mit *GeoScript* nach der in Abbildung 3.10 dargestellten Syntax definiert. Die Positionsdaten geben die Lage des Textfensters in der

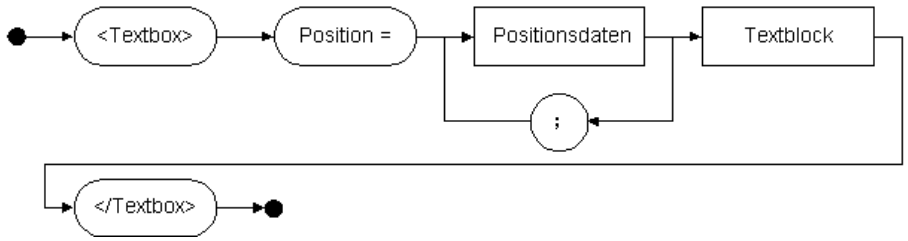

Abbildung 3.10: Syntaxdiagramm zur Definition eines Textfensters.

Zeichenfläche an. Sie bestehen aus den folgenden vier Werten:

1. x-Koordinate des linken, oberen Eckpunkts,
2. y-Koordinate des linken, oberen Eckpunkts,
3. Breite des Fensters in Bildschirmpunkten und
4. Höhe des Fensters in Bildschirmpunkten.

In dem Beispiel werden zwei Textfenster in den Zeilen 51-61 definiert. Von den beiden Textfenstern besitzt die linke, obere Ecke jeweils die Fensterkoordinaten $(340, 60)$. Der Wert -1 bewirkt, daß die Breite und Höhe des Textfensters automatisch berechnet wird. Der Textblock kann eine beliebige Anzahl von Textzeilen umfassen.

Definition von Hilfen

Die Abbildung 3.11 zeigt, welche Syntax erforderlich ist, um mit *GeoScript* Hilfen zu definieren. Der Textblock umfaßt eine beliebige Anzahl von Textzeilen.

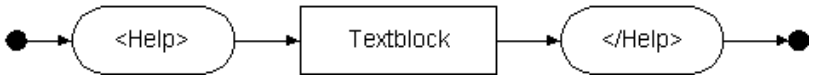

Abbildung 3.11: Syntaxdiagramm zur Definition einer Hilfe.

Eine Positionsangabe ist nicht erforderlich, da die Hilfetexte in einem separaten Fenster angezeigt werden. Der Schüler kann die Hilfetexte in der Reihenfolge abrufen, in der sie im Skript definiert sind. Im Beispiel wird eine Hilfe in den Zeilen 66-70 definiert. Die Abbildung 3.12 zeigt die Hilfe in einem separaten Fenster.

Einbinden von Bildern

Alle Bilddateien, die mit *GeoScript* eingebunden werden sollen, müssen innerhalb des Skripts durch eine Liste `image[1..n]` angegeben werden. Dabei ist pro Listeneintrag die folgende Syntax zu erfüllen (Abbildung 3.13).

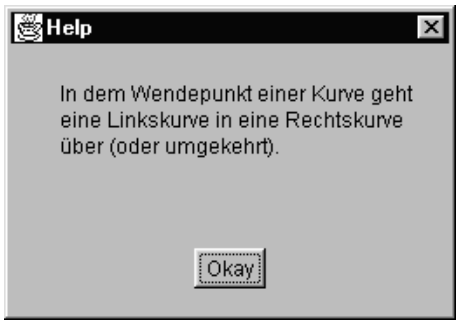

Abbildung 3.12: Hilfetext zum Lernbaustein aus Abbildung 3.3.

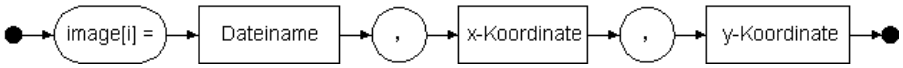

Abbildung 3.13: Syntaxdiagramm zum Einbinden von Bilddateien.

In dem Beispiel wird in der Zeile 75 die Bilddatei mit den Dateinamen "Funktionsgraph01.gif" mit der linken, oberen Ecke an der Position $(20, 270)$ in der Zeichenfläche dargestellt. In dem Bild wird der Aufgabentext und die Funktionsgleichung des Graphen angezeigt.

Definition einer Antwortanalyse

Die Definition einer Antwortanalyse mit *GeoScript* ist unterteilt in einen Hauptabschnitt und mehrere Unterabschnitte. In dem Hauptabschnitt wird die Gesamthöchstzahl und eine Liste `condition[1..n]` von Prüffunktionen festgelegt, in jedem Unterabschnitt wird jeweils ein Antwortwert definiert.[9] Die Abbildung 3.14 zeigt die Syntax des Hauptabschnitts. In dem vorangestellten Beispiel be-

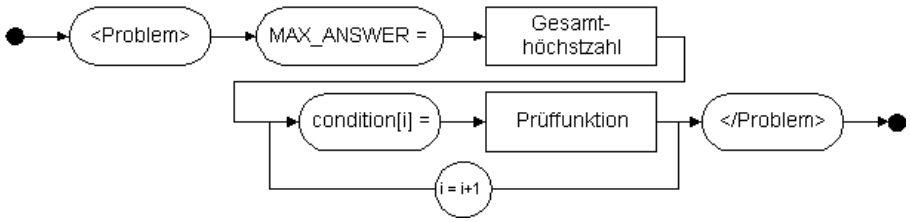

Abbildung 3.14: Syntaxdiagramm zur Definition einer Antwortanalyse.

steht die Aufgaben des Schülers darin, den Punkt A an den Wendepunkt des Funktionsgraphen zu bewegen. Die Antwortanalyse soll prüfen, ob der Punkt A im Koordinatenursprung und damit im Wendepunkt der Funktion liegt. Außerdem soll als spezielle Falschantwort erkannt werden, ob A mit einem der beiden Extremwerte der Funktion zusammenfällt.

[9] vgl. Abschnitt 3.1

In dem Beispielskript wird der Hauptabschnitt in den Zeilen 80-85 definiert. In der Zeile 81 ist die Gesamthöchstzahl auf 0 gesetzt. Vereinbarungsgemäß bedeutet dies, daß der Schüler für diese Aufgabe unbegrenzt viele Antwortversuche hat. In den Zeilen 82-84 werden drei Prüffunktionen beschrieben, die wiederum bei der Definition der einzelnen Antwortwerte referenziert werden. Die erste Prüffunktion testet, ob die x-Koordinate von Punkt A gleich 0 ist (Wendepunkt). Die zweite Prüffunktion untersucht, ob die x-Koordinate A gleich $-1,65144$ ist (lokales Minimum). Die dritte Prüffunktion vergleicht die x-Koordinate mit dem Wert $1,65144$ (lokales Maximum). Dabei wird jeweils eine Toleranz von $\pm 0,01$ berücksichtigt.

Jeder Unterabschnitt, in dem ein Antwortwert definiert wird, besitzt die in Abbildung 3.15 veranschaulichte Syntax. Die Prüffunktion definiert den Prüfbe-

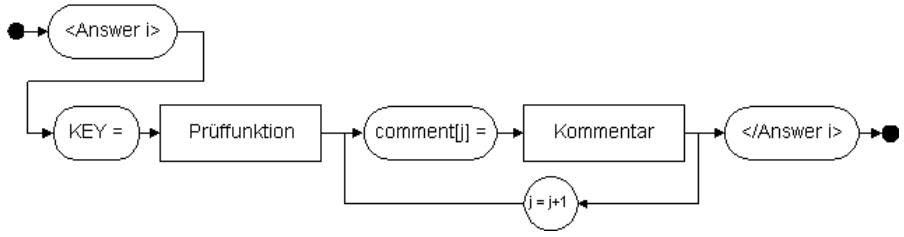

Abbildung 3.15: Syntaxdiagramm zur Definition eines Antwortwerts.

reich eines Antwortwerts. Im Beispiel besteht der Prüfbereich des ersten Antwortwerts (Zeile 87-92) aus allen Figurenzuständen, für die `condition[1]` erfüllt ist, wobei `condition[1]` nur auf die im Hauptabschnitt definierte Prüffunktion verweist, die prüft, ob der Punkt A im Koordinatenursprung liegt. Für die Definition der Prüffunktionen wurde der Hauptabschnitt bewußt so gewählt, daß damit möglichst keine redundanten Objekte der Klasse `MeasureCondition` erzeugt werden. Die Prüffunktionen zur Definition der Antwortwerte werden daher als boolesche Verknüpfungen der im Hauptabschnitt definierten Prüffunktionen realisiert.

Der Kommentar zu einem Antwortwert besteht aus der Liste `comment[1..n]` und entspricht dem Kommentar Kt_i in der Abbildung 3.1. Durch die Anzahl der Listeneinträge n wird gleichzeitig die spezifische Höchstzahl h_i festgelegt (Abschnitt 3.1). In dem Beispiel ist zu jedem Antwortwert jeweils nur ein Kommentar angegeben. Der erste Antwortwert ist die Lösung der Aufgabe, die drei folgenden sind Falschantworten.

3.2.4 Die Layout-Vorlage

Innerhalb der Layout-Vorlage beschreibt der Figurenautor einen Layout-Stil für das Erscheinungsbild eines Lernbausteins. Der Inhalt der Layout-Vorlage unterteilt sich in zwei Abschnitte:

Systemvariablen

Durch die Systemvariablen wird der Layout-Stil des Betrachterfensters und der Zeichenfläche sowie eine Reihe von Eigenschaften wie etwa Fenstergröße, Hin-

tergrundfarben, Schriftarten, usw. festgelegt. Für die Definition einer Systemvariablen ist die in Abbildung 3.16 dargestellte Syntax vorgeschrieben.

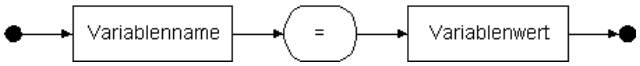

Abbildung 3.16: Syntaxdiagramm zur Definition von Systemvariablen.

Beispiel:

```
APPLET_WIDTH      = 480
APPLET_HEIGHT     = 360
WORLD_X_MAX       = +16.0
WORLD_X_MIN       = -16.0
WORLD_Y_MAX       = +12.0
WORLD_Y_MIN       = -12.0
FONTSIZE          = 14
FONT              = SERIF
APPLETBGCOLOR     = 255,225,200
LANGUAGE          = GERMAN
SHOWGRID          = FALSE
```

Sämtliche Systemvariablen, die in *GeoScript* zur Verfügung stehen, sind in der Konstruktionsreferenz beschrieben (Anhang B, Seite 238).

Layout-Stil der Figur

Der Layout-Stil, der das Aussehen der geometrischen Objekte bestimmt, wird durch eine Tabelle festgelegt. Dabei ist die in der Abbildung 3.17 dargestellte Syntax erforderlich.

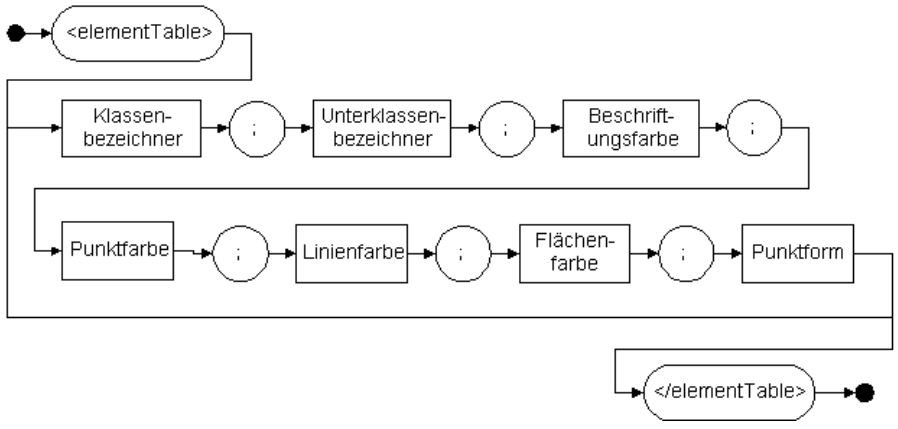

Abbildung 3.17: Syntaxdiagramm zur Definition des Layout-Stils der Figur.

Beispiel:

```
<elementTable>
    point; dragable;       black; red;   black; 0; smallCircle
    point; intersection;   black; blue;  black; 0; smallCircle
    point; functionDepend; black; blue;  black; 0; smallCircle
    point; fixed;          black; black; 0;     0; smallSquare
    line;  connect;        black; 0;     blue;  0;
    line;  straightLine;   black; 0;     black; 0;
[...]
</elementTable>
```

Der Klassenbezeichner und der Unterklassenbezeichner geben die Klasse des geometrischen Objekts an, dessen Layout-Stil durch die Variablen Beschriftungsfarbe, Punktfarbe, Linienfarbe, Flächenfarbe und Punktform festgelegt wird. Als Werte für die Farbvariablen sind Bezeichner für Farbkonstanten anzugeben. Die Punktform wird durch vordefinierte Formkonstanten bestimmt.

3.2.5 Der Betrachter

Als Betrachter wird die Softwarekomponente bezeichnet, mit der der Schüler einen Lernbaustein auf dem Bildschirm anschauen kann. Im Unterschied zu vielen DG-Systemen ist der Betrachter kein Konstruktionseditor, mit dem man Figuren konstruiert und speichert. Der Betrachter ist ein Laufzeitsystem, das mit *GeoScript* beschriebene Lernbausteine interpretieren und auf dem Bildschirm darstellen kann. Der Kern von *Geometria* besteht im wesentlichen aus den drei Klassen: `Geometria`, `Slate` und `Element` (Abbildung 3.2). Hinzu kommen noch eine Reihe von Unterklassen und Funktionsbibliotheken. Insgesamt sind alle zum Betrachter gehörenden Klassen in der Archiv-Datei[10] *Geometria.jar* zusammengefaßt.

Im folgenden will ich einen Überblick über die Funktionsweise des Betrachters geben. Dazu unterscheide ich zwischen vier Phasen: 1. Starten und Initialisieren, 2. Darstellen des Lernbausteins, 3. Realisation des Zugmodus und 4. Beenden des Betrachters. Zu jeder Phase skizziere ich grob, welche Aufgaben den Klassen `Geometria`, `Slate` und `Element` zukommen. Etwas ausführlicher werde ich die Realisation des Zugmodus behandeln.

Starten und Initialisieren (Klasse `Geometria`)

Um den Betrachter zu starten, muß dieser durch eine Web-Seite im HTML-Format aufgerufen werden (Beispiel auf Seite 91). Dazu benötigt man eine Java-Virtual-Machine, d. h. einen Interpreter, der Java-Programme ausführen kann. Eine solche Programmkomponente ist beispielsweise in allen gängigen Web-Browsern (z. B. Netscape Navigator oder Internet Explorer) enthalten, aber auch separat verfügbar.

Trifft ein Web-Browser auf die Definition des Java-Applets `Geometria` in einer Web-Seite, so wird der Betrachter gestartet. Dabei wird eine Instanz der Klasse `Geometria` erzeugt und deren `init`-Methode aufgerufen (Abbildung 3.18). Dieses geschieht im wesentlichen in vier Schritten:

[10] Java-Archive = Dateiname endet mit "jar".

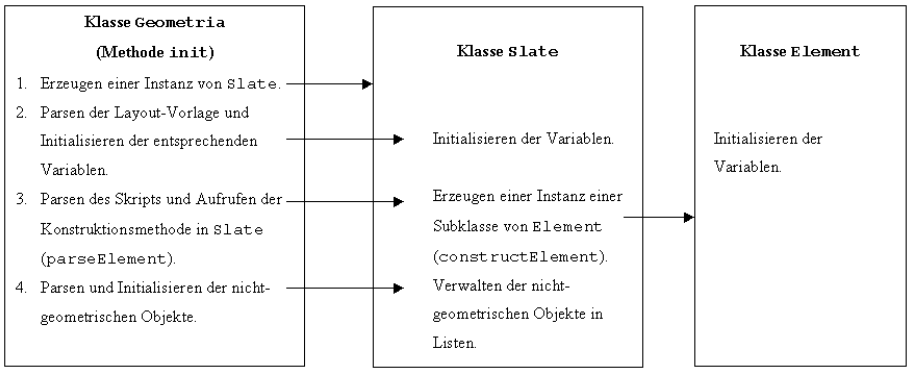

Abbildung 3.18: Starten und Initialisieren des Betrachters.

1. In der `init`-Methode von `Geometria` wird eine Instanz der Klasse `Slate` erzeugt, welche die Zeichenfläche eines Lernbausteins realisiert.

2. Im nächsten Schritt wird der Dateiname der Layout-Vorlage durch den HTML-Befehl `<PARAM>` eingelesen. Der Inhalt dieser Datei wird "geparst", d. h. die Textdatei wird Zeile für Zeile eingelesen und analysiert. Die in der Layout-Vorlage definierten Variablenwerte werden den entsprechenden Variablen der Klassen `Geometria` und `Slate` zugewiesen. Auf diese Weise übernimmt der Betrachter die globalen Voreinstellungen, die vom Figurenautor festgelegt wurden (Abschnitt 3.2.4).

3. Anschließend wird der Dateiname des Skripts eingelesen. Das Skript wird wiederum zeilenweise interpretiert. Die spezielle Funktion `parseElement` in der Klasse `Geometria` reicht die Daten jedes geometrischen Objekts an die Konstruktionsmethode `constructElement` in `Slate` weiter. Erst hier werden die geometrischen Objekte als Instanzen einer Unterklasse von `Element` erzeugt und ihre Beziehungen zueinander in Listen verwaltet (Abschnitt 3.2.3).

4. Analog zum vorigen Schritt werden die Daten der nicht-geometrischen Bestandteile eines Lernbausteins (Textfenster, Bilder, Hilfen, Antwortanalyse) eingelesen, geparst und in der Klasse `Slate` initialisiert und in Listen verwaltet.

Nachdem alle Informationen aus dem Skript und der Layout-Vorlage als Objekte erzeugt worden sind, ist die Initialisierungsphase des Objekts `Geometria` abgeschlossen. Der Lerninhalt wird auf dem Bildschirm angezeigt.

Darstellen des Lernbausteins (Klasse `Slate`)

Die sichtbaren Bestandteile eines Lernbausteins werden durch die `update`-Methode der Klasse `Slate` dargestellt. Darin wird das Verfahren der Doppelpufferung verwendet, das nach den folgenden drei Schritten abläuft (Abbildung 3.19):

1. Ein Bildobjekt `offscreen` wird während der Laufzeit im Arbeitsspeicher erzeugt, sofern es noch nicht vorhanden ist.

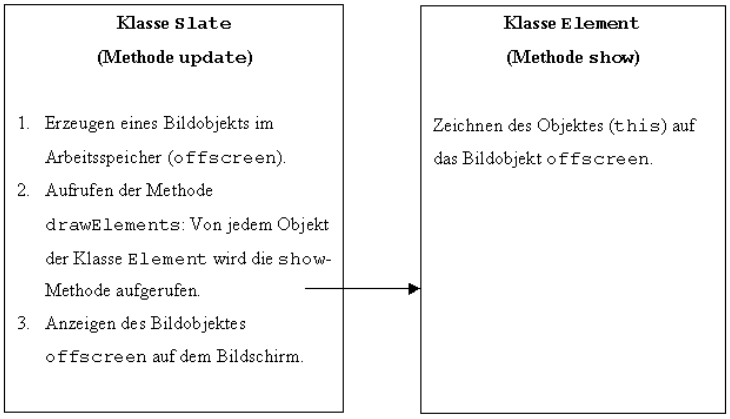

Abbildung 3.19: Darstellen des Lernbausteins.

2. Auf das Bildobjekt `offscreen` werden nun alle Objekte gezeichnet, die zu den sichtbaren Bestandteilen einer Figur gehören. Dies geschieht, indem die Methode `drawElements` aufgerufen wird. Diese Methode ruft wiederum für jedes erzeugte Objekt der Klasse `Element` die Methode `show` auf, in der das Objekt auf das Bildobjekt `offscreen` gezeichnet wird.

3. Zuletzt wird das Bildobjekt `offscreen` als Ganzes auf dem Bildschirm angezeigt.

Dieses Verfahren bietet im Unterschied zum direkten Zeichnen auf dem Bildschirm zwei Vorteile:

- Verschiebt der Schüler im Zugmodus ein Objekt, etwa eine Gerade, so muß beim direkten Zeichnen in einem Bildschirmfenster die "alte" Gerade gelöscht werden, bevor die "neue" Gerade gezeichnet werden kann. Praktisch muß dadurch jedes Objekt zweimal gezeichnet werden. Dieses senkt die Geschwindigkeit erheblich, vor allem wenn Kurvenobjekte aus den Klassen `CurveElement` oder `LocusElement` in der Figur verwendet werden, die aus vielen Punkten interpoliert sind. Durch die Methode der Doppelpufferung brauchen keine Objekte gelöscht werden, der Zeichenaufwand halbiert sich also.

- Nachdem ein geometrisches Objekt beim direkten Zeichnen gelöscht wurde und bevor es erneut gezeichnet wird, müssen seine aktuellen Objektdaten (z. B. die Koordinaten eines Punkts) neu berechnet werden. Je länger dieser Zeitabschnitt dauert, desto länger ist das Objekt auf der Zeichenfläche nicht sichtbar. Der Schüler erkennt dies daran, daß die Figur im Zugmodus flackert. Auch dieser unangenehme Nebeneffekt kann durch die Doppelpufferung vermieden werden.[11]

[11] Das Flackern kann beim direkten Zeichnen in einem Bildschirmfenster reduziert werden, indem man zuerst die aktuellen Daten zur Objektlage speichert und erst danach die Neuberechnung eines Objekts durchführt. Nun kann das Objekt an der alten Position gelöscht und sofort darauf neu gezeichnet werden. Die Vorgehensweise "Objekt löschen, neu berechnen,

Realisation des Zugmodus (Klasse `Slate`)

Die Variation im Zugmodus ist die wichtigste Form der Interaktion mit der geometrischen Figur. In der Klasse `Slate` wird der Zugmodus durch das von Mechling[12] beschriebene Verfahren der "doppelten Buchführung" realisiert. Dazu wird zu jedem geometrischen Objekt eine Liste aller seiner Elternobjekte (kurz: Eltern) und eine Liste aller seiner Kindobjekte (kurz: Kinder) bereitgestellt und geführt.

Wie diese Datenstruktur organisiert ist und was sie bedeutet, soll an einem Beispiel verdeutlicht werden. Gegeben sei eine Figur, die durch die folgende Konstruktionsbeschreibung erzeugt wird:

1. Punktobjekt `A` erzeugen (ziehbar innerhalb der Zeichenfläche)
2. Punktobjekt `B` erzeugen (ziehbar innerhalb der Zeichenfläche)
3. Punktobjekt `M` konstruieren (= Mittelpunkt von `A` und `B`)
4. Kreisobjekt `k` konstruieren (= Kreis mit Mittelpunkt `M` und Radius = `AM`, in der Abbildung 3.20 ohne Beschriftung)
5. Punktobjekt `C` erzeugen (ziehbar auf der Kreislinie)
6. Streckenobjekt `a` konstruieren (= Verbindungsstrecke zwischen `B` und `C`)
7. Streckenobjekt `b` konstruieren (= Verbindungsstrecke zwischen `A` und `C`)
8. Streckenobjekt `c` konstruieren (= Verbindungsstrecke zwischen `A` und `B`)

Die zugehörige Figur ist in der Abbildung 3.20 zu sehen. Den Abhängigkeitsgraphen der einzelnen Objekte dieser Konstruktion zeigt die Abbildung 3.21. Hierin sind die Eltern-Kind-Beziehungen dargestellt. Jedes Objekt, von dem ein Pfeil ausgeht, ist Elter des Objekts, auf das der Pfeil zeigt. Und umgekehrt: Jedes Objekt auf das ein Pfeil zeigt, ist Kind des Objekts, von dem der Pfeil ausgeht. Jedes Objekt speichert, wenn es erzeugt wird, in einer Liste alle sei-

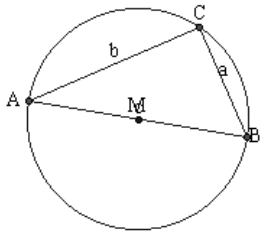

Abbildung 3.20: Beispielfigur.

ne Elternobjekte. Außerdem meldet sich jedes neu erzeugte Objekt bei seinen

zeichnen" wird also geändert in "Daten speichern, Objekt neu berechnen, löschen, zeichnen". Dieses erfordert einen etwas höheren Programmieraufwand, verringert aber deutlich das Flackern der Figur.

[12] Mechling 1997, S. 118

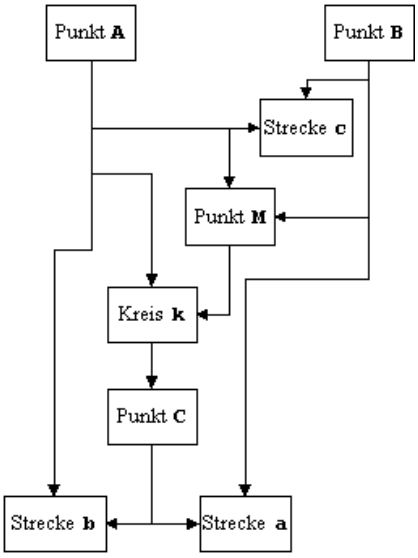

Abbildung 3.21: Abhängigkeitsgraph der Beispielkonstruktion.

Eltern als Kind an und wird in die entsprechende Liste mit aufgenommen. Mit Hilfe dieser Datenstruktur wird der Zugmodus nun wie folgt realisiert:[13]

1. Jedes zu ziehende Objekt vermerkt in allen seinen Kindobjekten, daß diese ebenfalls gezogen werden müssen. (Dazu wird bei jedem Objekt die boolesche Variable `marked` auf `true` gesetzt.)

2. Jedes vom Zugvorgang betroffene Objekt aktualisiert sich genau dann, wenn alle seine Elternobjekte sich schon aktualisiert haben. (Ein Objekt wird aktualisiert, indem die zugehörige `update`-Methode aufgerufen wird.)

Ein Beispiel soll dies verdeutlichen. Ausgangsbasis ist die Figur aus der Abbildung 3.20. Der Punkt `A` wird nun beispielsweise vom Schüler mit der Computermaus gezogen. Der Algorithmus beginnt mit dem ersten Schritt:

1. Der Punkt `A` vermerkt bei seinen Kindern (Punkt `M`, Kreis `k` und Strecken `b` und `c`), daß diese ebenfalls gezogen werden müssen. Die Variable `marked` wird jeweils auf `true` gesetzt.

2. Der Punkt `M` vermerkt bei seinem Kind (Kreis `k`), daß dieses ebenfalls gezogen werden muß.

3. Der Kreis `k` vermerkt bei seinem Kind (Punkt `C`), daß dieses ebenfalls gezogen werden muß.

4. Der Punkt `C` vermerkt bei seinen Kindern (Strecke `a` und `b`), daß diese ebenfalls gezogen werden müssen.

Ist der erste Schritt beendet, d. h. alle betroffenen Objekte sind markiert, dann wird der zweite Schritt ausgeführt:

[13] Mechling 1997, S. 118

1. Der Punkt `A` erhält die neuen Koordinaten zugewiesen und seine Markierung wird aufgehoben (`marked = false`).
2. Wenn die Punkte `A` und `B` nicht mehr markiert sind, dann wird der Punkt `M` aktualisiert.
3. Wenn die Punkte `A` und `M` nicht mehr markiert sind, dann wird der Kreis `k` aktualisiert.
4. Wenn der Kreis `k` aktualisiert ist, dann wird der Punkt `C` neu berechnet.
5. Wenn die Punkte `A` und `B` nicht mehr markiert sind, dann wird die Strecke `c` aktualisiert.
6. Wenn die Punkte `B` und `C` nicht mehr markiert sind, dann wird die Strecke `a` aktualisiert.
7. Wenn die Punkte `A` und `C` nicht mehr markiert sind, dann wird die Strecke `b` aktualisiert.

Nachdem beide Arbeitsschritte durchgeführt worden sind, wird die Figur mit den aktualisierten Daten neu gezeichnet. Die Datenstruktur der Eltern-Kind-Liste gewährleistet, daß nur diejenigen geometrischen Objekte neu berechnet werden, die auch wirklich vom Zugmodus betroffen sind.

Es soll angemerkt werden, daß der Zugmodus nicht unbedingt durch eine solche Listenstruktur realisiert werden muß. Ausreichend ist auch eine einfach verkettete Liste, die in jedem Glied ein geometrisches Objekt speichert. Dieses entspricht der linearen Konstruktionsbeschreibung, in der die Konstruktionsbefehle nacheinander aufgeführt werden. Dabei ist offensichtlich, daß die Konstruktion neuer Objekte sich nur auf schon zuvor definierte Objekte beziehen kann. Ausreichend wäre es, nur alle diejenigen Objekte zu aktualisieren, die nach dem gezogenen Objekt konstruiert worden sind. Dabei ist lediglich die Reihenfolge zu beachten. Es müßten dann nicht zwei Listen geführt und verwaltet werden. Die Struktur des Programms wäre einfacher. Das *Geometry-Applet* von Joyce arbeitet auf diese Weise. Eine Eltern-Kind-Listenstruktur ist jedoch besser, denn sie erzielt eine höhere Geschwindigkeit im Zugmodus. Bei einer linearen Liste müssen nämlich auch Objekte aktualisiert werden, deren Position durch das Verschieben im Zugmodus gar nicht verändert wurde. Das ist überflüssig und wirkt sich negativ auf die Geschwindigkeit im Zugmodus aus, insbesondere wenn es sich um sehr aufwendig zu berechnende Objekte (wie Kurven oder Ortslinien) handelt.

Beenden des Betrachters (Klasse `Geometria`)

Das Beenden des Betrachters kann auf zwei Arten erfolgen:

1. durch Schließen des Betrachterfensters, falls der Betrachter in einem separaten Fenster ausgeführt wird, oder
2. durch das Aufrufen einer neuen Web-Seite mit dem Browser, falls der Betrachter direkt in die Web-Seite eingebettet ist.

In beiden Fällen sorgt der Java-Interpreter automatisch dafür, daß die belegten Speicherressourcen wieder freigegeben werden.

3.3 Die Klassenhierarchie der geometrischen Objekte

In diesem Abschnitt wird die Klassenhierarchie der geometrischen Objekte in *Geometria* beschrieben. Dabei ist jeder geometrische Objekttyp (Punkt, Gerade, Kreis, usw.) softwaretechnisch durch eine spezielle Klasse realisiert. Die einzelnen Klassen sind hierarchisch angeordnet. Die tiefer stehenden Klassen erben die Variablen und Methoden der oberen Klassen. Wie diese Hierarchie im einzelnen aufgebaut ist und aus welchen Variablen und Methoden die Klassen bestehen, wird in den folgenden Abschnitten dargelegt. Dabei wird sich herausstellen, daß sich geometrische Objekte mit idealen Eigenschaften nicht ohne weiteres durch eine interaktive Software nachbilden lassen. Denn der Zugmodus besitzt keine direkte Entsprechung in der Zeichenblatt-Geometrie. Daher muß der Entwickler einer Geometriesoftware immer wieder Entscheidungen treffen, wie sich bestimmte Objekte im Zugmodus verhalten sollen. Er muß Antworten auf Fragen finden, die in der Zeichenblatt-Geometrie keine Rolle spielen. Ich werde im folgenden an den entsprechenden Stellen darauf eingehen.

3.3.1 Übersicht über die Klassenhierarchie

Die Klassenhierarchie der geometrischen Objekte besteht aus der abstrakten Klasse `Element` und neun davon abgeleiteten Unterklassen (Abbildung 3.22). In den Unterklassen wird jeweils das geometrische Objekt realisiert, das in Klammern unter dem Klassenbezeichner angegeben ist. Die oberste Wurzelklasse `Element` repräsentiert kein geometrisches Objekt, sondern umfaßt Variablen und Methoden, die in allen abgeleiteten Klassen verfügbar sind. Dazu zählen Variablen für den Objektbezeichner, die Objektfarben und -formen sowie Funktionen, zum Umrechnen von Weltkoordinaten in Fensterkoordinaten. Außerdem ist in dieser Klasse jeweils eine Liste für die Elternobjekte (`Parents`) und Kindobjekte (`Childs`) eines Objekts definiert.[14] Die Anzahl der Listenelemente wird in den Variablen `numParents` und `numChilds` gespeichert. Die Methode `addParent` erweitert die `Parents`-Liste um ein Objekt. Für den Zugmodus sind insbesondere die Methoden `markChilds` und `updateChilds` erforderlich. Die darin verwendete boolesche Variable `marked` dient zur Markierung eines Objekts. Im folgenden wird der kommentierte Quellcode[15] dieser Methoden auszugsweise wiedergegeben.

```
protected void addParent(Element P) {
    // Prüfe, ob das Objekt P definiert ist.
    if (P == null) {
        return;
    } // if

    // Prüfe, ob die Liste Parents das Objekt P bereits enthält.
```

[14] Die Listen der Eltern- und Kindobjekte dienen dazu, um den Zugmodus zu realisieren, dessen Funktionsweise bereits in Abschnitt 3.2.5 beschrieben wurde.

[15] Alle Quelltexte werden im folgenden so dargestellt, daß nur die wesentlichen Befehle und Algorithmen wiedergegeben werden. Alle Methoden und Variablen, die lediglich zur Darstellung von Objekten auf der Zeichenfläche erforderlich sind, werde ich zugunsten einer besseren Lesbarkeit weglassen.

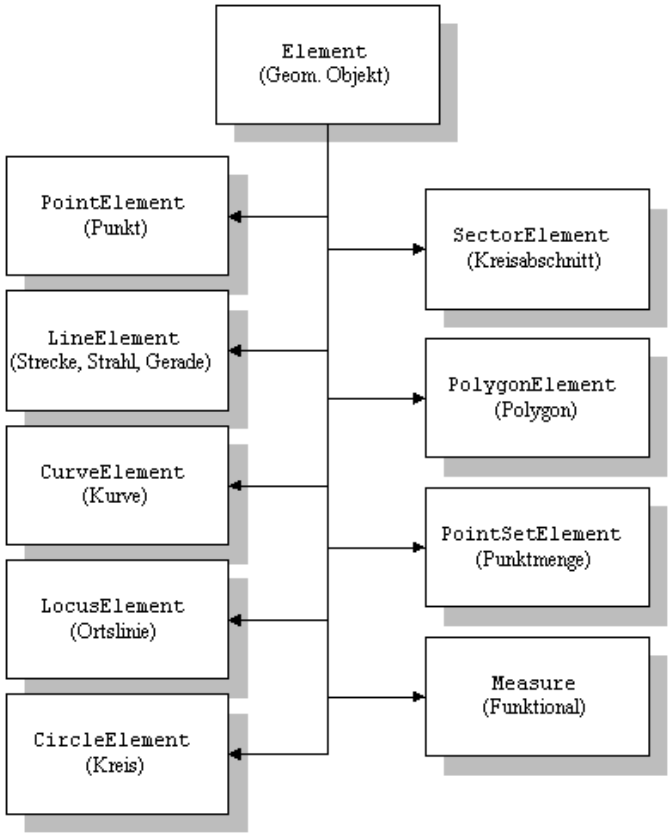

Abbildung 3.22: Die Klassenhierarchie der Klasse `Element`.

```
    // Falls nicht: Füge es zur Liste hinzu,
    // inkrementiere den Zähler numParents und
    // erweitere die Childs-Liste um das aktuelle Objekt.
    if (!Parents.contains(P)) {
         Parents.addElement(P);
         numParents++;
         P.addChild((Element) this);
    } // if
}

protected void markChilds() {
    // Prüfe, ob das aktuelle Objekt markiert ist.
    // Falls nicht, dann markiere es und rufe von
    // jedem Kindobjekt die Methode markChilds() auf,
    // wenn es noch nicht markiert ist.
    if (!this.marked) {
        this.marked = true;
        for (int i=0;i<Childs.size();i++) {
            if (!((Element) Childs.elementAt(i)).marked) {
                ((Element) Childs.elementAt(i)).markChilds();
            } // if
        } // for
    } // if
}

protected void updateChilds() {
    // Prüfe, ob das aktuelle Objekt markiert ist.
    if (marked) {
        boolean ok = true;
        // Prüfe, ob für alle Elternobjekte die Markierung auf-
        // gehoben ist (= alle Elternobjekte sind aktualisiert).
        for (int i=0;i<Parents.size();i++) {
            if (((Element) Parents.elementAt(i)).marked) {
                ok = false;
            } // if
        } // for

        // Falls alle Elternobjekte aktualisiert worden sind,
        // aktualisiere das aktuelle Objekt, lösche die
        // Markierung und rufe die Methode updateChilds
        // für alle Kindobjekte auf.
        if (ok) {
            this.update();
            this.marked = false;
            for (int i=0;i<Childs.size();i++) {
                ((Element) Childs.elementAt(i)).updateChilds();
            } // for
        } // if
    } // if
}
```

Der Aufbau aller Unterklassen von `Element` ist ähnlich und besteht im wesentlichen jeweils aus den folgenden drei Methoden:

1. Eine Konstruktor-Methode, die automatisch aufgerufen wird, sobald man eine neue Instanz eines Objekts erzeugt. Darin werden sämtliche Objektparameter übergeben und die Objektvariablen initialisiert.

2. Eine Aktualisierungsmethode `update`, die immer dann aufgerufen wird, wenn ein Objekt selbst oder ein Objekt aus der Liste seiner Elternobjekte verändert wurde. Die Methode sorgt dafür, daß die Lage des Objekts im Koordinatensystem neu berechnet wird.

3. Eine Darstellungsmethode `show`, die das Objekt auf die Zeichenfläche zeichnet. Die Darstellung erfolgt dabei entsprechend den Werten der Farb- und Formvariablen des Objekts.

Auch Funktionale der Klasse `Measure` werden quasi als geometrische Objekte behandelt. Dieses ist dadurch begründet, daß die Klasse `Measure` ebenfalls jeweils eine Konstruktor-, eine `update`- und eine `show`-Methode umfaßt. Objekte dieser Klasse können ebenso als Eingangsgrößen für die Konstruktion geometrischer Objekte dienen wie beispielsweise Punktobjekte.

In den folgenden Abschnitten wird der Aufbau der Unterklassen von `Element` dargestellt. Alternative Konzeptionen von Klassenhierarchien wurden beschrieben von: Kadunz[16] (*Thales*), Mechling[17] (*Euklid*), Kortenkamp[18] (*Cinderella*) und Grothmann[19] (*Circle and Ruler*). Rückschlüsse auf die Klassenhierarchie von *JavaSketchpad* lassen sich aus der Dokumentation von Jackiw[20] ziehen.

3.3.2 `PointElement`

Die Frage "Was ist ein Punkt?" stellt sich in der Geometrie mit Zirkel und Lineal eigentlich nicht. Die intuitive Vorstellung von einem Punkt ist in der Regel ausreichend, um geometrische Konstruktionen durchführen oder Aufgaben lösen zu können. Wenn man ein Geometrie-System entwickeln möchte, muß man allerdings eine Antwort auf diese Frage finden, um die geometrischen Objekte softwaretechnisch nachbilden zu können. Generell kann man zwischen zwei Arten von Punkten differenzieren, die realisiert werden müssen, wenn man eine im Zugmodus bewegliche Figur erhalten will: ziehbare und nicht-ziehbare Punkte.

Die nicht-ziehbaren Punkte sind diejenigen Punkte, die nicht mit der Computermaus bewegt werden können, weil deren Koordinaten durch eine bestimmte Abbildungsvorschrift f festgelegt sind. Der Definitionsbereich von f ist eine Menge M von geometrischen Objekten und es gilt: $f\colon M \to \mathbb{R}^2$. Der Bildbereich $\mathbb{R}^2$ umfaßt alle Koordinatenpaare des nicht-ziehbaren Punktobjekts. Beispielsweise liefert die partielle Abbildung $f(g_1, g_2)$ den Schnittpunkt S zweier Geraden g_1, g_2, falls $S \in g_1 \wedge S \in g_2 \wedge g_1 \nparallel g_2 \wedge g_1 \neq g_2$ ist.

Im Unterschied zu den nicht-ziehbaren Punkten haben die ziehbaren Punkte keine direkte Entsprechung in der Zeichenblatt-Geometrie. Sie entsprechen am

[16] Kadunz 1996, S. 58
[17] Mechling 1997, S. 116
[18] Kortenkamp 1999
[19] Grothmann 1999, http://mathsrv.ku-eichstaett.de/MGF/homes/grothmann/zul.html
[20] Jackiw, http://forum.swarthmore.edu/dynamic/java_gsp/jsp_home.htm

ehesten den Punkten, deren Lage am Anfang einer Konstruktion auf dem Zeichenblatt beliebig gewählt werden kann. Alle nachfolgenden Konstruktionen beziehen sich dann auf diese (festen) Punkte. Bei einem Geometrie-System können diese Punkte nun mit der Computermaus im Zugmodus gezogen werden, wobei die konstruktiv festgelegten Relationen zwischen den davon abhängigen Objekten erhalten bleiben.

Die Klasse `PointElement` selbst umfaßt keine speziellen Punktobjekte, sondern stellt für die abgeleiteten Klassen Variablen und Methoden zur Verfügung. Dazu zählen:

- die numerischen Variablen `x` und `y`, zum Speichern der Koordinaten eines Punkts,
- die boolesche Variable `dragable`, zum Unterscheiden, ob ein Punkt ziehbar ist oder nicht,
- die Variable `layoutStyle`, zum Festlegen einer Darstellungsform für ein Punktobjekt,
- die Funktion `getProperty`, die Zugriff auf die Koordinaten und ggf. auf einen Kurvenparameter ermöglicht,
- die booleschwertige Funktion `defined`, die prüft, ob die Variablen `x` und `y` reelle Werte enthalten,
- die Methode `show`, zum Darstellen eines Punkts durch ein Zeichen auf der Zeichenfläche sowie
- eine Anzahl von Funktionen (wie `product`, `difference`, `sum`, usw.), mit denen man analytische Operationen auf Punktobjekte anwenden kann.

Alle speziellen Punktobjekte werden durch Unterklassen von `PointElement` definiert. Die Abbildung 3.23 zeigt einen Ausschnitt aus der Klassenhierarchie. Weil aus Platzgründen nicht alle Unterklassen von `PointElement` beschrieben werden können, soll exemplarisch der Aufbau der vier Klassen `Dragable`, `Intersection`, `Rotation` und `FunctionDepend` betrachtet werden.

Die Klasse `Dragable`

Alle ziehbaren Punktobjekte werden durch die Klasse `Dragable` implementiert. Abhängig von der Punktmenge, in der ein Punktobjekt bewegt werden kann, lassen sich verschiedene Typen von ziehbaren Punktobjekten unterscheiden. Die Punktmenge bezeichne ich als Definitionsbereich D eines ziehbaren Punktobjekts. Wenn D durch ein geometrisches Objekt gegeben ist, so wird dieses auch als Bezugsobjekt bezeichnet. Im einfachsten Fall läßt sich ein Punktobjekt innerhalb der gesamten Zeichenfläche Z verschieben, dann ist $D = Z$. Der Definitionsbereich kann aber auch auf die Fläche eines Polygons, auf eine Kreislinie oder auf eine Kurve eingeschränkt sein.

Weiterhin ist zu fragen, ob der Definitionsbereich selbst im Zugmodus veränderbar oder nicht veränderbar ist. Dadurch können verschiedene Typen von ziehbaren Punktobjekten unterteilt werden, die in der Tabelle 3.1 aufgelistet sind. Die wichtigsten Methoden in der Klasse `Dragable` sind: die Konstruktor-, die `update`- und die `drag`-Methode. Innerhalb der Konstruktor-Methode werden

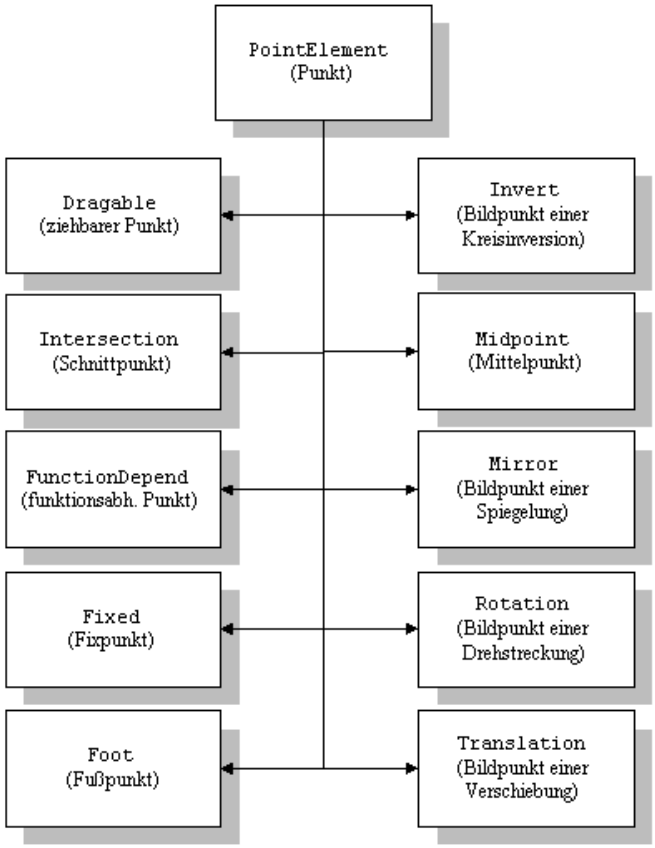

Abbildung 3.23: Ausschnitt aus der Klassenstruktur von `PointElement`.

Tabelle 3.1: Typen von ziehbaren Punktobjekten.

`type`	$D =$	D ist ...	Klasse des Bezugsobjekts
1	ges. Zeichenfläche	nicht veränderbar	—
2	Gerade	veränderbar	`StraightLine`
3	Strahl	veränderbar	`Ray`
4	Strecke	veränderbar	`LineElement`
5	Kreislinie	veränderbar	`CircleElement`
6	Kreisfläche	veränderbar	`CircleElement`
7	Polygon-Kantenzug	veränderbar	`PolygonElement`
8	Polygonfläche	veränderbar	`PolygonElement`
9	Kurve	veränderbar	`CurveElement`
10	Ortslinie	veränderbar	`LocusElement`
11	Punktmenge	nicht veränderbar	`PointSetElement`

dem Punktobjekt die Anfangskoordinaten zugewiesen. Es wird das Bezugsobjekt übergeben, welches den Definitionsbereich festlegt und in der Liste der Elternobjekte gespeichert ist. Zu jedem Definitionsbereich gibt es eine separate Konstruktor-Methode. Die Variable `type` dient zur Unterscheidung dieser Untertypen. Die Konstruktor-Methode für einen auf einem Kreis ziehbaren Punkt sieht beispielsweise wie folgt aus:

```
Dragable(double xval, double yval, CircleElement cval) {
    type = 5;
    dragable = true;
    circle = cval;
    addParent(circle);
    x = xval;
    y = yval;
}
```

Die Methode `drag` wird immer dann aufgerufen, wenn der Schüler einen Punkt dieser Klasse mit der Computermaus anklickt und zieht. Dabei werden die aktuellen Koordinaten des Mauszeigers übergeben und den Variablen `x` und `y` zugewiesen. Die `update`-Methode sichert anschließend, daß ein Punktobjekt nicht aus seinem Definitionsbereich herausbewegt werden kann. Die Methode ist wie folgt aufgebaut:

```
protected boolean drag(double tox, double toy) {
    x = tox;
    y = toy;
    update();
    return true;
}
```

Durch die `update`-Methode werden die Koordinaten eines Punkts so bestimmt, daß dieser stets innerhalb seines Definitionsbereichs liegt. Die `update`-Methode wird aufgerufen, wenn die Koordinaten des Punktobjekts in der `drag`-Methode verändert wurden (1. Fall) oder wenn sich das Bezugsobjekt verändert hat (2. Fall). Je nach Punkttyp wird innerhalb der `update`-Methode verzweigt und es werden unterschiedliche Berechnungsverfahren aufgerufen. Ziel ist es in jedem Fall, den Punkt so zu verschieben, daß er (wieder) innerhalb des Definitionsbereichs liegt. Dabei wird in den implementierten Algorithmen folgende Minimalitätsforderung berücksichtigt: Der Abstand zwischen Ausgangslage und Ziellage des Punktobjekts soll so gering wie möglich sein. Die Methode `update` ist wie folgt aufgebaut:

```
protected void update() {
    switch (type) {
        case 1:
            return;
        case 2:
            toLine(A, B, false);
            return;
        case 3:
            ray.pointToRay(this);
            return;
```

```
        case 4:
            toLine(A, B, true);
            return;
        case 5:
            toCircle(circle);
            return;
        case 6:
            if (!circle.contains(this)) {
                toCircle(circle);
            } // if
            return;
        case 7:
            toPolygon(polygon);
            return;
        case 8:
            if (!polygon.contains(this)) {
                toPolygon(polygon);
            } // if
            return;
        case 9:
            curve.pointToCurve(this, t);
            return;
        case 10:
            locus.pointToCurve(this);
            return;
        case 11:
            pointSet.pointToPointSet(this);
            return;
    } // switch
}
```

Exemplarisch soll für den Fall 5 (D = Kreislinie) die Methode `toCircle` näher betrachtet werden. Darin wird das aktuelle Punktobjekt (`this`) zu dem am nächsten gelegenen Punkt auf dem Kreis bewegt.

```
PointElement toCircle(CircleElement C) {
    double factor = C.radius() / distance(C.Center);
    this.x = C.Center.x + factor*(x - C.Center.x);
    this.y = C.Center.y + factor*(y - C.Center.y);
    return this;
}
```

Wie wirken sich nun die oben beschriebenen Methoden auf die Variation einer Figur im Zugmodus aus? Wird ein Punkt, der an ein Objekt gebunden ist, angeklickt und gezogen, entsteht für den Schüler der Eindruck, als würde er den Punkt an einem Gummiband ziehen. Durch die Minimalitätsforderung ist der Abstand zwischen Mauszeiger und Punkt stets minimal. Ein solches Figurenverhalten kann als leicht nachvollziehbar bezeichnet werden. Wird nun anstatt des ziehbaren Punkts das Bezugsobjekt variiert, so entsteht für den Schüler der Eindruck, als würde der Punkt ein gewisses Trägheitsverhalten besitzen. Unabhängig davon, in welche Lage der Schüler ein Bezugsobjekt bringt, verschiebt sich der ziehbare Punkt so wenig wie möglich.

Andere Geometrie-Systeme aktualisieren diesen Fall nach einem anderen Prinzip. Konstruiert man beispielsweise mit *Cabri-Géomètre* einen ziehbaren Punkt Z auf einer Strecke AB und verschiebt diese, dann wird Z so mitbewegt, daß das Teilverhältnis der drei Punkte konstant bleibt.

Ein merkwürdiges Verhalten zeigt *Sketchpad*. Es sei ein Fünfeck $ABCDE$ gegeben, auf dessen Kantenzug ein Punkt Z bewegt werden kann. Es sei weiter angenommen, Z liege auf der Seite AB. Verschiebt man nun die Punkte C, D oder E, so wird man überrascht feststellen, daß sich auch Z bewegt. Nach welcher Bewegungsvorschrift dies geschieht, ist jedoch nicht zu erkennen.

Ein Systemverhalten, wie in den beiden Beispielen aufgeführt, erscheint geometrisch unbegründet und erschwert das Verständnis des Zugmodus. Ein Punktobjekt, wie im *Cabri*-Beispiel, hat eine doppelte Bedeutung und zeigt ein ambivalentes Verhalten: Einerseits ist der Punkt ziehbar und kann verschoben werden, andererseits verhält sich der Punkt, als wäre er nicht-ziehbar und nach einer bestimmten Vorschrift konstruiert, die das Teilverhältnis zu den Streckenendpunkten konstant hält. Diese Ambivalenz erfordert vom Schüler eine zusätzliche Differenzierung zwischen den ohnehin schon vielfältigen Bedeutungen von Punkten in einem DG-System. Man sollte ihm dieses nicht auch noch zumuten. Ein Beispiel, in dem sich dieser Sachverhalt negativ auf das Lösen einer Konstruktionsaufgabe auswirkt, ist von Hölzl[21] beschrieben worden.

Die Klasse `Intersection`

Die Schnittpunkte zwischen Geraden und Kreisen in der euklidischen Geometrie werden durch die Klasse `Intersection` realisiert. Die Umsetzung eines Schnittpunkts der euklidischen Geometrie durch ein Objekt erfordert einige Zusatzüberlegungen.

Zwischen zwei Geraden, zwei Kreisen oder einem Kreis und einer Gerade gibt es jeweils 0, 1 oder 2 Schnittpunkte. Diese drei Fälle muß das Geometrie-System unterscheiden. Die Klasse `Intersection` wurde deshalb wie folgt realisiert: Je nachdem, ob zwei Geraden, eine Gerade und ein Kreis oder zwei Kreise miteinander geschnitten werden, gibt es drei Konstruktor-Methoden. Diese Fälle werden durch den Wert der Variable `type` unterschieden. In den beiden Fällen, in denen es maximal zwei Schnittpunkte geben kann, müssen grundsätzlich zwei Punktobjekte erzeugt werden, die durch die Variable `num` (`num = 1` und `num = 2`) auseinander gehalten werden. Berühren sich nun etwa zwei Kreise in einem Punkt, so werden den beiden Punktobjekten dieselben x- und y-Koordinaten zugewiesen. Existiert kein Schnittpunkt, so werden die Variablen `x` und `y` auf den Wert `infinity` gesetzt. Im folgenden sind die drei Konstruktor-Methoden aufgeführt:

```
// Intersection StraightLine-StraightLine
Intersection(PointElement gA, PointElement gB,
             PointElement hA, PointElement hB) {
    A = gA;
    B = gB;
    C = hA;
    D = hB;
    addParent(A, B, C, D);
```

[21] Hölzl 1994, S. 80f

```
    type = 1;
}

// Intersection StraightLine-Circle
Intersection(LineElement g, CircleElement Cval, int i) {
    num = i;
    G1 = g.A;
    G2 = g.B;
    Circle = Cval;
    addParent(G1, G2, Circle);
    type = 2;
}

// Intersection Circle-Circle
Intersection(CircleElement C1val, CircleElement C2val, int i) {
    num = i;
    Circle1 = C1val;
    Circle2 = C2val;
    addParent(Circle1, Circle2);
    type = 3;
}
```

Die Koordinaten des Schnittpunktobjekts werden in der Methode `update` berechnet. Diese wird immer dann aufgerufen, wenn wenigstens ein Elternobjekt seine Lage in der Zeichenfläche verändert hat. Abhängig vom Wert der Variable `type` wird eine von drei speziellen Berechnungsmethoden aufgerufen, welche die Schnittpunktkoordinaten analytisch bestimmt.

```
protected void update() {
    switch (type) {
        case 1:
            this.toIntersection(A, B, C, D);
            return;
        case 2:
            this.toIntersectionCircleLine(Circle1, G1, G2, num);
            return;
        case 3:
            this.toIntersectionCircleCircle(Circle1, Circle2, num);
            return;
    } // switch
}
```

Die Methoden `toIntersection`, `toIntersectionCircleLine` und `toIntersectionCircleCircle` liefern die Schnittpunktkoordinaten, wobei mit Hilfe der Variablen `num` die Lösungen der quadratischen Gleichung unterschieden werden. Der Nachteil dieser auf reellen Koordinaten beruhenden Berechnungen besteht darin, daß bei der Variation im Zugmodus die Schnittpunkte in bestimmten Situationen plötzlich springen; sie verhalten sich nicht stets kontinuierlich. Dieses Problem wurde eingehend von Kortenkamp[22] untersucht. Das von Kortenkamp

[22] Kortenkamp 1999

und Gebert entwickelte Geometrie-System *Cinderella* arbeitet deshalb mit komplexen Zahlen und einem projektiven Geometriemodell, das diese Unzulänglichkeit vermeidet.

Die Klasse `Rotation`

Die Klasse `Rotation` realisiert Punktobjekte, die den Bildpunkten einer Drehstreckung in der Abbildungsgeometrie entsprechen. Je nachdem, ob der Drehwinkel `alpha` und der Streckfaktor `scale` variabel oder konstant sein sollen, sind drei Konstruktor-Methoden vorhanden. Innerhalb dieser wird als Urbild und Drehzentrum jeweils ein Punktobjekt an die Variablen `P` und `Z` übergeben:

1. Die beiden Variablen `alpha` und `scale` beinhalten konstante numerische Werte (`type = 1`).
2. Der Winkel `alpha` wird durch drei im Zugmodus veränderbare Punktobjekte `A`, `B`, `C` definiert, während die Variable `scale` konstant bleibt (`type = 2`).
3. Die Winkelgröße und der Skalierungsfaktor werden durch zwei Funktionale als Instanzen der Klasse `Measure` definiert (`type = 3`).

Die entsprechenden Konstruktor-Methoden lauten:

```
Rotation(PointElement Pval, PointElement Zval,
         double a, double s) {
    alpha = a;
    scale = s;
    Z = Zval;
    P = Pval;
    addParent(P, Z);
    type = 1;
}

Rotation(PointElement Pval, PointElement Zval,
         PointElement Aval, PointElement Bval,
         PointElement Cval, double s) {
    A = Aval;
    B = Bval;
    C = Cval;
    scale = s;
    Z = Zval;
    P = Pval;
    addParent(A, B, C, Z, P);
    type = 2;
}

Rotation(PointElement Pval, PointElement Zval,
         Measure m0, Measure m1) {
    Z = Zval;
    P = Pval;
    M0 = m0;
```

```
    M1 = m1;
    addParent(P, Z, M0, M1);
    type = 3;
}
```

Die Methode `update` sorgt dafür, daß die Werte für `scale` und `angle` ggf. neu berechnet werden. Zur Koordinatenberechnung wird die Methode `rotate` aufgerufen. Diese berechnet die Koordinaten des Bildpunkts bei einer Drehstreckung von `P` um das Drehzentrum `Z` mit dem Winkel `alpha` und dem Streckfaktor `scale`. Die Koordinaten werden dem aufrufenden Punktobjekt (`this`) zugewiesen.

```
protected void update() {
    switch (type) {
        case 1:
            this.rotate(P, Z, scale*Math.cos(alpha),
                              scale*Math.sin(alpha));
            break;
        case 2:
            alpha = B.angle2D(A, C);
            this.rotate(P, Z, scale*Math.cos(alpha),
                              scale*Math.sin(alpha));
            break;
        case 3:
            alpha = M0.getValue();
            scale = M1.getValue();
            this.rotate(P, Z, scale*Math.cos(alpha),
                              scale*Math.sin(alpha));
            break;
    } // switch
}
```

Die Klasse `FunctionDepend`

Die Klasse `FunctionDepend` zeichnet sich dadurch aus, daß darin kein spezieller Punkttyp der Geometrie umgesetzt wird. Stattdessen realisiert die Klasse ein allgemeines Punktobjekt. Erst durch die Angabe von zwei Koordinatenfunktionen `Mx` und `My` ist eine Abbildungsvorschrift definiert, durch welche die Koordinaten des Punktobjekts bestimmt sind. Die beiden Koordinatenfunktionen werden durch Objekte der Klasse `Measure` realisiert und können damit beliebige Funktionen darstellen (Abschnitt 3.3.10). Die zugehörige Konstruktor-Methode sieht wie folgt aus:

```
FunctionDepend(Measure mxval, Measure myval) {
    Mx = mxval;
    My = myval;
    addParent(Mx, My);
}
```

Ändert sich der Wert eines der beiden Elternobjekte, so wird die Methode `update` ausgeführt. Durch Aufrufen der Funktion `getValue`, die den Wert eines Funktionals liefert, werden den Variablen `x` und `y` die aktuellen Koordinaten

zugewiesen. Die Funktionen `X_WorldToWindow` und `Y_WorldToWindow` rechnen dabei die Weltkoordinaten in Fensterkoordinaten um.

```
protected void update() {
    x = X_WorldToWindow(Mx.getValue());
    y = Y_WorldToWindow(My.getValue());
}
```

Mit einem solchen Punktobjekt hat der Figurenautor weitreichende Möglichkeiten, beliebige Abbildungsvorschriften für Punkte zu realisieren und entsprechende Objekte zu erzeugen. Genau genommen wäre es für ein Geometrie-System ausreichend, wenn nur Punktobjekte der beiden Klassen `Dragable` und `FunctionDepend` implementiert wären. Die Menge der darstellbaren Punkte würde dadurch nicht verkleinert. Dennoch ist es sinnvoll, spezielle Klassen für spezielle Punktobjekte zu definieren, weil durch optimierte Berechnungsmethoden eine höhere Geschwindigkeit im Zugmodus erreicht wird. Außerdem arbeitet der Figurenautor ökonomischer, wenn er auf eine Reihe häufig verwendeter, bereits vordefinierter Punktobjekte zugreifen kann.

3.3.3 `LineElement`

Mit der Klasse `LineElement` und den beiden davon abgeleiteten Klassen `Ray` und `StraightLine` werden die geometrischen Objekte Strecke, Strahl und Gerade realisiert (Abbildung 3.24). Die Lage im Koordinatensystem wird durch die Angabe von zwei Punktobjekten `A` und `B` festgelegt. Die drei Klassen bestehen

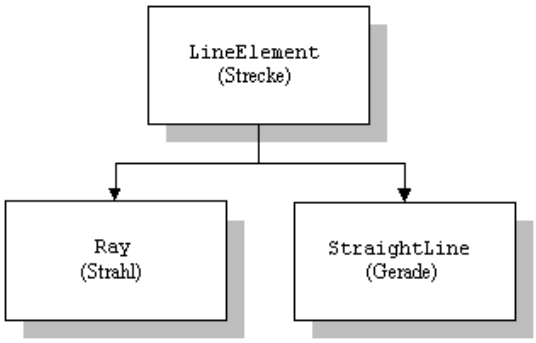

Abbildung 3.24: Die Klassenstruktur von `LineElement`.

jeweils aus einer Konstruktor-Methode und einer `show`-Methode. Eine `update`-Methode ist nicht erforderlich, da die Elternobjekte `A` und `B` die Lage vollständig definieren. Die Konstruktor-Methode ist in allen drei Klassen gleich aufgebaut, darin werden lediglich die beiden Punktobjekte übergeben und in die Liste der Elternobjekte aufgenommen.

```
LineElement(PointElement Aval, PointElement Bval) {
    A = Aval;
    B = Bval;
    addParent(A, B);
}
```

Das Zeichnen eines solchen Objekts auf der Zeichenfläche erfolgt durch die Methode `show`. In der Klasse `LineElement` wird dabei die Verbindungsstrecke zwischen `A` und `B` gezeichnet, in `StraightLine` wird die Gerade durch die beiden Punkte dargestellt. In der Klasse `Ray` wird ausgehend vom Punktobjekt `A` ein Strahl durch `B` gezeichnet.
Die Funktion `getProperty` in `LineElement` ermöglicht den Zugriff auf eine Reihe von Objekteigenschaften, wie etwa der Abstand `AB`, die Parameter der Funktionsform der Geradengleichung $y = mx + b$ oder die Parameter der allgemeinen Geradengleichung $ax + by + c = 0$.

```
protected double getProperty(int num) {
    switch (num) {
        case 0:
            // Abstand zwischen A und B
            return getLength();
        case 1:
            // Steigung m aus g: y = mx + b
            return A.worldSlope(B);
        case 2:
            // Achsenabschnitt b aus g: y = mx + b
            return Y_WindowToWorld(A.y) -
                   (A.worldSlope(B)*X_WindowToWorld(A.x));
        case 3:
            // Parameter a aus g: ax + by + c = 0
            return Y_WindowToWorld(A.y);
        case 4:
            // Parameter b aus g: ax + by + c = 0
            return -X_WindowToWorld(A.x);
        case 5:
            // Parameter c aus g: ax + by + c = 0
            return Y_WindowToWorld(B.y)*X_WindowToWorld(A.x)-
                   X_WindowToWorld(B.x)*Y_WindowToWorld(A.y);
    } // switch
    return Double.NaN;
}
```

Geometrische Objekte wie Parallelen, Orthogonalen oder Winkelhalbierende werden innerhalb von *Geometria* mit Hilfe von Punktobjekten konstruiert und durch Objekte der Klasse `LineElement` auf der Zeichenfläche dargestellt.

3.3.4 `CircleElement`

Durch die Klasse `CircleElement` wird das geometrische Objekt Kreis realisiert. In der Klasse wird ein Kreis durch den Mittelpunkt `Center` und den Abstand zwischen zwei Punkten `AB` als Radius definiert. Die Konstruktor-Methode sieht wie folgt aus:

```
CircleElement(PointElement Oval, PointElement Aval,
              PointElement Bval) {
    Center = Oval;
    A = Aval;
```

```
    B = Bval;
    addParent(Center, A, B);
}
```

Da die Lage eines Kreises durch Mittelpunkt und Radius vollständig bestimmt ist, ist keine `update`-Methode erforderlich. Die Methode `show` stellt die Kreislinie und die Kreisfläche auf der Zeichenfläche dar. Mit Hilfe der Funktion `getProperty` kann auf die Parameter der Kreisgleichung

$$k : (x - x_0)^2 + (y - y_0)^2 = r^2$$

und auf die folgenden Eigenschaften zugegriffen werden: Flächeninhalt, Umfang, Radius, Durchmesser und Mittelpunktkoordinaten.

```
protected double getProperty(int num) {
    switch (num) {
        case 0:
            // Flächeninhalt
            return getArea();
        case 1:
            // Umfang
            return getCircumference();
        case 2:
            // Radius
            return worldRadius();
        case 3:
            // Durchmesser
            return 2.0*worldRadius();
        case 4:
            // x-Koordinaten des Mittelpunkts
            return X-WindowToWorld(Center.x);
        case 5:
            // y-Koordinaten des Mittelpunkts
            return Y-WindowToWorld(Center.y);
        case 6:
            // Parameter x0 der Kreisgleichung
            return X-WindowToWorld(Center.x);
        case 7:
            // Parameter y0 der Kreisgleichung
            return Y-WindowToWorld(Center.y);
        case 8:
            // Parameter r^2 der Kreisgleichung
            return Math.pow(worldRadius(), 2);
    } // switch
    return Double.NaN;
}
```

Neben dem Zugriff auf Objekteigenschaften sind außerdem folgende Funktionen implementiert:

- `defined()` – Prüft, ob die Punktobjekte `A`, `B` und `Center` definiert sind.

- `isCongruent(CircleElement C)` – Prüft, ob das aktuelle Kreisobjekt kongruent zum Kreisobjekt `C` ist.
- `contains(PointElement P)` – Prüft, ob der Punkt `P` innerhalb der Fläche des aktuellen Kreisobjekts liegt.

3.3.5 SectorElement

Durch die Klasse `SectorElement` kann ein Kreisbogen und die Fläche eines Kreissektors realisiert werden. Ein solches Objekt wird durch drei Punktobjekte `Center`, `A` und `B` definiert. Das Punktobjekt `Center` legt den zugehörigen Kreismittelpunkt fest. Der Abstand zwischen `Center` und `A` bestimmt den Kreisbogenradius. Der Kreisbogen verläuft – ausgehend von `A` im mathematisch positiven Sinn – in Richtung auf `B` und wird begrenzt von der Geraden durch `Center` und `B` (Abbildung 3.25). In der Konstruktor-Methode werden die definie-

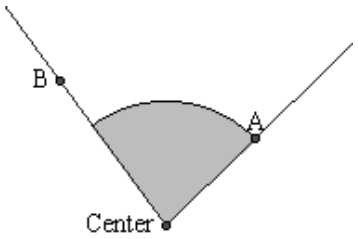

Abbildung 3.25: Kreisbogen (`SectorElement`).

renden Punktobjekte übergeben und in der Liste der Elternobjekte gepeichert. Die Methode ist wie folgt aufgebaut:

```
SectorElement(PointElement Cval, PointElement Aval,
              PointElement Bval) {
    Center = Cval;
    A = Aval;
    B = Bval;
    addParent(Center, A, B);
}
```

Eine `update`-Methode ist nicht erforderlich. Die Methode `show` zeichnet Kreisbogen und Kreissektor auf die Zeichenfläche. Zugriff auf die Objekteigenschaften Kreisbogenradius, Bogenlänge und Winkel erhält man durch die Funktion `getProperty`:

```
protected double getProperty(int num) {
    switch (num) {
        case 0:
            // Kreisbogenradius
            return A.worldDistance(Center);
        case 1:
```

```
            // Bogenlänge
            return getAngleValue()*(Math.PI/180.0)*
                   A.worldDistance(Center);
        case 2:
            // Winkel zwischen den Punkten A, Center und B
            return getAngleValue();
    } // switch
    return Double.NaN;
}
```

3.3.6 CurveElement

Mit Hilfe der Klasse `CurveElement` werden ebene, parametrisierte Kurven als geometrische Objekte realisiert. Dabei kann die Kurve in kartesischen Koordinaten in der Form

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} f_x(t) \\ f_y(t) \end{pmatrix} \text{ und } t \in [t_0, t_1]$$

oder durch polare Koordinaten r, t in der Form

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} r\cos t \\ r\sin t \end{pmatrix} \text{ mit } r = F(t) \text{ und } t \in [0, t_1]$$

angegeben werden. Für jede der beiden Definitionsformen sind in der Klasse `CurveElement` spezielle Methoden implementiert. Sie sind jedoch ähnlich aufgebaut. Aus Platzgründen werden im folgenden nur die Methoden für – die in kartesischen Koordinaten definierten – Kurvenobjekte erläutert.

Um ein Kurvenobjekt zu erzeugen, werden in der Konstruktor-Methode die Funktionen $f_x(t)$ und $f_y(t)$ als Zeichenketten an zwei Variablen `xTerm` und `yTerm` übergeben. Diese Zeichenketten enthalten als Variablen jedoch nicht nur den Parameter t, sondern ggf. noch weitere veränderliche Größen, die mit M_0, M_1, M_2, usw. bezeichnet sind. Jede Variable M_i steht stellvertretend für den Wert eines Funktionals der Klasse `Measure`. Die `Measure`-Objekte sind in den Listen `xList` und `yList` gespeichert. Die Intervallgrenzen des Kurvenparameters t werden in den Variablen `t0` und `t1` festgehalten und die Variable `num` speichert die Anzahl von Punkten, durch die die Kurve approximiert wird. Alle diese Variablen werden beim Aufruf an die Konstruktor-Methode übergeben.

Bevor der Quellcode der Konstruktor-Methode und der `update`-Methode betrachtet wird, soll kurz die Berechnung eines `CurveElement`-Objekts skizziert werden. Das Kernproblem besteht darin, die Zeichenketten `xTerm` und `yTerm` numerisch zu evaluieren und die Koordinaten der einzelnen Kurvenpunkte zu bestimmen. Um dies zu erreichen, müssen zuerst in den beiden Zeichenketten sowohl der Bezeichner t für den Kurvenparameter als auch alle Variablenbezeichner $M_0, M_1, M_2, \ldots$ durch die entsprechenden numerischen Werte ersetzt werden. Erst danach können mit Hilfe der Funktion `parseString` in der Klasse `MathParser`[23] die Zeichenketten als mathematische Terme interpretiert und berechnet werden. Die einzelnen Punktkoordinaten werden dann in den Listen

[23] Die Klasse `MathParser` wurde von mir mit dem Java-Compiler-Compiler *JavaCC* entwickelt. Als Ausgangsbasis diente eine von Chuck McManis programmierte Klasse. Ihren Quellcode habe ich so erweitert, daß zahlreiche mathematische Funktionen verfügbar sind (http://www.javaworld.com/javaworld/jw-12-1996/jw-12-jack.html).

`xPoint` und `yPoint` gespeichert, so daß die Kurven durch einen Streckenzug approximiert sind. Dazu ein Beispiel:

Dargestellt werden soll eine Ellipse mit der Parametergleichung

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a\cos t \\ b\sin t \end{pmatrix} \text{ und } t \in [0, 2\pi]\,.$$

Angenommen die beiden Parameter a und b können durch zwei Schieberegler auf der Zeichenfläche (als Objekte der Klasse `Measure`) variiert werden. Die Funktionen $f_x(t)$ und $f_y(t)$ würden dann durch `xTerm = "M0*cos(t)"` mit `xList[0]` = {`Measure`-Objekt `a`} und durch `yTerm = "M0*sin(t)"` und dem Variablenwert `yList[0]` = {`Measure`-Objekt `b`} wiedergegeben werden. Um nun beispielsweise die Koordinaten für den Parameterwert $t = 2,5$ berechnen zu können, muß in `xTerm` und `yTerm` das Zeichen `t` durch den numerischen Wert $2,5$ ersetzt werden. Das gleiche gilt für den Bezeichner `M0`. Sind etwa die Schieberegler auf die Werte $a = 2$ und $b = 3$ eingestellt, dann ist `xTerm = "2*cos(2.5)"` und `yTerm = "3*sin(2.5)"`. Jetzt kann man die Zeichenketten mit Hilfe der Klasse `MathParser` auswerten. Der Aufruf von `parseString("2*cos(2.5)")` liefert $-1,60224$ und der Aufruf `parseString("3*sin(2.5)")` ergibt $1,7954$. Die Punktkoordinaten werden in den Variablen `xPoint[i]` und `yPoint[i]` gespeichert. Auf diese Weise können alle Kurvenpunkte im Intervall $[t_0, t_1]$ bestimmt werden. Der folgende kommentierte Quellcode zeigt die Konstruktor-Methode:

```
CurveElement(String xF, Vector xVect, String yF, Vector yVect,
             double t_0, double t_1, int n) {
    type = 1;
    // Anzahl der Kurvenpunkte
    num = n;
    usePolarCoordinates = false;
    // xTerm und yTerm enthalten die Kurvengleichung
    // als Zeichenketten. Alle Funktionale sind darin
    // mit M0, M1, M2, usw., der Kurvenparameter
    // ist darin mit t bezeichnet.
    // Die Vector-Listen xVect und yVect enthalten die
    // zugehörigen Objekte der Klasse Measure.
    xTerm = xF;
    yTerm = yF;
    t0 = t_0;
    t1 = t_1;
    delta = (t1-t0)/(num-1);
    // param_t ist die Liste mit den Kurvenparameterwerten:
    // param_t[0] = t0,
    // param_t[1] = t0 + 1*delta,
    // param_t[2] = t0 + 2*delta,
    // ...
    // param_t[num-1] = t1.
    param_t = new double[num];
    for (int i=0;i<num;i++) {
        param_t[i] = t0 + (i*delta);
    } // for
```

```
    // Die Methode prepareTermStrings sorgt dafür, daß
    // die Listen xList und yList der Measure-Objekte
    // erstellt werden. Alle Objekte werden außerdem als
    // Elternobjekte dieser Kurve angemeldet.
    prepareTermStrings(xVect, yVect);
}
```

Innerhalb der update-Methode werden die einzelnen Kurvenpunkte berechnet. Die Methode ist im wesentlichen wie folgt implementiert:

```
protected void update() {
    // Die Funktion fx(t) wird als
    // Zeichenkette xTerm zusammengesetzt.
    // Darin werden anstatt der Bezeichner M0, M1, M2, ...
    // die aktuellen Werte der Measure-Objekte eingesetzt.
    if (nX>0) {
        xTerm = "";
        for (int i=0;i<nX;i++) {
            xTerm += xSubTerm[i] + xList[i].getValue();
        } // for
        xTerm += xSubTerm[nX];
    } // if

    // Die Funktion fy(t) wird als
    // Zeichenkette yTerm zusammengesetzt.
    // Darin werden anstatt der Bezeichner M0, M1, M2, ...
    // die aktuellen Werte der Measure-Objekte eingesetzt.
    if (nY>0) {
        yTerm = "";
        for (int i=0;i<nY;i++) {
            yTerm += ySubTerm[i] + yList[i].getValue();
        } // for
        yTerm += ySubTerm[nY];
    } // if

    // Berechne alle Kurvenpunkte mit den Parametern
    // param_t[i] und i=0...(num-1).
    for (int i=0;i<num;i++) {

        // Ersetze in xTerm das Zeichen "t" durch den Wert
        // von param_t[i].
        xTermPrep = MathFunc.prepareString(xTerm, param_t[i]);

        // Ersetze in yTerm das Zeichen "t" durch den Wert
        // von param_t[i].
        yTermPrep = MathFunc.prepareString(yTerm, param_t[i]);

        try {
            // Interpretiere die Zeichenketten xTermPrep und
            // yTermPrep als Terme und berechne ihren Wert.
            tdx = MathParser.parseString(xTermPrep);
```

```
            tdy = MathParser.parseString(yTermPrep);
        } // try
        catch (Exception e) {
            showDebugInfo("CurveElement.update: Fehler beim" +
                          "Aufruf von MathParser!");
        } // catch

        // Speichere die berechneten Koordinaten in den
        // Listen xPoint und yPoint.
        xPoint[i] = (int) X-WorldToWindow(tdx);
        yPoint[i] = (int) Y-WorldToWindow(tdy);
    } // for
}
```

Die Methode `show` stellt ein Kurvenobjekt auf der Zeichenfläche dar. Mit dem Befehl `drawPolyline(xPoint, yPoint)` wird darin ein Streckenzug auf die Zeichenfläche gezeichnet.

Wie bereits in Abschnitt 3.3.2 über ziehbare Punktobjekte angedeutet, ist in der Klasse `CurveElement` eine Methode `pointToCurve` definiert, die es möglich macht, daß man ein Punktobjekt auf einer Kurve bewegen und dabei auf den Wert des Kurvenparameters t zugreifen kann. Die Methode `pointToCurve` erwartet dazu die Übergabe eines Punktobjekts M mit den aktuellen Koordinaten des Mauszeigers und einer numerischen Variable t_Q zum Speichern des Kurvenparameters. Als Ergebnis liefert sie den Kurvenpunkt Q, dessen Koordinaten minimalen Abstand zum Punkt M besitzen. In der Variablen t_Q wird der Kurvenparameter vom Kurvenpunkt Q gespeichert. Dieses Verfahren erfolgt in zwei Schritten:

1. Der Index i von Kurvenpunkt P_i = (`xPoint[i]`, `yPoint[i]`) wird so bestimmt, daß der Abstand zwischen dem gezogenen Punktobjekt M und P_i minimal ist.

2. Anschließend wird geprüft, welcher der Kurvenpunkte P_{i-1} oder P_{i+1} näher an M liegt. Der Punkt Q ist derjenige Punkt auf der Strecke P_iP_{i-1} (oder P_iP_{i+1}), für den der Abstand QM minimal wird. Analog dazu wird der Kurvenparameter zwischen den Punkten P_i und P_{i-1} (oder P_i und P_{i+1}) linear interpoliert und in der Variablen t_Q gespeichert.

Ergänzend zu dem beschriebenen Verfahren gibt es in der Klasse `CurveElement` eine weitere Möglichkeit, Kurven als Objekte zu definieren. Über eine Schnittstelle können geeignete Java-Klassen eingebunden werden, die eine Kurve berechnen und den tabellierten Kurvenverlauf an ein Objekt der Klasse `CurveElement` übergeben. Der Kurvenverlauf kann dabei durch eine beliebige Anzahl von Streckenzügen beschrieben werden (und nicht bloß durch einen einzigen). Auf diese Weise kann der Figurenautor in einem Lernbaustein auch Kurvenobjekte darstellen, die durch rekursive Gleichungen oder durch unendliche Folgen definiert sind. Dabei wird jedoch stets nur eine spezielle Iterationsstufe angezeigt. Beispielsweise läßt sich so das – aus der fraktalen Geometrie bekannte – Sierpinski-Dreieck in einem Lernbaustein darstellen (Seite 164). Das Fraktal wird mit Hilfe einer externen Java-Klasse (Seite 396) durch eine endliche Zahl von Streckenzügen in Form von ähnlichen Dreiecken angenähert. Das ggf. aus

mehreren Streckenzügen bestehende Kurvenobjekt wird durch die folgenden vier Variablen realisiert:

1. Die ganzzahlige Variable `numInterval` speichert die Anzahl der Streckenzüge, durch die die Kurve dargestellt wird.
2. Die Feldvariable `nInterval[i]` mit $0 \leq$ `i` $\leq$ `numInterval` speichert zu jedem einzelnen Streckenzug die Anzahl der Kurvenpunkte, aus denen der jeweilige Streckenzug besteht.
3. Die Feldvariable `xPoint[i][j]` mit $0 \leq$ `i` $\leq$ `numInterval` und mit $0 \leq$ `j` $\leq$ `nInterval[i]` speichert die x-Koordinaten der Kurvenpunkte zu jedem Streckenzug.
4. Die Feldvariable `yPoint[i][j]` mit $0 \leq$ `i` $\leq$ `numInterval` und mit $0 \leq$ `j` $\leq$ `nInterval[i]` speichert die y-Koordinaten der Kurvenpunkte zu jedem Streckenzug.

Um ein solches Kurvenobjekt durch eine externe Java-Klasse zu berechnen, müssen drei Voraussetzungen erfüllt sein:

1. Die Klasse muß von der Klasse `Curve` abgeleitet werden.
2. Die Klasse muß eine Konstruktor-Methode enthalten, in der als Parameter eine String-Liste `elementList` mit Parametern für die Kurvenberechnung, eine Variable `numElement` mit der Anzahl der Einträge in `elementList`, das aktuelle `Slate`-Objekt und das aufrufende Kurvenobjekt übergeben werden.
3. Die Klasse muß eine Methode `update` enthalten, in der die Variablen `numInterval`, `nInterval`, `xPoint` und `yPoint` berechnet werden.

Erwähnenswert ist, daß sich auch ziehbare Punkte an – aus mehrfachen Streckenzügen bestehende – Kurvenobjekte binden lassen.

Beispiele für weitere Lernbausteine, die durch externe Java-Klassen definierte Kurvenobjekte enthalten, sind: Fraktaler Baum (Seite 163), Koch-Kurve (Seite 163) und die algebraische Kurve $y^2 = ax^4 + bx^2$ (Seite 154).

3.3.7 `LocusElement`

Mit Hilfe der Klasse `LocusElement` können Ortslinien als dynamische Objekte realisiert werden. Wird beispielsweise ein Punkt P auf einer Führungslinie gezogen, so bewegt sich ein – von P abhängiger – Punkt L auf einer bestimmten Bahn. Diese Bahn wird als die Ortslinie von L bezeichnet und kann in der Klasse `LocusElement` durch einen Streckenzug angenähert werden. Zur Definition eines Ortslinienobjekts sind drei Angaben erforderlich:

1. Ein ziehbares Punktobjekt `P`, das auf einer Führungslinie bewegt werden kann. Als Führungslinie sind Objekte der Klassen `LineElement`, `CircleElement`, `CurveElement` und `LocusElement` möglich.
2. Ein Punktobjekt `L`, dessen Bahn aufgezeichnet werden soll und das abhängig vom Punktobjekt `P` ist.

3. Eine Anzahl `num` von Stützpunkten, durch welche die Ortslinie approximiert werden soll. Dabei gilt: Je größer der Wert von `num` ist, desto genauer wird der Streckenzug an die Ortslinie angenähert, aber desto zeitaufwendiger ist die Berechnung.

Abhängig von den Objektklassen der Führungslinien werden unterschiedliche Konstruktor-Methoden verwendet. Durch die Variable `type` wird der Führungslinientyp gespeichert. Der folgende Quellcode zeigt die Konstruktor-Methode für eine Führungslinie der Klasse `CurveElement`.

```
LocusElement(PointElement Lval, Dragable Pval,
             CurveElement sCurve) {
    L = Lval;
    P = Pval;
    curve = sCurve;
    addParent(L, P, curve);

    // Die Ortslinie wird durch gleich viele Punkte
    // approximiert wie die Führungslinie.
    num = curve.num;

    // Punktliste, zum Speichern der Koordinaten der
    // Stützpunkte der Ortslinie.
    xLocus = new int[num];
    yLocus = new int[num];

    // Führungslinie = Klasse CurveElement
    type = 4;
}
```

Die Koordinaten sämtlicher Stützpunkte einer Ortslinie werden in der Methode `update` berechnet. Entsprechend dem Wert der Variable `type` wird der passende Algorithmus aufgerufen. Exemplarisch soll dies für den Fall gezeigt werden, daß als Führungslinie ein Objekt der Klasse `CurveElement` vorliegt:

```
protected void update() {
    switch (type) {
        [...]
        case 4: {
            // Speichere die aktuellen Koordinaten von
            // P (= ziehbarer Punkt auf CurveElement).
            oldX = P.x;
            oldY = P.y;

            // Ersetze die Koordinaten von P nacheinander durch
            // die Koordinaten (xPoint[i], yPoint[i]) der Kurve
            // und berechne dann alle von P abhängigen Objekte
            // neu. Die Koordinaten von L werden in der
            // Punktliste (xLocus[i], yLocus[i]) gespeichert.
            for (int i=0;i<num;i++) {
                P.x = curve.xPoint[i];
```

```
                P.y = curve.yPoint[i];

                // Markiere alle von P abhängigen Objekte.
                P.markChilds();

                // Dieses Ortslinien-Objekt darf nicht markiert
                // sein, sonst entsteht eine Rekursion ohne
                // Abbruchbedingung.
                this.marked = false;

                // Aktualisiere alle von P abhängigen Objekte,
                // insbesondere das Objekt L.
                P.updateChilds();

                xLocus[i] = (int) L.x;
                yLocus[i] = (int) L.y;
            } // for

            // Weise P seine ursprünglichen Koordinaten zu
            P.x = oldX;
            P.y = oldY;

            // ... und aktualisiere noch einmal alle von
            // P abhängigen Objekte.
            P.markChilds();
            this.marked = false;
            P.updateChilds();
            break;
        } // case
        [...]
    } // switch
}
```

Die Darstellung des Streckenzugs auf der Zeichenfläche erfolgt durch die Methode `show`. Damit ein Punktobjekt der Klasse `Dragable` auf einer Ortslinie bewegt werden kann, ist die Methode `pointToCurve` implementiert. Diese ist analog zur gleichnamigen Methode in `CurveElement` aufgebaut (Abschnitt 3.3.6).

Beim Experimentieren mit Ortslinienobjekten muß beachtet werden, daß die angezeigte Ortslinie nur durch einen Streckenzug approximiert ist. Deren Stützpunkte sind zwar numerisch berechnet, aber je nach Anzahl der verwendeten Punkte kann es sein, daß diese zu weit auseinanderliegen und die Kurve dadurch nicht ausreichend geglättet erscheint. Problematisch ist das oben beschriebene Aktualisierungsverfahren, wenn einzelne Punktkoordinaten unendliche Werte (`infinity`) annehmen. Dies ist etwa der Fall, wenn ein Schnittpunkt zwischen zwei Objekten nicht mehr existiert, d. h., wenn die zugehörige analytische Gleichung keine reelle Lösung mehr besitzt. In einer solchen Situation wird die Ortslinie mathematisch nicht mehr vollständig korrekt angezeigt. Dieses Problem tritt allerdings bei allen aktuellen Geometrie-Systemen (Stand: Juli 2000) auf. Eine Ausnahme bildet das Programm *Cinderella*, das Algorithmen

verwendet, die komplexe Werte verarbeiten.[24]

3.3.8 PolygonElement

Die Klasse `PolygonElement` realisiert geometrische Vielecke. Um ein Vieleck zu definieren, sind n Punktobjekte vorzugeben. Die besonderen Vorzüge eines Polygonobjekts – im Vergleich zu einem geschlossenen Streckenzug – liegen darin, daß man die Polygonfläche farbig darstellen kann und daß man Zugriff auf die Eigenschaften des Polygonobjekts erhält. Außerdem ist es möglich, den Kantenzug oder die Fläche des Polygonobjekts als Bezugsobjekt für ziehbare Punkte zu nutzen (Abschnitt 3.3.2).

Zur Initialisierung wird innerhalb der Konstruktor-Methode eine Liste `V` bereitgestellt, in der die Eckpunkte eines Polygonobjekts gespeichert werden. Mit Hilfe der Methode `addPointElement` werden dann alle Punktobjekte zur Liste `V` hinzugefügt und der Zähler `n` jeweils inkrementiert.

```
PolygonElement() {
    n = 0;
    V = new PointElement[MAX_VERTEX];
    // MAX_VERTEX = maximale Anzahl von Punkten,
    // die verwaltet werden können.
}

public void addPointElement(PointElement Pval) {
    V[n] = Pval;
    addParent(Pval);
    n++;
}
```

Eine `update`-Methode benötigt die Klasse nicht, da das Polygonobjekt durch die Eckpunkte vollständig bestimmt ist. Mit Hilfe der Methode `getProperty` kann auf die folgenden Polygoneigenschaften zugegriffen werden: Flächeninhalt, Umfang, Koordinaten des Schwerpunkts, Anzahl der Symmetrieachsen, Anzahl der Drehsymmetrien und Anzahl der inzidierenden Eckpunkte. Außerdem läßt sich durch booleschwertige Funktionen prüfen, ob das Polygon konvex, regulär oder affinregulär ist. Ist das Polygon ein Dreieck, so kann getestet werden, ob es gleichseitig, gleichschenklig, rechtwinklig, stumpfwinklig oder spitzwinklig ist. Ist das Polygon ein Viereck, so können folgende Viereckssklassen erkannt werden: Quadrate, Rechtecke, Rauten, Parallelogramme, gleichschenklige Trapeze, schiefe Trapeze, gleichschenklige Drachen, schiefe Drachen, schräge Drachen, Sehnenvierecke und pythagoräische Vierecke.

Die Algorithmen, die verwendet werden, um die Polygoneigenschaften zu bestimmen, beruhen auf Berechnungen der Koordinaten der Eckpunkte. Dabei muß aus praktischen Gründen eine gewisse Unschärfe berücksichtigt werden. Dies zeigt sich zum Beispiel in der (häufig verwendeten) booleschwertigen Funktion `isIncident`, die prüft, ob zwei Punkte miteinander inzidieren:

```
public boolean isIncident(PointElement A, PointElement B,
                          double t) {
```

[24] vgl. Kortenkamp 1999

```
    boolean b = false;
    if (Math.abs(A.distance(B)) <= t) {
        b = true;
    } // if
    return b;
}
```

Neben der Angabe von zwei Punktobjekten `A` und `B` ist als dritter Parameter ein Toleranzwert `t` zu übergeben, durch den eine gewisse Unschärfe erzielt werden soll. Dabei bedeutet beispielsweise `t` $= 1,5$, daß eine Toleranz von einem Bildschirmpunkt berücksichtigt wird. Dies hat sich in der Praxis als ausreichend herausgestellt. Dem Problem, daß sich durch diese Unschärfe Fehler addieren und dadurch eine Funktion ein falsches Ergebnis liefert, kann der Figurenautor vorbeugen, indem er in einem Lernbaustein den Rasterfangmodus einschaltet. Das bedeutet, daß Punkte im Zugmodus nur noch auf den Schnittpunkten eines Gitternetzes bewegt werden können. Mit Hilfe des Rasterfangmodus kann man außerdem dafür sorgen, daß ein Punkt nur ganzzahlige Koordinaten annehmen kann. Für den Anwender ist dies i. d. R. ebenfalls eine Erleichterung, denn die Figur läßt sich einfacher positionieren. Sollen beispielsweise vier Punkte so verschoben werden, daß sie ein gleichschenkliges Trapez bilden, so kann der Rasterfangmodus und ein eventuell im Hintergrund angezeigtes Gitternetz die Positionierung vereinfachen.

3.3.9 `PointSetElement`

Durch Objekte der Klasse `PointSetElement` können beliebige Punktmengen innerhalb der Zeichenfläche eines Lernbausteins dargestellt werden. Eine Punktmenge ist in diesem Zusammenhang eine Teilmenge der Menge aller Zeichenflächenpunkte. Diese wird jedoch nicht durch die Lösungsmenge einer Aussageform definiert, sondern ist festgelegt durch die Menge aller schwarz gefärbter Punkte einer Schwarz-Weiß-Grafik. Ein solches Punktmengenobjekt kann daher nicht analytisch beschrieben werden. Es besitzt keine Elternobjekte und ist demnach auch nicht im Zugmodus veränderbar. Konkret wird ein Punktmengenobjekt wie folgt erzeugt:

1. Als Konstruktionsparameter ist der Dateiname einer Schwarz-Weiß-Grafik anzugeben. Diese Bilddatei wird geladen, in einer Instanz der Klasse `Image` gespeichert und auf die Größe der Zeichenfläche skaliert.

2. Für jeden Punkt des Bildobjekts wird nun geprüft, ob dieser schwarz gefärbt ist. Ist das der Fall, dann gehört er zur Punktmenge und seine Koordinaten werden in den Listen `xPoints` und `yPoints` gespeichert. Die Variable `num` zählt, wie viele Punkte die Punktmenge umfaßt.

3. Sind alle Bildobjektpunkte überprüft, wird die Konstruktor-Methode der Klasse `PointSetElement` aufgerufen und eine Instanz erzeugt. Als Parameter werden die Punktlisten `xPoints` und `yPoints` und die Variable `num` übergeben.

Die wichtigste Aufgabe von Objekten der Klasse `PointSetElement` ist, daß diese als Bezugsobjekte für ziehbare Punkte dienen können. Dazu ist die Methode `pointToPointSet` implementiert, die von der `update`-Methode eines ziehbaren

Punktobjekts aufgerufen wird (Abschnitt 3.3.2). Als Parameter wird darin ein Punktobjekt P übergeben, das die aktuellen Mauszeigerkoordinaten gespeichert hat. Die `pointToPointSet`-Methode bestimmt zu P einen Punkt Q, der in der Punktmenge liegt und gibt diesen an das aufrufende Objekt zurück. Um die Koordinaten von Q zu ermitteln, sind zwei alternative Verfahren realisiert.

In dem ersten Verfahren werden die Listen `xPoints` und `yPoints` verwendet und Q als derjenige Punkt mit den Koordinaten (`xPoints[i]`, `yPoints[i]`) mit $0 \leq$ `i` $\leq$ `num` bestimmt, dessen Abstand zu P minimal ist. Diese Vorgehensweise sichert, daß zu jedem beliebigen Punkt P stets ein Punkt aus der Punktliste gefunden wird. Für den Schüler entsteht dadurch der Eindruck, daß er im Zugmodus Punkte wie an einem Gummiband ziehen kann. Dieses Verhalten ermöglicht eine einfache, intuitive Bedienung und ist deshalb (meiner Ansicht nach) als optimal zu betrachten (Beispielfigur: Satz von Holditch, Seite 131). Der Nachteil dieser Methode ist, daß sie bei großen Punktmengen (`num` $>$ 10000) zu langsam ist. Aus diesem Grund wurde eine zweite Variante entwickelt, die auch für große Punktmengen sehr effektiv funktioniert.

Dieses zweite Verfahren greift auf eine alternative Datenstruktur zurück, um die Punktmenge zu speichern. Diese Datenstruktur besteht aus einem booleschen Array `offscreenImage[w][h]`, wobei `w` die Breite und `h` die Höhe der Zeichenfläche in Bildschirmpunkten ist. Für jeden Punkt mit den Koordinaten (x, y) auf der Zeichenfläche ist in dem Array gespeichert, ob dieser zur Punktmenge gehört (`offscreenImage[x][y] = true`) oder ob dieser nicht zur Punktmenge gehört (`offscreenImage[x][y] = false`). Auf diese Weise läßt sich sehr schnell prüfen, ob ein Punkt P in der Punktmenge enthalten ist. Liegt P nicht innerhalb der Punktmenge, dann wird die Verschiebung mit der Maus zurückgenommen, indem dem Punkt P die Koordinaten zugewiesen werden, die er vor der Verschiebung besaß. Der Nachteil dieses Verfahrens ist, daß ein ziehbarer Punkt sehr eng am Mauszeiger entlang geführt werden muß. Der Schüler ist dadurch zu einer erhöhten Präzision beim Positionieren mit der Computermaus gezwungen (Beispielfigur: Labyrinth, Seite 171).

Der kommentierte Quellcode der Methode `pointToPointSet` sieht wie folgt aus:

```
public void pointToPointSet(PointElement P) {

    if (!useOffscreenImage) {
        // Berechnungen mit Punktlisten:

        minDist = Double.MAX_VALUE;
        bestX = P.x;
        bestY = P.y;

        // Betrachte jeden Punkt (xPoints[i], yPoints[i]).
        for (int i=0;i<num;i++) {

            // Miß den Abstand zwischen (xPoints[i], yPoints[i])
            // und P.
            aktDist = Math.sqrt(
                          (xPoints[i]-P.x)*(xPoints[i]-P.x) +
                          (yPoints[i]-P.y)*(yPoints[i]-P.y));
```

```
            if (aktDist < minDist) {
                minDist = aktDist;
                bestX = xPoints[i];
                bestY = yPoints[i];
            } // if
        } // for

        // Weise P die Koordinaten des Punkts mit minimalem
        // Abstand zu.
        P.x = bestX;
        P.y = bestY;
    } // if
    else {
        // Berechnungen mit offscreenImage:

        // Speichere die Koordinaten von P als
        // ganze Zahlen in den Variablen Px und Py.
        int Px = (int) Math.round(P.x);
        int Py = (int) Math.round(P.y);

        // Prüfe, ob P = (Px, Py) zur Punktmenge gehört.
        // Die booleschwertige Funktion checkPixel führt die
        // entsprechende Abfrage aus.
        if (!checkPixel(Px, Py)) {

            // Prüfe, ob die "Nachbarpunkte" von P zur
            // Punktmenge gehören. Die Variable PIXEL_TOLERANCE
            // bestimmt, in wievielen Zeilen und Spalten die
            // Nachbarpunkte von P  getestet werden sollen.
            for (int i=1;i<PIXEL_TOLERANCE;i++)
                for (int j=-i;j<=i;j++) {

                    if (checkPixel(Px-i, Py+j)) {
                        P.x = Px-i;
                        P.y = Py+j;
                        return;
                    } // if

                    if (checkPixel(Px+j, Py+i)) {
                        P.x = Px+j;
                        P.y = Py+i;
                        return;
                    } // if

                    if (checkPixel(Px+j, Py-i)) {
                        P.x = Px+j;
                        P.y = Py-i;
                        return;
                    } // if
```

```
                    if (checkPixel(Px+i, Py+j)) {
                        P.x = Px+i;
                        P.y = Py+j;
                        return;
                    } // if
                } // for
            } // for

            // Wenn P nicht innerhalb der Punktmenge liegt,
            // weise dem Punkt P die Koordinaten zu, die vor
            // dem Verschieben gespeichert wurden.
            P.x = P.xBackup;
            P.y = P.yBackup;
        } // if
    } // else
}
```

3.3.10 Measure

Durch die Klasse `Measure` und ihre Unterklassen werden geometrische Funktionale F realisiert. Ein solches Funktional liefert zu einem Objekttupel $(O_1, \ldots, O_n)$ eine reelle Zahl.

Die Abbildung 3.26 zeigt den strukturellen Aufbau der Klasse `Measure` und ihren Unterklassen. Die Mehrzahl der Unterklassen realisieren Funktionale, die spezielle geometrische Relationen zwischen mehreren Objekten durch numerische oder logische Werte ausdrücken. Die Art der Relation ist in der Abbildung 3.26 jeweils in Klammern unter dem Klassenbezeichner angegeben (z. B. `MeasureSimilarity` liefert den Wert 1, wenn zwei Objekte ähnlich sind).

Die Eigenschaften und Zustände einzelner Objekte werden durch die Klasse `MeasureProperty` durch numerische Werte ausgedrückt. Jede Klasse eines geometrischen Objekts besitzt zu diesem Zweck eine Funktion `getProperty`, die bestimmte Zustandsparameter als numerische Werte zurückgibt. In den vorangegangenen Klassenbeschreibungen (Abschnitt 3.3.2 - 3.3.9) wurde jeweils erwähnt, welche Eigenschaften eines Objekts auf diese Weise gemessen werden können. Die Klasse `MeasureCalculate` bietet zudem dem Figurenautor die Möglichkeit, eigene Funktionale zu definieren. Zusätzlich lassen sich mit Hilfe der Klasse `MeasureFunction` über eine Schnittstelle auch externe Funktionsbibliotheken einbinden.

Im folgenden soll zuerst der allgemeine Klassenaufbau von `Measure` betrachtet werden. Anschließend beschreibe ich exemplarisch die wesentlichen Methoden der Klassen `MeasureIncidence`, `MeasureCalculate` und `MeasureFunction`.

Der allgemeine Aufbau der Unterklassen von `Measure`

Die Unterklassen von `Measure` sind ähnlich aufgebaut wie die Klassen geometrischer Objekte. Sie enthalten eine Konstruktor-Methode, in der die Elternobjekte und Parameterwerte übergeben werden, eine `update`-Methode, in der der Wert eines Funktionals berechnet wird, und eine `show`-Methode, in der der Wert auf der Zeichenfläche dargestellt wird. Die oberste Klasse `Measure` stellt die allgemeinen Variablen und Methoden für die speziellen Unterklassen zur Verfügung.

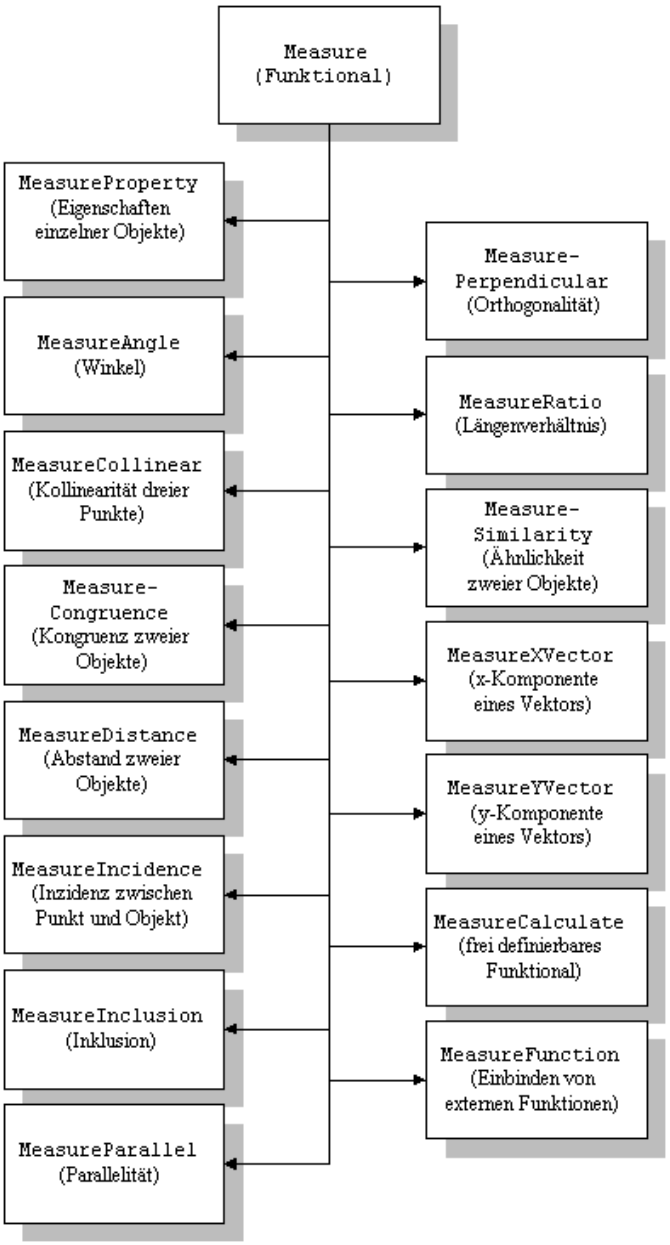

Abbildung 3.26: Klassenstruktur von `Measure`.

Dazu zählen:

- die numerische Variable `value`, die den aktuellen Wert eines Funktionals speichert,
- die Funktion `getValue`, die anderen Objekten ermöglicht, auf den Wert von `value` zuzugreifen,
- die numerischen Variablen `x` und `y`, welche Zeichenflächenkoordinaten enthalten,
- die beiden Zeichenkettenvariablen `preString` und `postString`, die zur Bezeichnung eines Funktionals dienen und
- die Methode `show`, die den gerundeten Wert von `value` zusammen mit den Zeichenketten `preString` und `postString` an der Position (`x`, `y`) auf der Zeichenfläche darstellt.

Die Klasse `MeasureIncidence`

Die Klasse `MeasureIncidence` realisiert eine booleschwertige Funktion, die zu zwei Objekten O_1, O_2 angibt, ob sie inzidieren. Dabei muß O_1 ein Objekt der Klasse `PointElement` sein und O_2 ein Objekt der Klasse `PointElement`, `LineElement`, `StraightLine` oder `CircleElement`. Je nachdem, ob das Punktobjekt O_1 mit dem Objekt O_2 inzidiert oder nicht, nimmt das Funktional den Wert 1 oder 0 an. Diese Entscheidung wird aufgrund von numerischen Berechnungen getroffen, die eine gewisse Toleranz berücksichtigen (vgl. Anmerkung auf Seite 75).

Entsprechend der Klasse des Objekts O_2 sind vier Konstruktor-Methoden implementiert. Der folgende Ausschnitt aus dem Quellcode zeigt exemplarisch eine der Konstruktor-Methoden.

```
// Prüfe auf Inzidenz zwischen PointElement und CircleElement.
MeasureIncidence(PointElement P_0, CircleElement C_0,
                 double x0, double y0, String pre, String post) {
    x = x0;
    y = y0;
    preString = pre;
    postString = post;
    P = P_0;
    circle = C_0;
    addParent(P, circle);
    type = 1;
}
```

Der Wert des Funktionals wird in der Methode `update` bestimmt, wie der folgende kommentierte Quelltextausschnitt zeigt.

```
protected void update() {
    value = 0.0;
    switch (type) {
        case 0:
            // Prüfe, ob P, P1 und P2 auf einer Geraden liegen.
```

```
            if (P.isCollinear(P1, P2)) {
                value = 1.0;
            } // if
            break;
        case 1:
            // Prüfe, ob P mit dem Kreis circle inzidiert.
            d = Math.abs(P.distance(circle.Center) -
                circle.radius());
            if (d < TOLERANCE) {
                value = 1.0;
            } // if
            break;
        case 2:
            // Prüfe, ob die beiden Punkte P und Q inzidieren.
            if (PointElement.isIncident(P, Q, TOLERANCE)) {
                value = 1.0;
            } // if
            break;
        case 3:
            // Prüfe, ob P, P1 und P2 auf einer Geraden liegen
            // und ob P zwischen P1 und P2 liegt.
            if (P.isCollinear(P1, P2) &&
                Math.abs(P1.distance(P0) + P2.distance(P0) -
                         P1.distance(P2)) < 0.0001) {
                value = 1.0;
            } // if
            break;
    } // switch
}
```

Einen ähnlichen Aufbau wie diese Klasse besitzen alle weiteren Klassen, mit denen geometrische Maße bestimmt und Relationen geprüft werden. Das sind: `MeasureCollinear`, `MeasureRatio`, `MeasureDistance`, `MeasureInclusion`, `MeasureParallel`, `MeasureSimilarity`, `MeasureCongruence`, `MeasureAngle`, `MeasurePerpendicular`, `MeasureXVector` und `MeasureYVector`.

Die Klasse `MeasureCalculate`

Mit Hilfe der Klasse `MeasureCalculate` kann der Figurenautor eigene Funktionen der Form $f\colon \mathbb{R}^n \to \mathbb{R}$ definieren. Der Definitionsbereich besteht aus n Funktionsvariablen, die im folgenden als $M_0, M_1, \ldots, M_{n-1}$ bezeichnet werden. Im Funktionsterm $f(M_0, M_1, \ldots, M_{n-1})$ können diese Funktionsvariablen durch die gängigen mathematischen Operationen und Funktionen verknüpft werden.

Die Klasse `MeasureCalculate` ist wie folgt implementiert: Jede der Funktionsvariablen $M_0, M_1, \ldots, M_{n-1}$ wird durch ein Objekt der Klasse `Measure` realisiert. Diese Objekte werden in dem Array `measureList[0..n-1]` gespeichert. Der Funktionsterm liegt in Form einer Zeichenkette in der Variablen `term` vor. Die Funktionsvariablen sind darin mit `M0`, `M1`, `M2`, ... bezeichnet. Um den Wert der Funktion zu berechnen, werden nun in der Zeichenkette `term` alle Variablenbezeichner durch die entsprechenden numerischen Werte der `Measure`-Objekte ersetzt. Mit Hilfe der Funktion `parseString` der Klasse `MathParser`

kann die Zeichenkette als mathematischer Term interpretiert und ausgewertet werden. Das Ergebnis wird in der Variablen `value` festgehalten. Die beiden folgenden kommentierten Programmausschnitte zeigen die Konstruktor- und die `update`-Methode der Klasse.

```
MeasureCalculate(String t, Vector mList, double x0, double y0,
                 String pre, String post) {
    type = 1;
    x = x0;
    y = y0;
    term = t;
    preString = pre;
    postString = post;

    // Die Vector-Liste mList enthält alle Termvariablen
    // (Measure-Objekte), die als Funktionsvariablen dienen.
    // Speichere jedes Measure-Objekt in dem Array measureList
    // und melde es als Elternobjekt an.
    n = mList.size();
    measureList = new Measure[n];
    for (int i=0; i<n; i++) {
        measureList[i] = (Measure) mList.elementAt(i);
        addParent((Measure) mList.elementAt(i));
    } // for

    // Um die Bezeichner M0, M1, M2, ... in der Zeichenkette term
    // sehr schnell durch die entsprechenden Funktionalwerte
    // ersetzen zu können, zerlege "term" nach dem Schema:
    // term = subTerm[0] + "M0" +
    //        subTerm[1] + "M1" +
    //        subTerm[2] + "M2" + ... +
    //        subTerm[n-1] + "M(n-1)" +
    //        subTerm[n]
    subTerm = new String[n+1];
    subTerm[0] = MathFunc.grepStringBetween(term, "", "M0");
    for (int i=1;i<n;i++) {
        subTerm[i] =
            MathFunc.grepStringBetween(term, "M"+(i-1), "M"+i);
    } // for
    subTerm[n] =
        MathFunc.grepStringBetween(term, "M"+(n-1), "");
}

protected void update() {
    expression = "";
    // Ersetze im String term alle Bezeichner Mi durch
    // den numerischen Wert des entsprechenden Funktionals.
    // Speichere den Term in der Variablen expression.
    for (int i=0; i<n; i++) {
        expression += subTerm[i] + measureList[i].getValue();
```

```
    } // for
    expression += subTerm[n];

    // Interpretiere die Zeichenkette expression als
    // mathematischen Term und speichere das Ergebnis
    // in der Variablen value.
    try {
        value = MathParser.parseString(expression);
    } // try
    catch (Exception e) {
        System.out.println("MeasureCalculate.getValue:" +
                           "Fehler im MathParser!");
        value = 0.0;
    } // catch
}
```

Basierend auf der Klasse `MeasureCalculate` wurde `MeasureCondition` entwickelt. Mit dieser Klasse wird eine Funktion der Form $f\colon \{T_1, T_2\} \rightarrow \{0, 1\}$ realisiert. Dabei sind T_1 und T_2 zwei Terme, die durch $<$, $>$, $=$ oder $!=$ verknüpft werden. Je nachdem, ob die so formulierte Aussage wahr oder falsch ist, wird die Variable `value` des `MeasureCondition`-Objekts auf 1 oder 0 gesetzt. Beide Terme T_1 und T_2 müssen Objekte der Klasse `MeasureCalculate` sein. Mit den Objekten der Klasse `MeasureCondition` erhält man booleschwertige Funktionen, die als sog. Prüffunktionen dienen. Solche Prüffunktionen werden wiederum bei der Definition eines Lernbausteins mit *GeoScript* benötigt (Abschnitt 3.2.3).

Ferner können durch Prüffunktionen auch Terme gebildet werden, die einfache Fallunterscheidungen enthalten. Dies erreicht man durch den folgenden *GeoScript*-Ausdruck:
`term = "if (MeasureCondition`-Objekt`) then (MeasureCalculate`-Objekt`) else (MeasureCalculate`-Objekt`)"`. Ein Beispiel findet sich auf Seite 169.

Die Klasse `MeasureFunction`

Die Klasse `MeasureFunction` bietet dem Figurenautor die Möglichkeit, eigene Funktionale $F\colon M \rightarrow \mathbb{R}$ durch externe Java-Klassen (Funktional-Klassen) zu definieren. Im Unterschied zur Klasse `MeasureCalculate` können auf diese Weise Funktionale zur Figurendefinition eingesetzt werden, die auf den kompletten Befehlssatz der Programmiersprache Java zurückgreifen. Dadurch lassen sich beispielsweise sehr spezielle und umfangreiche Antwortanalysen realisieren.

Die Klasse `MeasureFunction` ist wie folgt implementiert: Die Definitionsmenge M umfaßt alle Objekte und Variablen, die die Eingangsparameter des Funktionals darstellen. Sie wird durch die Einträge der Liste `elementList` definiert. Die Anzahl der Einträge ist in der Variablen `numElement` festgehalten. Die Liste wird beim Erzeugen einer Instanz der Klasse `MeasureFunction` übergeben. Vereinbarungsgemäß enthält dabei der Listeneintrag `elementList[0]` den Bezeichner für die einzubindende Funktional-Klasse. In der Konstruktor-Methode wird nun mit Hilfe dieses Bezeichners eine Instanz der Funktional-Klasse erzeugt. Damit kein Fehler auftritt, muß die Funktional-Klasse die folgenden drei Voraussetzungen erfüllen:

1. Die Funktional-Klasse muß von der Klasse `Functional` abgeleitet werden.
2. Die Klasse muß eine Konstruktor-Methode enthalten, in der als Parameter die Variable `numElement`, die Liste `elementList`, das aktuelle `Slate`-Objekt und das aufrufende `Measure`-Objekt übergeben werden.
3. Die Klasse muß eine Methode `getValue` enthalten, die den numerischen Wert des Funktionals zurückgibt.

Sind diese Voraussetzungen erfüllt, so kann mit einem `MeasureFunction`-Objekt auf beliebige Funktional-Klassen zugegriffen werden, ohne daß der Quellcode von *Geometria* neu übersetzt werden müßte. Der genaue Aufbau der Klasse `MeasureFunction` ist aus dem folgenden kommentierten Programmausschnitt ersichtlich.

```
public class MeasureFunction extends Measure {
    int numElement;
    String functionName;
    String[] elementList;
    Slate slate;
    Functional functional;

    MeasureFunction(int n, String[] eList, Slate s, double x0,
                    double y0, String pre, String post) {
        x = x0;
        y = y0;
        preString = pre;
        postString = post;
        slate = s;
        numElement = n;
        elementList = eList;

        // Der erste Listeneintrag enthält den Klassenbezeichner.
        functionName = elementList[0];

        try {
            // Erzeuge eine Instanz einer Unterklasse von
            // "Functional"
            functional = (Functional)
                Class.forName(functionName).newInstance();
            // ... und initialisiere dieses Objekt.
            // Die Parameter haben die folgende Bedeutung:
            // numElement := Anz. der Einträge in "elementList"
            // elementList := Liste mit allen Eingangsparametern
            // slate := das aktuelle Slate-Objekt (für den Zugriff
            //          auf Systemeigenschaften und -funktionen)
            // this := das aktuelle Measure-Objekt
            functional.init(numElement, elementList, slate, this);
        } // try
        catch (Exception e) {
            System.out.println("MeasureFunction.init: " +
```

```
                                "Die Klasse " + functionName +
                                " konnte nicht erzeugt werden.");
            e.printStackTrace();
        } // catch
    }

    protected void update() {
        value = functional.getValue();
    }
}
```

Beispiele für Lernbausteine, die durch externe Java-Klassen definierte Funktionale enthalten, sind: Picksche Formel (Seite 105), Problem der acht Damen (Seite 174), Würfelnetze (Seite 133) und Flächeninhalt unter einer Kurve (Seite 154).

3.4 *Geometria* im Vergleich

In den vorangegangenen Abschnitten wurde der Aufbau der wichtigsten Klassen von *Geometria* erläutert. Nun soll die Handhabung und der Funktionsumfang von *Geometria* mit der Handhabung und dem Funktionsumfang anderer Geometrie-Systeme verglichen werden. Dazu wird in Abschnitt 3.4.1 der allgemeine Arbeitszyklus des Figurenautors beim Erstellen eines Lernbausteins beschrieben und an einem Beispiel konkretisiert. Anschließend vergleiche ich das Arbeiten mit *Geometria* zuerst mit dem Konstruieren mit einem DG-System und anschließend mit dem Programmieren einer Individualentwicklung[25]. In Abschnitt 3.4.2 werden die wichtigsten besonderen Funktionen von *Geometria* genannt, die im Kapitel 4 durch unterschiedliche Beispiele veranschaulicht werden.

3.4.1 Der Arbeitszyklus des Figurenautors

Das praktische Entwickeln von Lernbausteinen auf der Grundlage einer Skriptsprache unterscheidet sich deutlich vom Konstruieren mit einem Konstruktionswerkzeug. Um einen Lernbaustein zu entwickeln, benötigt der Figurenautor einen Texteditor und einen Web-Browser. Der Texteditor ist erforderlich, um den Inhalt eines Lernbausteins mit *GeoScript* zu beschreiben. Der Web-Browser ist notwendig, um den Betrachter auszuführen und den Lernbaustein zu testen.

Um einen Lernbaustein zu entwickeln, sind die folgenden drei Arbeitsschritte erforderlich.

1. **Skriptdatei anlegen** Das Anlegen einer Skriptdatei ist der umfangreichste Teil des Entwicklungsprozesses. In einem Texteditor beschreibt der Figurenautor sämtliche Bestandteile (Figur, Textfenster, Hilfen, usw.) eines Lernbausteins mit *GeoScript* und speichert alles in einer Datei, die die

[25] Unter einer Individualentwicklung soll in diesem Zusammenhang ein interaktives Arbeitsblatt im weiteren Sinne verstanden werden, das direkt mit einer Programmiersprache entwickelt wurde. Hierzu zähle ich etwa die Java-Applets zur Mathematik von Fendt (http://home.augsburg.baynet.de/walter.fendt/math/indexm.htm) und die Sammlung *Bewegte Mathematik* von Stauff (http://www.muenster.de/~ stauff/bewmath.html).

Dateiendung ".script" besitzen muß. Die Syntax von *GeoScript* wurde bereits in Abschnitt 3.2.3 erläutert. Die Konstruktionsreferenz bietet dem Figurenautor eine Beschreibung sämtlicher Befehle (Anhang B, Seite 238).

2. **Layout-Vorlage vorbereiten** Das Erscheinungsbild eines Lernbausteins legt der Figurenautor in einer Layout-Vorlage fest. Diese besteht aus einer Textdatei, in der den Systemvariablen von *Geometria* konkrete Werte zugewiesen werden (Abschnitt 3.2.4). Um eine neue Layout-Vorlage zu erstellen, öffnet man am einfachsten eine vorhandene, ändert die gewünschten Variablenwerte und speichert sie unter einem neuen Namen ab. Die Dateiendung muß ".style" lauten. Um eine einheitliche Darstellung von mehreren Lernbausteinen zu erreichen, ist es sinnvoll, eine geeignete Layout-Vorlage mehrfach zu verwenden.

3. **HTML-Dokument erstellen** Im dritten Arbeitsschritt muß der Figurenautor ein HTML-Dokument erstellen, das den Betrachter durch den HTML-Befehl `<APPLET>` einbindet. Durch den `<PARAM>`-Befehl werden dem Applet die Dateinamen von Skript und Layout-Vorlage als Parameter übergeben. Sobald dieses Dokument von einem Web-Browser interpretiert wird, erzeugt dieser eine Instanz von `Geometria` und übergibt dabei sämtliche Parameter, die zur Darstellung des Lernbausteins erforderlich sind.

Nach dem dritten Schritt beginnt der Arbeitszyklus des Figurenautors in der Regel bei Punkt 1, da meistens der erste Entwurf eines Skripts noch verbessert werden kann. Sind außerdem noch Fehler in einem Skript enthalten, so werden entsprechende Meldungen und Kommentare vom Web-Browser ausgegeben. Der *Netscape Navigator* verwendet hierzu eine spezielle Java-Konsole, während der *Internet Explorer* alle Fehlermeldungen in die Datei "javalog.txt" schreibt.

Beispiel: Lernbaustein zum Begriff der Ableitung

Um den Arbeitszyklus zu veranschaulichen, soll ein Beispiel betrachtet werden. Die Abbildung 3.27 zeigt einen Lernbaustein zum Begriff der Ableitung (vgl. Seite 117). In dem Lernbaustein ist der Graph der Funktion

$$f(x) = \frac{x^5}{300} - \frac{11x^3}{360} + \frac{3}{2}x$$

dargestellt. Auf dem Graphen kann ein Punkt A bewegt werden. Durch A verläuft die Tangente an den Graphen. Zusätzlich wird die Steigung der Tangente angezeigt. Die zugehörige Skriptdatei "Ableitung.script", in welcher der Lernbaustein beschrieben ist, zeigt der folgende Quelltext:[26]

```
//
// Datei:    Ableitung.script
//

// Figurenbeschreibung
// ===================
```

[26] Zur einfacheren Beschreibung habe ich in dem folgenden Skript Zeilennummern hinzugefügt.

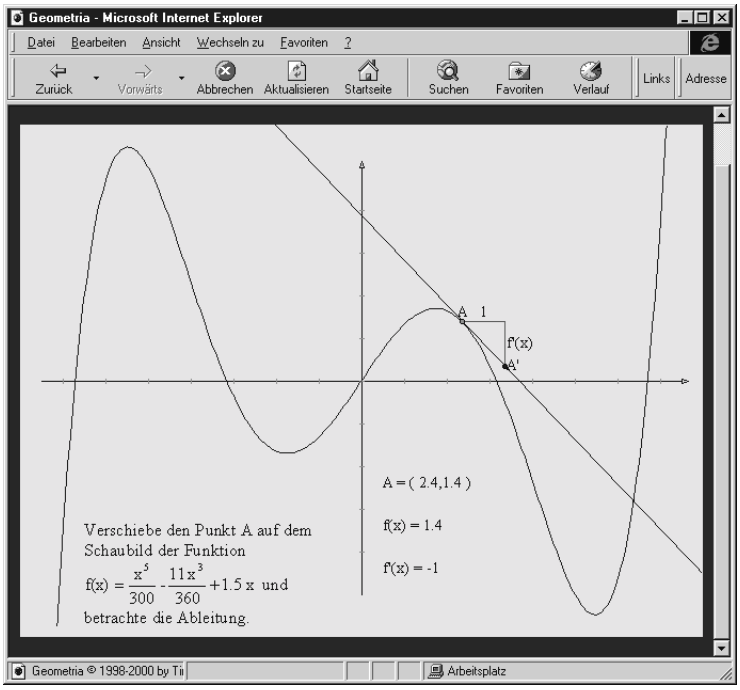

Abbildung 3.27: Lernbaustein zum Begriff der Ableitung.

```
   e[1]  = O;     point;    fixed;       0.0,0.0; "hidden"
   e[2]  = coord; point;    coordSystem; O,300,300,200,200;
10 e[3]  = curve; line;     curve;
           "t","t^5/300-11*t^3/60+1.5*t",-8.0,8.0,300;
   e[4]  = A;     point;    dragable;    0.5,0.5,curve;
   e[5]  = m0;    measure;  coordinates; A,0.5,-2.5,"A = ","";
   e[6]  = m1;    measure;  calculate;
15         "coordinateX(A)^5/300-11*coordinateX(A)^3/60+
           1.5*coordinateX(A)",0.5,-3.5,"f(x) = ","";
   e[7]  = m2;    measure;  calculate;
           "coordinateX(A)^4/60-33*coordinateX(A)^2/60+1.5",
           0.5,-4.5,"f'(x) = ","";
20 e[8]  = A';    point;    functionDepend;
           "coordinateX(A)+1.0","coordinateY(A)+calculate(m2)";
   e[9]  = t;     line;     straightLine; A,A'; "hideLabel"
   e[10] = A";   point;    functionDepend;
           "coordinateX(A)+1.0", "coordinateY(A)"; "hidden"
25 e[11] = f'(x); line;     connect; A',A";
   e[12] = l;     line;     connect; A",A;


30 // Ein- und Ausblenden von Objekten
   // ================================
```

```
   hidden[1] = "if (abs(calculate(m2)) > 0.01) hide (Textbox_1)"

35
   // Definition von Textfenstern
   // ===========================

   <TextBox>
40  Position = 340;20;-1;-1
     Extrempunkt
   </TextBox>


45 // Einbinden von Bild-Dateien
   // ========================

   image[1] = "Ableitung.gif", 60, 330

50 // Definition von Hilfen
   //======================

   <Help>
   Verschiebe A, so daß die Tangente
55 durch A horizontal verläuft.
   Welchen Wert hat dann die Ableitung?
   </Help>
```

Das Erstellen eines Skripts beginnt mit einer sorgfältigen Planung. Der Figurenautor sollte sich überlegen, aus welchen Bestandteilen der Lernbaustein bestehen soll. Vor allem bei der Definition der Figur muß der Figurenautor genaue Vorstellungen von den konstruktiven Abhängigkeiten der geometrischen Objekte besitzen. Dabei kann es hilfreich sein, erst einmal mit Papier und Bleistift eine Planskizze zu erstellen. In dem oben aufgeführten Skript erfolgt die Definition der Figur in den Zeilen 8-26.[27] Das Beispiel zeigt, wie der Figurenautor analytische Objekte und eigene Funktionale definieren kann (Zeilen 10-12, 14-21). Ein Textfenster ist in den Zeilen 39-42 beschrieben. Dieses wird jedoch ausgeblendet, bis die in der Zeile 33 definierte Prüffunktion den Wert 1 annimmt. Durch den Befehl in Zeile 48 wird die Bilddatei "Ableitung.gif" eingebunden, die eine Textinformation mit typographisch korrekter Darstellung einer Formel enthält. In den Zeilen 53-57 wird eine abrufbare Hilfe definiert. Das Skript ist unter dem Dateinamen "Ableitung.script" gespeichert.

Der folgende Quelltext zeigt den Aufbau der Layout-Vorlage "Web-Site.style". Aus Platzgründen will ich an dieser Stelle nicht beschreiben, welche Bedeutung die einzelnen Systemvariablen besitzen (Zeile 8-27). Stattdessen sei auf den entsprechenden Abschnitt in der Konstruktionsreferenz (Anhang B, Seite 238) verwiesen.

[27] In der ursprünglichen Skriptdatei wurde jeweils pro Objektbeschreibung nur eine Zeile benötigt. Aus Platzgründen mußte hier an einigen Stellen jedoch ein Zeilenumbruch eingefügt werden.

```
   //
   // Datei:    Web-Site.style
   //

5  // Globale Variablen
   // =================

   APPLET_WIDTH      = 640
   APPLET_HEIGHT     = 480
10 WORLD_X_MAX       = +8.0
   WORLD_X_MIN       = -8.0
   WORLD_Y_MAX       = +6.0
   WORLD_Y_MIN       = -6.0
   GridSize          = 10
15 GridColor         = 235, 205, 180
   FontSize          = 14
   Font              = Serif
   appletBgColor     = 255, 255, 255
   BackgroundColor   = 255, 225, 200
20 useSeparateWindow = false
   language          = German
   showLabel         = TRUE
   showGrid          = FALSE
   showAxis          = FALSE
25 snapToGrid        = FALSE
   allPointsDragable = FALSE
   MEASURE_EXACTNESS = 1


30 // Layout der geometrischen Objekte
   // ================================
  <elementTable>
    point; dragable;        black; red;   black; 0; smallCircle
    point; horizontal;      black; red;   black; 0; smallCircle
35  point; vertical;        black; red;   black; 0; smallCircle
    point; functionDepend;  black; blue;  black; 0; smallCircle
    point; fixed;           black; black; 0;     0; smallSquare
    line;  connect;         black; 0;     blue;  0;
    line;  curve;           0;     0;     blue;  0;
40  line;  straightLine;    black; 0;     black; 0;
  </elementTable>
```

Wie bereits in Abschnitt 3.2.4 erwähnt, wird das Aussehen der geometrischen Objekte tabellarisch festgelegt (Zeile 32-41). In den ersten beiden Spalten werden die jeweiligen Klassenbezeichner angegeben. Die Spalten 3-6 enthalten Farbbezeichner für die Beschriftungsfarbe, Punktfarbe, Linienfarbe und Flächenfarbe. Die 7. Spalte enthält eine Formkonstante für Punktobjekte.

Ein HTML-Dokument, wie es im dritten Arbeitsschritt erstellt worden ist, zeigt der folgende Quelltext. Entscheidend sind darin die Zeilen 12-22. Von Zeile 12-18 wird das einzubindende Applet angegeben. Die Dateinamen von Skript

und Layout-Vorlage werden durch die Zeilen 19 und 20 übergeben. Der Wert des Parameters `StartButton` legt fest, ob der Lernbaustein sofort beim Aufrufen des HTML-Dokuments angezeigt werden soll oder – als platzsparende Alternative – erst nach Auslösen eines Buttons in einem separaten Fenster.

```
1  <HTML>
     <HEAD>
       <TITLE>Geometria</TITLE>
     </HEAD>
5
     <BODY TEXT = "#FFFF99" BGCOLOR = "#004080">

       <CENTER>
       <FONT SIZE = +2>Ableitung</FONT>
10     <p></p>

       <APPLET
           code = "Geometria"
           codebase = ""
15         archive = "Geometria.jar"
           width = "640"
           height = "480"
       >
          <PARAM name = "Script"      value = "Ableitung.script">
20        <PARAM name = "Style"       value = "Web-Site.style">
          <PARAM name = "StartButton" value = "0">
       </APPLET>

       </CENTER>
25   </BODY>
   </HTML>
```

Der Arbeitszyklus im Vergleich

In diesem Abschnitt möchte ich die Figurenerstellung mit *GeoScript* und die Konstruktion einer Figur mit einem Konstruktionseditor vergleichen. Anschließend betrachte ich kurz das Programmieren einer Individualentwicklung.

Stellt man dem Arbeiten mit *GeoScript* das Bedienen eines Konstruktionswerkzeugs gegenüber, so kann ein Figurenautor mit einem interaktiven Konstruktionseditor schneller eine Figur konstruieren. Das Schreiben eines Skripts ist etwas zeitaufwendiger, zumal die Fehlersuche einen gewissen Raum einnimmt. Grundsätzlich besteht jedoch die Möglichkeit, nachträglich für *GeoScript* einen passenden Konstruktionseditor zu entwickeln. Eine Alternative wäre auch ein Übersetzer-Programm, das Figurendateien, die mit einem DG-System erzeugt wurden, nach *GeoScript* konvertiert.

Der Vorteil der Skriptsprache liegt dagegen in den erweiterten Möglichkeiten und besonderen Funktionen (Abschnitt 3.4.2), durch die spezielle – auf den jeweiligen Lernkontext zugeschnittene – Lernbausteine erzeugt werden können und die in einem Konstruktionswerkzeug nicht vorhanden sind.

Vergleicht man das Arbeiten mit *GeoScript* und das Programmieren einer Individualentwicklung, so arbeitet der Figurenautor ökonomischer, wenn er die

Skriptsprache verwendet. Er braucht in einem Skript lediglich den geometrischen Inhalt zu beschreiben und muß keine Datenstrukturen entwickeln, die die Variation im Zugmodus ermöglichen. Der Vorteil einer Individualentwicklung liegt in den vielfältigen Möglichkeiten, sehr spezielle Interaktionsformen zu realisieren. Dazu ist der volle Befehlsumfang einer höheren Programmiersprache verfügbar. Der Befehlsumfang von *GeoScript* kann damit natürlich nicht verglichen werden. Als Ausgleich ist jedoch eine Schnittstelle für externe Java-Klassen vorgesehen. Dadurch können Funktionsbibliotheken eingebunden werden, so daß sich auch komplexe Figurenanalysen und besondere Interaktionen realisieren lassen.

3.4.2 Besondere Funktionen von *Geometria*

Die Bestandteile eines Lernbausteins wurden in Abschnitt 3.1 bereits formal beschrieben. Nun geht es darum, die besonderen Funktionen von *Geometria* noch einmal zusammenfassend herauszustellen. In aufzählender Form werden Möglichkeiten und Funktionen genannt, die mit *Geometria* realisiert werden können.

Besondere ziehbare Punktobjekte Eine Figur muß nicht nur aus ziehbaren Punktobjekten bestehen, die innerhalb der gesamten Zeichenfläche oder entlang von eindimensionalen Bahnen bewegt werden können, wie es bei den aktuellen DG-Systemen üblich ist. Die Menge der Bezugsobjekte wurde so erweitert, daß Punktobjekte auch innerhalb von Polygon- und Kreisflächen sowie innerhalb beliebiger Punktmengen gezogen werden können. Beispielfiguren: Abstandssumme im gleichseitigen Dreieck (Seite 115), Beweis des Satzes von Pythagoras (Seite 98), Satz von Holditch (Seite 131), gleichseitiges Dreieck im Quadrat (Seite 137).

Beliebige funktionsabhängige Punktobjekte Durch die Klasse `FunctionDepend` lassen sich Punktobjekte erzeugen, deren Koordinaten durch zwei analytische Terme bestimmt werden. Diese Form der Definition eröffnet viele Möglichkeiten bei der Figurenerstellung. Sie ist beispielsweise zweckmäßig, um geometrische Objekte wie einen Krümmungskreismittelpunkt oder einen Tangentenvektor darzustellen. Insbesondere können dadurch spezielle Selbstabbildungen der Ebene realisiert werden, indem der Figurenautor eigene Vorschriften zum Abbilden von Punktobjekten definiert. Er ist auf diese Weise nicht durch eine vorgegebene und somit begrenzte Anzahl von Abbildungsvorschriften eingeschränkt, wie es bei vielen DG-Systemen der Fall ist. Beispielfiguren: Evolute (Seite 121), affine Abbildungen (Seite 146), nicht-affine Abbildung (Seite 147).

Kurvenobjekte Durch *GeoScript* lassen sich die Schaubilder von Kurven ohne Umwege erzeugen. Eine Kurve wird direkt durch eine Parametergleichung definiert und muß nicht erst umständlich durch ein Ortslinienobjekt konstruiert werden, wie es bei den meisten DG-Systemen erforderlich ist. An ein Kurvenobjekt lassen sich außerdem ziehbare Punktobjekte binden. Durch diese Punkte hat der Figurenautor Zugriff auf den jeweiligen Kurvenparameter und kann dadurch Tangenten, Normalen, Krümmungskreise, usw. darstellen. Über eine Schnittstelle können außerdem externe Java-Klassen eingebunden werden. Dadurch ist es

möglich, auch Kurven zu berechnen, die nicht durch Parametergleichungen beschrieben werden können. Dies ist beispielsweise bei Kurven der Fall, die durch Rekursionen oder Limites unendlicher Folgen definiert sind. Beispielfiguren: Ableitung (Seite 118), Evolute der Parabel (Seite 121), Sierpinski-Dreieck (Seite 163), Koch-Kurve (Seite 163), fraktaler Baum (Seite 163), Lernbaustein-Sequenz zur Behandlung von Funktionen (Seite 148), Spiralen (Seite 157).

Ortslinien Die Bahnbewegung von Punktobjekten läßt sich in einem Lernbaustein auf zwei Arten visualisieren. Als erstes kann ein dynamisches Ortslinienobjekt der Klasse `LocusElement` verwendet werden, das die Ortslinie als Kurve darstellt. Ein solches Ortslinienobjekt kann referenziert werden und es lassen sich ziehbare Punktobjekte daran binden. Eine zweite Möglichkeit besteht darin, die Ortsspur von Punktobjekten auf der Zeichenfläche aufzuzeichnen. Die Ortsspur ist dann jedoch kein dynamisches Objekt mehr. Beispielfiguren: Schachtelvolumen (Seite 101), Kissoide (Seite 160), Bézier-Kurven (Seite 160).

Punktmengen Mit *GeoScript* kann der Figurenautor beliebige Punktmengen auf der Zeichenfläche eines Lernbausteins darstellen. Eine Punktmenge läßt sich dann auch als Bezugsobjekt für ziehbare Punktobjekte verwenden. Ein Beispiel mit einer Eilinie als Punktmenge ist der Satz von Holditch[28], bei dem es um ein Ringflächengebiet innerhalb einer Eilinie geht (Seite 131). Weitere Beispiele sind zwei Labyrinth-Figuren (Seite 171).

Zustandsparameter anzeigen Zu einer Figur lassen sich zahlreiche Zustandsparameter anzeigen. Dazu zählen sämtliche Objekteigenschaften und Funktionale, die geometrische Relationen zwischen Objekten durch numerische Werte ausdrücken. Auf diese Weise läßt sich neben der visuellen Rückmeldung durch den Figurenzustand auch Rückmeldung durch numerische Größen geben. Beispielfiguren: Ableitung (Seite 118), Picksche Formel (Seite 109), Kurvenparameter (Seite 150), Taxi-Metrik (Seite 166), Maximum-Metrik (Seite 167), Eisenbahn-Metrik (Seite 169).

Termevaluation Mit *Geometria* können nicht nur Terme evaluiert werden, die die gängigen mathematischen Operationen und Funktionen enthalten, sondern der Figurenautor kann auch eingebaute und selbst definierte Funktionale in einen Term mit einbinden. Außerdem lassen sich beim Auswerten eines Terms einfache Fallunterscheidungen der Form if-then-else treffen. Über eine Schnittstelle können zusätzlich externe Funktionen aufgerufen werden, für deren Definition der volle Umfang der Programmiersprache Java verfügbar ist. Dadurch lassen sich komplexe Figurenanalysen und Interaktionsformen realisieren, die mit gängigen DG-Systemen nicht erzielt werden können. Beispielfiguren: Picksche Formel (Seite 109), Würfelnetze (Seite 133), Problem der acht Damen (Seite 174), Flächeninhalt unter Kurven (Seite 154), Eisenbahn-Metrik (Seite 169).

Bilder einbinden Mit *GeoScript* ist es möglich, Bilddateien einzubinden und diese auf der Zeichenfläche eines Lernbausteins anzuzeigen. Dieses kann zu illustrativen Zwecken dienen oder dazu, komplizierte Formeln typographisch kor-

[28] Blaschke & Müller 1956, S. 120

rekt darzustellen. Ein Bild kann an die Koordinaten eines Punktobjekts gebunden und so zu einem quasi-dynamischen Objekt als ein Teil der interaktiven Figur werden. Beispielfiguren: Schaltkreis Volladdierer (Seite 102), Problem der acht Damen (Seite 174), Labyrinthe (Seite 171), Ableitung (Seite 118).

Objekte ein- und ausblenden In *GeoScript* können mit Hilfe von booleschwertigen Prüffunktionen spezielle Figurenzustände definiert werden. Wird ein solcher Figurenzustand im Zugmodus hergestellt, so lassen sich beliebige Objekte ein- und ausblenden. Beispielsweise können dadurch dem Schüler zu bestimmten Figurenzuständen Texte mit Zusatzinformationen angezeigt werden. Auch läßt sich so zu einem Viereck immer dann der Umkreis darstellen, wenn es ein Sehnenviereck ist. Beispielfiguren: Satz von Pythagoras (Seite 98), Eckenschwerpunkt im Dreieck (Seite 98), Schaltkreis Volladdierer (Seite 102), Ableitung (Seite 118), Flächeninhalt umfangsgleicher Vierecke (Seite 107), Tangram (Seite 171), Haus der Vierecke (Seite 135).

Zustandsraum der Figur begrenzen Mit *GeoScript* kann der Zustandsraum einer Figur gezielt eingeschränkt werden. Auf diese Weise läßt sich die Figur genau dem Definitionsbereich des darzustellenden Lerninhalts anpassen. Soll etwa in einem Lernbaustein ein Satz visualisiert werden, der nur für konvexe Polygone gilt, so kann der Figurenautor den Zustandsraum so begrenzen, daß keine nicht-konvexen Polygonobjekte im Zugmodus hergestellt werden können. Beispielfiguren: Satz von Grashof (Online-Skript zur Elementargeometrie, Figur 2.1), Lotsumme im gleichseitigen Dreieck (Seite 115), gleichseitiges Dreieck im Quadrat (Seite 137).

Hilfetexte Hilfetexte können den Schüler unterstützen, eine Aufgabe in einem Lernbaustein zu lösen. Dazu lassen sich Textinformationen definieren, die durch Auslösen eines Buttons wiederholt angezeigt werden können. Es ist aber auch möglich, eine Sequenz von Hilfen festzulegen, die der Schüler nacheinander jeweils nur einmal abrufen kann. Beispielfigur: Satz von Ceva (Online-Skript zur Elementargeometrie, Figur 1.3), Ableitung (Seite 118).

Antwortanalyse Um dem Schüler eine Möglichkeit zur Selbstkontrolle zu bieten, kann der Figurenautor mit *GeoScript* eine Antwortanalyse definieren. Dadurch läßt sich der vom Schüler hergestellte Figurenzustand automatisch Bewerten und Kommentieren. Eine besondere Stärke ist dabei, daß auch Schülerantworten erkannt werden können, die nur teilweise richtig sind. Beispielfiguren: Würfelnetze (Seite 133), Haus der Vierecke (Seite 135), Aufgabe zur Geradenspiegelung (Seite 145), Drehstreckung eines Dreiecks (Seite 146), Mengensprache (Seite 170).

Kapitel 4

Didaktische Analyse zum Einsatz von Lernbausteinen

In diesem Kapitel untersuche ich die didaktischen Möglichkeiten von Lernbausteinen und unterscheide dabei die folgenden Anwendungsbereiche: Lernbausteine im Kontext der Instruktion (Abschnitt 4.1), Lernbausteine zum Explorativen Lernen (Abschnitt 4.2) und Lernbausteine zur Selbstkontrolle (Abschnitt 4.3). Eine Beispielsammlung (Abschnitt 4.4) gibt einen Überblick über Themengebiete, in denen Lernbausteine gewinnbringend eingesetzt werden können.

4.1 Lernbausteine im Kontext der Instruktion

Im Kontext der Instruktion können Lernbausteine als Medium zum Lehren von mathematischen Sachverhalten eingesetzt werden. Dabei soll das Lehrziel (beispielsweise die Kenntnis eines Satzes und seine Gültigkeit vermitteln) möglichst effektiv erreicht werden.[1] Zu diesem Zweck kann der Figurenautor Lernbausteine vorbereiten, die einen Sachverhalt anschaulich verdeutlichen. Solche Lernbausteine lassen sich an unterschiedlichen Lernorten einsetzen: beim Computervortrag im Klassenzimmer oder in interaktiven Lernumgebungen wie etwa innerhalb eines Online-Tutorials auf Web-Seiten. Besonders geeignet sind die folgenden Bereiche der Instruktion:

1. Visualisierung von geometrischen Sachverhalten,
2. Demonstration von Bewegungsphasen und
3. Simulation von geometrischen und technischen Modellen.

4.1.1 Visualisierung

Bei der Visualisierung von geometrischen Sachverhalten mit Lernbausteinen geht es darum, durch eine bewegliche Figur Sätze und Beweise zu veranschaulichen und die Bedeutung von Begriffen in verschiedenen Ausprägungen darzustellen. Mit Hilfe von Lernbausteinen kann ein Sachverhalt in drei Stufen präsentiert werden:

[1] vgl. Holland 1996, S. 139

1. **Gesamtüberblick über die Figur geben** In dieser Phase soll der Schüler die Figur kennenlernen. Dazu zeigt der Lehrende, aus welchen (sichtbaren) geometrischen Objekten die Figur besteht und welche konstruktiven Abhängigkeiten zwischen ihnen vorhanden sind. Er weist darauf hin, welche Teile der Figur im Zugmodus beweglich und welche konstruktiv abhängig sind.

2. **Sachverhalt an einem Figurenzustand erklären** Der Lehrende versetzt die Figur in einen bestimmten Anfangszustand und erläutert den Sachverhalt, der dem Schüler vermittelt werden soll. Der Lernbaustein ist in dieser Phase noch vergleichbar mit einer Zeichnung, die einen Sachverhalt (statisch) veranschaulicht.

3. **Sachverhalt durch Variation der Figur verallgemeinern** Der Lehrende verallgemeinert den dargestellten Sachverhalt, indem er die Figur direkt oder indirekt variiert und dabei zeigt, daß der Sachverhalt auch für andere Figurenzustände gültig ist. Unter direkter Variation verstehe ich das Ziehen von Objekten im Zugmodus, unter indirekter Variation das Verändern von numerischen Parameterwerten durch Schieberegler. Das Verallgemeinern durch Variation stellt einen besonderen Vorteil von Lernbausteinen gegenüber einer statischen Zeichnung dar.

Beispiel: Affine Abbildungen

Der folgende Lernbaustein ist ein Beispiel für die Visualisierung der Eigenschaften affiner Abbildungen. In dem Lernbaustein wird die allgemeine affine Abbildung $f\colon \mathbb{R}^2 \to \mathbb{R}^2$ mit $f(x,y) := (a_1x + b_1y + v_1, a_2x + b_2y + v_2)$ visualisiert. In der Figur können die Parameter a_1, a_2, b_1, b_2, v_1 und v_2 durch Schieberegler auf spezielle Werte eingestellt werden. Die sich daraus ergebende spezielle Abbildungsvorschrift wird in einem Textfenster angezeigt. Die Zeichenfläche enthält außerdem ein Viereck $ABCD$, dessen Eckpunkte frei bewegt werden können. Das Viereck $A'B'C'D'$ ist das zugehörige Bild von $ABCD$ unter der speziellen Abbildung f (Abbildung 4.1). In der Ausgangslage der Figur sind folgende Parameterwerte eingestellt: $a_1 = 1$, $a_2 = 0$, $b_1 = 0$, $b_2 = -1$, $v_1 = 0$ und $v_2 = 0$ (Abbildung 4.1). Die spezielle Abbildung f ist dann eine Spiegelung an der x-Achse. Der Lehrende verdeutlicht die Berechnung der Bildpunkte, indem er die Koordinaten einzelner Punkte abliest und in die Abbildungsvorschrift f einsetzt. Beispielsweise für den Punkt $A = (4,4)$ ist $A' = f(4,4) = (1 \cdot 4 + 0 \cdot 4 + 0, 0 \cdot 4 - 1 \cdot 4 + 0) = (4,-4)$. Durch Variation von $ABCD$ können folgende Eigenschaften der Achsenspiegelung als Invarianzaussagen herausgestellt werden: Ist eine Strecke zur x-Achse parallel, so ist auch das Bild der Strecke zur x-Achse parallel. Steht eine Strecke senkrecht zur Abszisse, so auch das Bild der Strecke. Schneiden sich zwei Strecken, so schneiden sich auch ihre Bilder. Alle Punkte, die auf der Abszisse liegen, sind Fixpunkte.

Durch Variieren der Parameterwerte kann der Lehrende andere Abbildungsvorschriften einstellen, etwa eine Spiegelung an der y-Achse oder an der ersten Winkelhalbierenden, eine Parallelverschiebung, eine Halbdrehung oder eine Drehung um 90° um den Ursprung, eine zentrische Streckung von einem bestimmten Punkt aus oder eine Dehnung parallel zur Abszisse oder Ordinate. Auch hier können die Eigenschaften affiner Abbildungen Inzidenztreue, Geradentreue und Parallelentreue anhand von Bild- und Urbildviereck visualisiert werden.

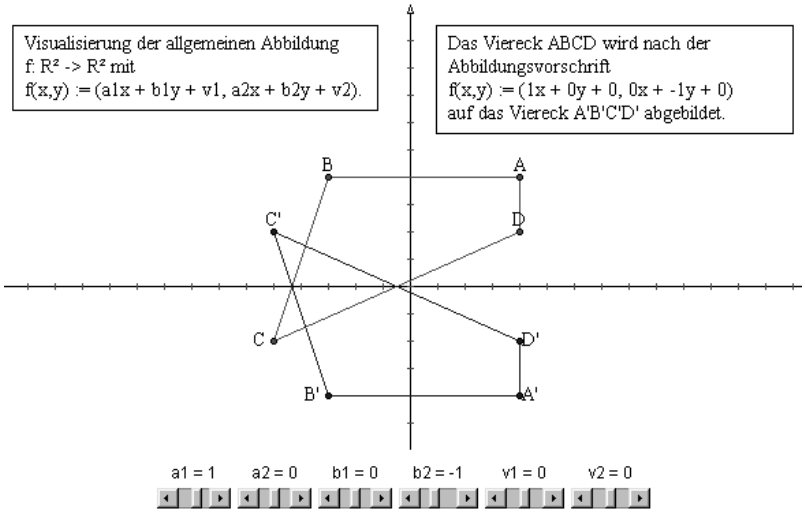

Abbildung 4.1: Visualisierung von affinen Abbildungen.

4.1.2 Demonstration

Bei der Demonstration von Bewegungsphasen geht es darum, einen Vorgang, der aus mehreren Phasen besteht, anschaulich vorzuführen. Die einzelnen Phasen werden durch Verziehen der Figur demonstriert. Inhaltlich kann es sich dabei beispielsweise um das Vorführen eines Beweises in mehreren Stufen handeln. Es lassen sich aber auch Sachverhalte durch kontinuierliche Bewegung demonstrieren, etwa das dynamische Generieren einer Ortsspur. Bei der anschaulichen Demonstration von Bewegungsphasen lassen sich folgende Tätigkeiten unterscheiden:

1. **Gesamtüberblick über die Figur geben** Das Vermitteln eines Gesamtüberblicks entspricht der ersten Phase beim Visualisieren von geometrischen Sachverhalten. Der Schüler soll die Figur kennenlernen. Dazu zeigt der Lehrende auf, welche Objekte beweglich und welche konstruktiv abhängig sind, und beschreibt die Besonderheiten der Figur in der Ausgangslage.

2. **Vorführen des Vorgangs** Ausgehend von der Ausgangslage, versetzt der Lehrende die Figur nacheinander in verschiedene Lagen und demonstriert so den zu zeigenden Vorgang. Er erläutert und kommentiert dabei jede einzelne Phase. Dieses kann durch den Lernbaustein unterstützt werden, indem entsprechend dem Figurenzustand Textinformationen eingeblendet werden.

3. **Sachverhalt durch Variation der Figur verallgemeinern** Die Gültigkeit des Sachverhalts wurde für die Figur in der Ausgangslage gezeigt. Um dieses zu verallgemeinern, versetzt der Lehrende die Figur in einen anderen Anfangszustand und demonstriert die Bewegungsphasen erneut.

Beispiel: Satz von Pythagoras

Mit Hilfe des folgenden Lernbausteins kann ein Beweis des Satzes von Pythagoras in mehreren Phasen vorgeführt werden. Gegeben ist ein rechtwinkliges Dreieck ABC mit innerhalb der Zeichenfläche frei beweglichen Ecken A, B. Der Eckpunkt C ist auf dem Thaleskreis über AB ziehbar, so daß das Dreieck stets rechtwinklig bleibt. Über den Dreiecksseiten a, b, c sind die Quadrate eingezeichnet. Die Flächen der beiden Kathetenquadrate sind hellgrau und dunkelgrau gefärbt. Der zu beweisende Satz besagt: Die Summe der Flächeninhalte der beiden Kathetenquadrate ist gleich dem Flächeninhalt des Hypotenusenquadrats (Abbildung 4.2).

Um den Beweis vorzuführen, demonstriert der Lehrende eine Scherung der Kathetenquadrate in drei Schritten. Er zieht den Punkt P parallel zur Seite AC, bis P auf der (verlängerten) Höhengeraden h_C (nicht eingezeichnet) liegt. Das Kathetenquadrat über b wird dadurch in ein flächengleiches Parallelogramm transformiert, dessen Seiten paarweise die Länge b und c haben (Abbildung 4.3 und 4.4). In der Figur bewegt sich das Kathetenquadrat über a dabei entsprechend zum Kathetenquadrat über b mit. Im nächsten Schritt werden die beiden Parallelogramme mit den Flächeninhalten a^2 und b^2 senkrecht zur Hypotenuse in Richtung auf C verschoben. Ihr Flächeninhalt ändert sich dabei nicht (Abbildung 4.5 und 4.6). Die beiden Parallelogramme werden parallel zur verlängerten Höhengeraden h_C geschert und so in zwei flächengleiche Rechtecke transformiert, die zusammen den Flächeninhalt c^2 ergeben (Abbildung 4.7).

Damit wurde der Scherungsbeweis für ein spezielles Dreieck vorgeführt. Die Einsicht, daß der Beweis auch für andere rechtwinklige Dreiecke gilt, kann vermittelt werden, indem der Lehrende die Form und Lage des Ausgangsdreiecks verändert und den Beweisvorgang erneut durchführt.

Beispiel: Eckenschwerpunkt im Dreieck

In dem folgenden Lernbaustein wird der Eckenschwerpunkt eines Dreiecks in drei Stufen durch kontinuierliches Variieren bestimmt. Die Figur besteht aus einem Dreieck ABC, dessen Eckpunkte frei in der Zeichenfläche bewegt werden können. Dabei soll angenommen werden, daß in den Eckpunkten drei gleich schwere Punktmassen m_A, m_B, und m_C anliegen (Abbildung 4.8). Der Eckenschwerpunkt ist der Gleichgewichtspunkt der drei Punktmassen. In der ersten Bewegungsphase soll der Schwerpunkt zwischen den Massen m_A und m_B bestimmt werden. Dazu wird der Punkt A in Richtung auf B verschoben. Der Punkt B bewegt sich dabei entsprechend in Richtung auf A (Abbildung 4.9). Sie treffen sich im Mittelpunkt der Seite AB. Die dort vereinigte Punktmasse besitzt doppeltes Gewicht und wird durch einen größeren Punkt angedeutet (Abbildung 4.10). In der zweiten Bewegungsphase gilt es, den Schwerpunkt zwischen der Zweifach-Masse und C zu bestimmen. Zu diesem Zweck wird die Zweifach-Masse in Richtung auf C verschoben. Dabei bewegt sich C doppelt so schnell auf den Mittelpunkt von AB zu. Die Unterteilungen der Hilfslinie veranschaulichen diesen Sachverhalt (Abbildung 4.11 und 4.12). Der Schwerpunkt der Zweifach-Masse und der Masse in C ist der Eckenschwerpunkt des Dreiecks. Er teilt die Mittellinie im Verhältnis 2:1 (Abbildung 4.13). Daß dieses Verfahren der Schwerpunktbestimmung allgemein gültig ist, kann der Lehrende an weiteren Dreiecken vorführen.

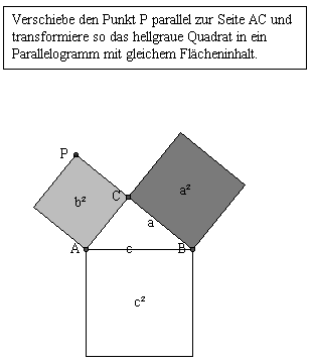

Abbildung 4.2: Figur zum Satz von Pythagoras in der Ausgangslage.

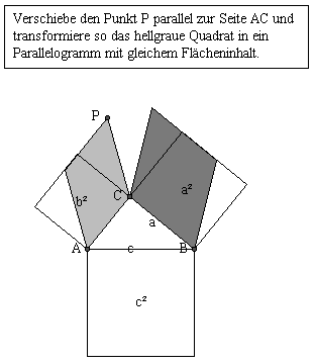

Abbildung 4.3: Der Punkt P wird parallel zur Seite AC verschoben.

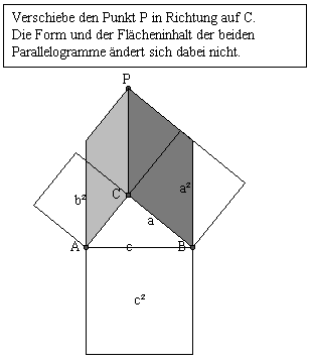

Abbildung 4.4: Die Kathetenquadrate sind in zwei flächengleiche Parallelogramme transformiert worden.

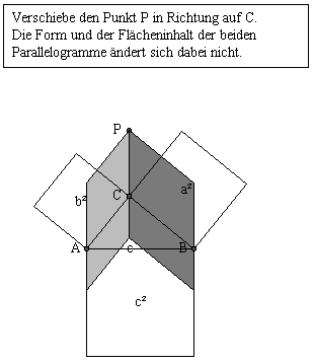

Abbildung 4.5: Die beiden Parallelogramme werden lotrecht in Richtung auf C verschoben.

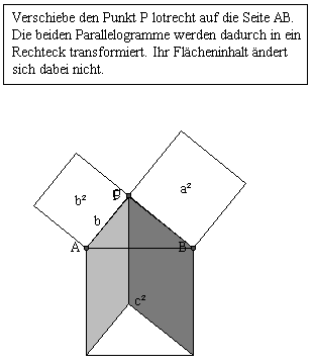

Abbildung 4.6: Die Parallelogramme müssen noch in zwei flächengleiche Rechtecke transformiert werden.

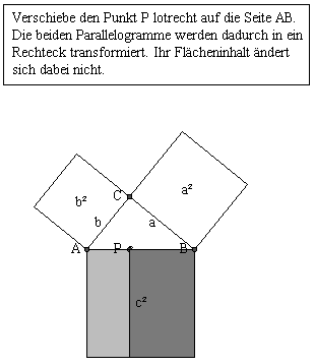

Abbildung 4.7: Die Summe der beiden Kathetenquadrate ist gleich dem Hypotenusenquadrat.

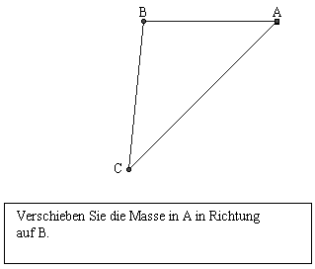

Abbildung 4.8: Dreieck ABC in Ausgangslage.

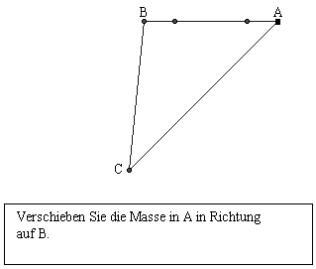

Abbildung 4.9: Die Masse in A wird in Richtung auf B verschoben.

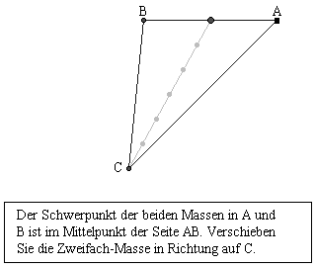

Abbildung 4.10: Es entsteht eine Zweifach-Masse.

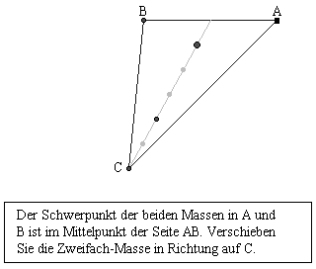

Abbildung 4.11: Die Zweifach-Masse wird in Richtung auf C verschoben.

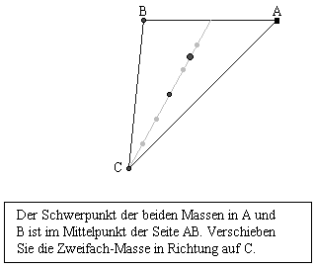

Abbildung 4.12: Die Zweifach-Masse wird in Richtung auf C verschoben.

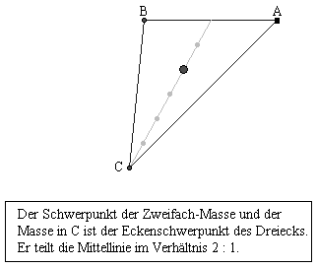

Abbildung 4.13: Die Massen treffen sich im Eckenschwerpunkt des Dreiecks.

4.1.3 Simulation

Bei einer Simulation wird der Ablauf und das Verhalten von geometrischen und technischen Modellen nachgeahmt. Dazu können bei einer entsprechend vorbereiteten Figur diejenigen Einflußgrößen des Modells verändert werden, die speziellen Handlungen in der Realität entsprechen. Durch eine Simulation lassen sich verschiedene Szenarien erproben und Modellzustände vorführen. Dieses kann in zwei Phasen geschehen:

1. **Gesamtüberblick über das Modell geben** In dieser Phase soll ein Überblick über das Modell gegeben werden. Dazu erklärt der Lehrende, welcher Sachverhalt durch die Figur simuliert wird, und beschreibt, aus welchen Bestandteilen sich das Modell zusammensetzt. Er weist auf die Einflußgrößen hin und demonstriert, wie diese variiert werden können. Außerdem zeigt er, wie die Ausgangsgrößen des Modells in dem Lernbaustein dargestellt werden und welche Ausprägungen sie annehmen können.

2. **Vorführen verschiedener Modellzustände** Der Lehrende führt verschiedene Modellzustände vor, indem er die Einflußgrößen des Modells verändert und auf die Auswirkungen bei den Ausgangsgrößen hinweist. Dabei kann der funktionale Zusammenhang zwischen den Eingangs- und Ausgangsgrößen des Modells bewußt herausgestellt werden. Gegebenenfalls ist es gewinnbringend, auf spezielle Modellzustände aufmerksam zu machen.

Beispiel: Schachtelvolumen

Das Thema des folgenden Lernbausteins orientiert sich an einem von Schumann[2] veröffentlichten interaktiven Arbeitsblatt. Der Lernbaustein ist ein Beispiel dafür, wie der Lehrende ein geometrisches Modell anschaulich simulieren kann.

Der Simulation liegt das folgende Thema zugrunde: Aus einem Quadrat mit der Seitenlänge a sollen an dessen Ecken gleich große Quadrate ausgeschnitten werden, so daß man durch Auffalten der Restfläche eine Schachtel erhält, die nach oben hin offen ist. Auf der Zeichenfläche ist das Quadrat mit den Schnittflächen, ein Schrägbild der entsprechenden Schachtel sowie ein Koordinatensystem mit dem Graphen der Funktion $f(x) = b^2x$ dargestellt. Als Einflußgröße kann die Seitenlänge x der zu entfernenden Quadrate verändert werden (Abbildung 4.14). Oberhalb des Schrägbilds werden die numerischen Werte der Parameter a, x, b $(= a - 2x)$ sowie das Volumen der Schachtel angezeigt. Die Frage ist nun, wie verändert sich das Volumen, wenn der Wert für x variiert wird. Der Lehrende kann verschiedene Modellzustände vorführen, indem er die Seitenlänge x der Eckquadrate kontinuierlich von 0 bis $\frac{a}{2}$ variiert. Dabei ändert sich das Ausgangsquadrat mit der Seitenlänge a und das Schrägbild der Schachtel stetig. Für $x = 0$ und $x = \frac{a}{2}$ ist sowohl aus dem Schrägbild als auch durch die numerischen Parameter ersichtlich, daß das Volumen gleich 0 ist. Bei dem stetigen Variieren von x ist zu erkennen, daß es eine Schachtel mit maximalem Volumen gibt. Der Lehrende liest hier den besten Näherungswert ab (Abbildung 4.15).

[2] Schumann 1997

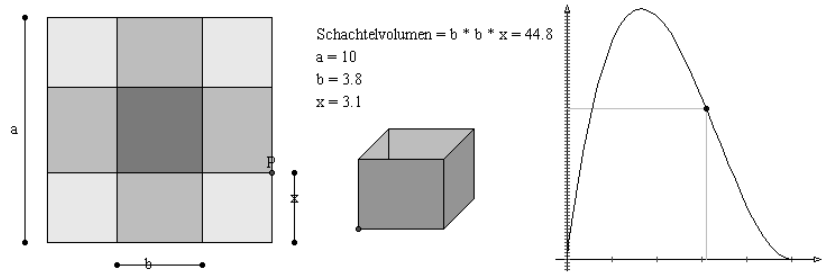

Abbildung 4.14: Simulation der Größe des Volumens einer Schachtel bei Variation des Faltnetzes.

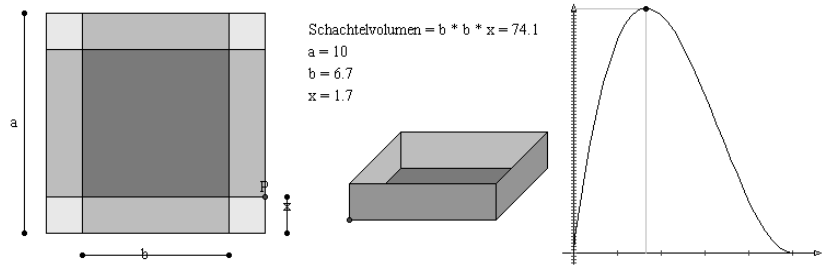

Abbildung 4.15: Bei $x = 1,7$ ist das Volumen der Schachtel maximal.

Beispiel: Schaltkreis Volladdierer

Der folgende Lernbaustein ist ein Beispiel dafür, wie Verknüpfungen aus der Schaltalgebra durch eine Figur simuliert werden können.

Als Volladdierer bezeichnet man eine Schaltung, welche die Addition dreier boolescher Variablen realisiert. Die Eingangsparameter der Figur sind die Variablen a, b, c, deren Werte durch jeweils einen Schalter auf 1 oder 0 gesetzt werden können. Als Ausgangsgrößen gibt es die Summe f_S und den Übertrag $f_{\ddot{U}}$. Die zugehörigen Schaltfunktionen lauten:

$$f_S(a,b,c) := \left(\overline{(a \wedge \overline{b}) \vee (\overline{a} \wedge b)} \wedge c\right) \vee \left(\left((a \wedge \overline{b}) \vee (\overline{a} \wedge b)\right) \wedge \overline{c}\right)$$

und

$$f_{\ddot{U}}(a,b,c) := (a \wedge b) \vee \left(c \wedge \left((a \wedge \overline{b}) \vee (\overline{a} \wedge b)\right)\right).$$

Die Verknüpfungen innerhalb der Schaltfunktionen werden in der Figur durch sogenannte Schaltsymbole dargestellt. Der besondere Wert der Simulation liegt nun darin, daß die an den Schaltsymbolen anliegenden Werte durch Farben symbolisiert werden. Ist das Ergebnis einer Verknüpfung gleich 1, so ist die entsprechende Verbindungsstrecke in der Figur dunkelgrau dargestellt. Ist das Ergebnis gleich 0, so ist die Verbindungsstrecke hellgrau eingefärbt. Diese zusätzliche Visualisierung ermöglicht dem Schüler, die Richtigkeit und den Aufbau der Schaltung leicht nachzuvollziehen.

Die folgenden vier Abbildungen zeigen Modellzustände beim Volladdierer. Sind die Variablen a, b, c allesamt gleich 0, so ist $f_{\ddot{U}} = 0$ und $f_S = 0$. An allen Schaltsymbolen liegt der Wert 0 an. Alle Verbindungsstrecken sind hellgrau eingefärbt (Abbildung 4.16). Ist nur die Variable $a = 1$, so ist $f_{\ddot{U}} = 0$ und $f_S = 1$ (Abbildung 4.17). Sind $a = 1$ und $b = 1$ mit $c = 0$, so ist die Summe $f_S = 0$ und der Übertrag $f_{\ddot{U}} = 1$ (Abbildung 4.18). Sind alle drei Variablen gesetzt, ist $f_{\ddot{U}} = 1$ und $f_S = 1$ (Abbildung 4.19).

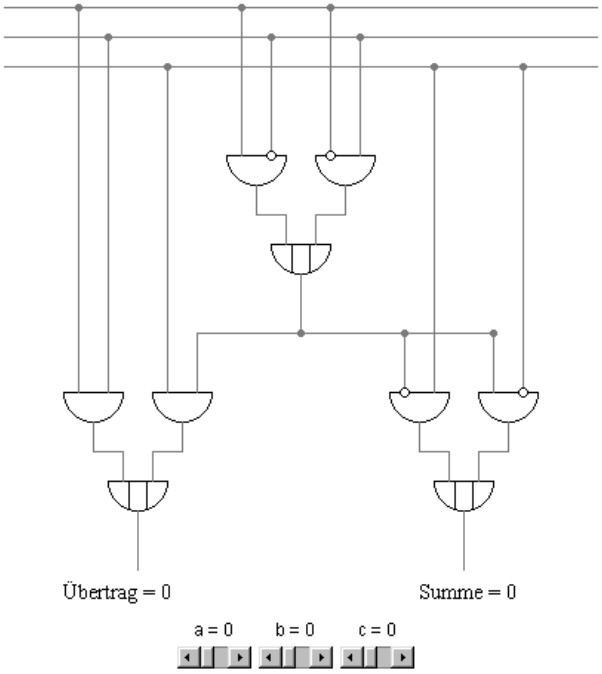

Abbildung 4.16: Volladdierer mit $f_S(0,0,0) = 0$ und $f_{\ddot{U}}(0,0,0) = 0$.

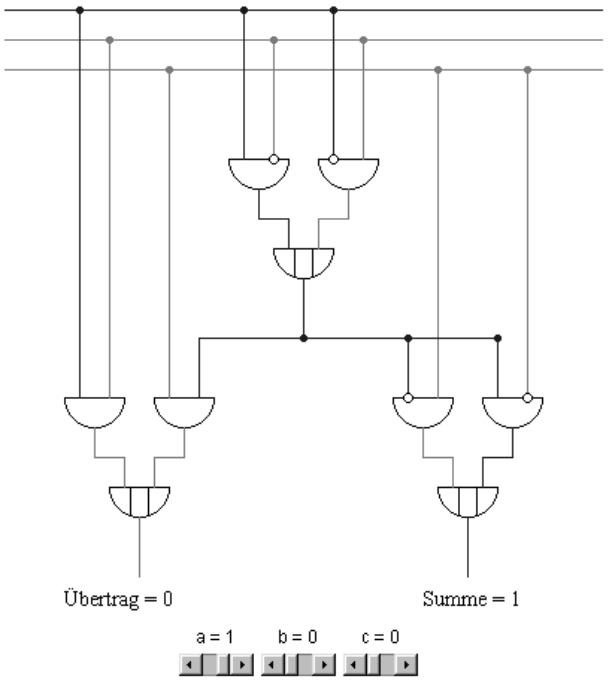

Abbildung 4.17: Volladdierer mit $f_S(1,0,0) = 1$ und $f_{\ddot{U}}(1,0,0) = 0$.

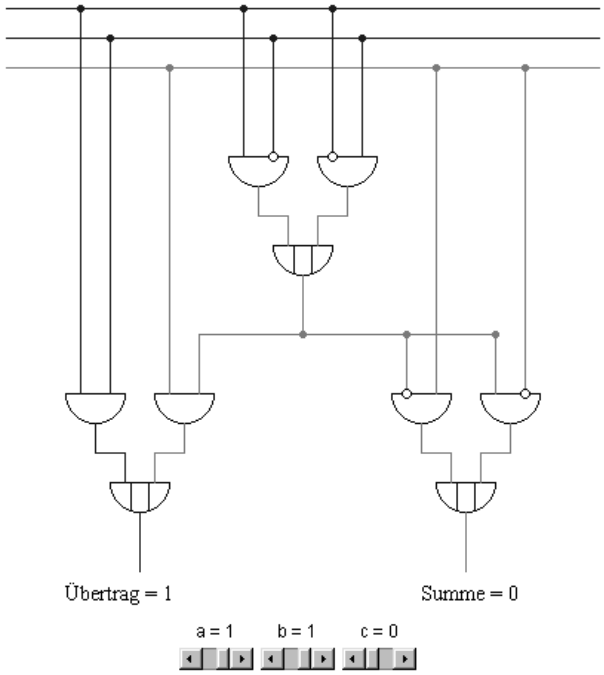

Abbildung 4.18: Volladdierer mit $f_S(1,1,0) = 0$ und $f_{\ddot{U}}(1,1,0) = 1$.

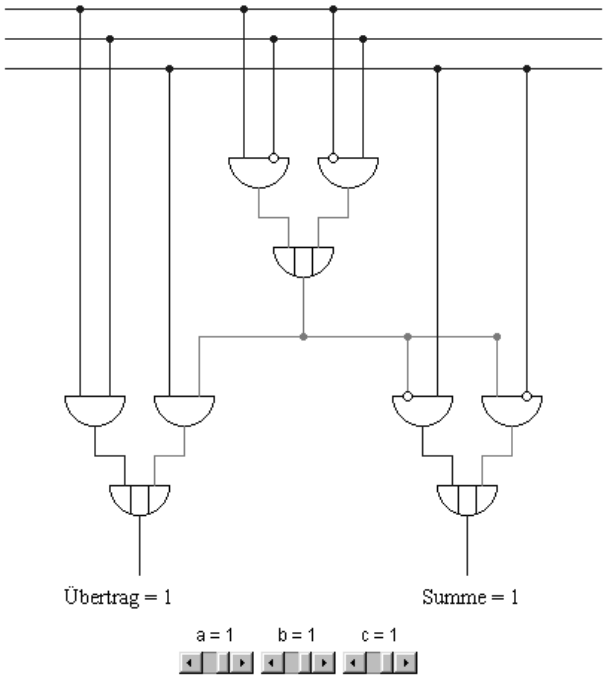

Abbildung 4.19: Volladdierer mit $f_S(1,1,1) = 1$ und $f_{\ddot{U}}(1,1,1) = 1$.

4.2 Lernbausteine für Exploratives Lernen

Im Kontext des Explorativen Lernens dienen Lernbausteine als ein Medium, mit dem der Schüler einen geometrischen Sachverhalt möglichst eigenständig interaktiv erforschen kann. Dazu unterscheide ich im folgenden zwischen Phasen und Aktivitäten bei der Satzfindung (Abschnitt 4.2.1), Beweisfindung (Abschnitt 4.2.2) und Begriffsbildung (Abschnitt 4.2.3).

4.2.1 Satzfindung

Bei der Satzfindung wird dem Schüler durch einen Lernbaustein eine offene Problemstellung angeboten. Seine Aufgabe ist es, mit der Figur zu interagieren und dabei besondere geometrische Sachverhalte zu entdecken. Der Schüler variiert dazu die Figur und stellt Vermutungen auf. Er verifiziert diese und formuliert sie schließlich als einen Satz. Die Bezeichnung Satzfindung ist streng genommen nicht ganz zutreffend, weil es sich bei dem gefundenen Sachverhalt vorerst noch um eine Hypothese handelt, deren Gültigkeit noch nicht bewiesen ist. Weil die Bezeichnung Satzfindung aber in der Literatur[3] gebräuchlich ist, soll sie auch in dieser Arbeit verwendet werden. Als fundamentale Einsichten, die der Schüler

[3] vgl. Schumann 1991

bei der Satzfindung gewinnen soll, nennt Schumann: "Der Schüler soll einsehen, daß

- er Geometrie betreiben kann im Sinne der Wiederentdeckung schon entdeckter und im Rahmen von Begriffs- und Satzgefügen auch bewiesener Sätze;
- es noch viele unentdeckte Sätze in der Geometrie gibt, die unter Umständen auch von ihm entdeckt werden können (obwohl die 'weniger komplexen' Sätze alle schon entdeckt und bewiesen sind)."[4]

In Anlehnung an die Phasen der Problemlösung von Polya[5] kann der Prozeß der Satzfindung durch folgende vier Stufen beschrieben werden.

1. **Verstehen der Aufgabe** In dieser Phase geht es darum, die Aufgabenstellung zu verstehen. Dazu sind die gegebenen und die gesuchten Größen in der Figur zu identifizieren und es ist zu untersuchen, aus welchen Objekten die Figur konstruiert ist. Außerdem ist zu prüfen, welche Relationen zwischen den einzelnen geometrischen Objekten bestehen und welches die Eingangs- und Ausgangsgrößen der Figur sind.
2. **Heuristische Tätigkeiten** Die eigentliche Problemlösung das Finden einer begründeten Hypothese erreicht der Schüler, indem er heuristische Tätigkeiten anwendet. Schreiber[6] unterscheidet vier Gruppen von Heurismen (Induktion, Variation, Interpretation, Reduktion), in denen heuristische Strategien mit gemeinsamen Merkmalen zusammengefaßt sind. Von diesen Heurismen eignen sich zur Satzfindung insbesondere die folgenden Strategien:
   - *Probiere systematisch* Der Schüler probiert systematisch, indem er einzelne, spezielle Figurenzustände überprüft und daran Beobachtungen sammelt. Er verändert gezielt einzelne Eingangsparameter der Figur und hält die übrigen konstant.
   - *Versuche zu verallgemeinern* Der Schüler verallgemeinert die Problemstellung, indem er eine oder mehrere Bedingungen fallen läßt. Eventuell kann er aus konstanten Vorgaben variable Parameter machen.
   - *Variiere das Gegebene* Der Schüler variiert die Figur im Zugmodus und versucht zu erkennen, welche funktionalen Abhängigkeiten bestehen und welche geometrischen Relationen sich als invariant erweisen. Bleiben bestimmte Symmetrien erhalten? Sind Teilfiguren zueinander ähnlich? Bleiben mehrere Geraden parallel oder schneiden sie sich in einem Punkt? Gibt es vier Punkte, die auf einem Kreis liegen?
   - *Variiere den Allgemeinheitsgrad* Um den Allgemeinheitsgrad zu variieren, kann der Schüler generalisieren oder spezialisieren. Das Generalisieren läßt sich dadurch erreichen, daß die Figur von einer speziellen Lage in eine allgemeine Lage verschoben wird. Spezialisieren

[4] Schumann 1991, S. 89
[5] Polya 1949
[6] Schreiber 1999, http://www.uni-flensburg.de/mathe/zero/veranst/heuristik/heuristik-.html

kann man, indem eine Figur von einer allgemeinen Lage in eine besondere Lage (z. B. eine Grenzlage) versetzt wird. In beiden Fällen läßt sich untersuchen, welche Relationen dabei invariant oder nicht invariant bleiben.

- *Variiere die Exaktheitsstufe* Der Schüler variiert die Exaktheitsstufe, indem er anstelle einer exakten Lösung mit Hilfe der Figur graphische Näherungslösungen sucht.

3. **Formulieren einer Hypothese** Der Schüler unterscheidet zwischen interessierenden und nicht interessierenden Vermutungen und untersucht, ob eine vermutete Aussage noch weiter generalisierbar oder spezialisierbar ist. Er versucht zu erkennen, ob seine Hypothese unter- oder übergeneralisiert ist, ob die geometrischen Daten für eine Vermutung ausreichen und welche geometrischen Daten die Vermutung genau festlegen.[7] Der Schüler überprüft seine Vermutung an weiteren speziellen Figurenzuständen und berücksichtigt dabei ggf. auch Fallunterscheidungen. Schließlich formuliert er seine Hypothese in Form einer Wenn-Dann-Aussage oder All-Aussage mit entsprechenden Figuren-, Eigenschafts- und Relationsbezeichnungen.

4. **Rückschau und Einordnung** In dieser Phase soll die aufgestellte Hypothese zu anderen Inhalten der Geometrie in Beziehung gesetzt werden. Es gehört dazu zu prüfen, in welcher Verbindung die Hypothese zu anderen Sätzen steht. Eventuell ist die Hypothese schon als Satz bekannt oder sie ist vielleicht ein Spezialfall von einem schon bekannten Satz. Vielleicht kann auch ein Satz gefunden werden, der ein Spezialfall der gefundenen Hypothese ist.

Beispiel: Flächeninhalt umfangsgleicher Vierecke

In dem folgenden Beispiel wird ein Lernbaustein zur Satzfindung beschrieben, bei dem es darum geht, elementargeometrische Sachverhalte zu entdecken. Die darin umgesetzte isoperimetrische Aufgabe hat Schumann[8] diskutiert. Der Lernbaustein enthält ein Viereck $ABCD$, das in seiner Form und Lage variiert werden kann. Die Seiten lassen sich durch vier aneinandergesetzte Strecken verlängern oder verkürzen. Der Umfang des Vierecks ist durch die Gesamtlänge der Seitenlängen zu erkennen. Ein horizontaler Balken visualisiert den Flächeninhalt des Vierecks. Die Größen Umfang und Flächeninhalt werden außerdem durch numerische Werte angezeigt. Die Seitenlängen, der Gesamtumfang sowie die Lage und Form des Vierecks sind die veränderbaren Eingangsgrößen der Figur (Abbildung 4.20).

Die Aufgabe des Schülers ist es, zu untersuchen, für welche Vierecke der Flächeninhalt maximal ist. Ein erstes Variieren der Eckpunkte (*Variiere das Gegebene*) zeigt an der sich verändernden Länge des Balkens, daß der Flächeninhalt von der Form des Vierecks abhängt. Um systematisch vorzugehen (*Probiere systematisch*), wird zuerst ein Viereck mit vier unterschiedlichen Seitenlängen untersucht. Danach wird dann spezialisiert (*Variiere den Allgemeinheitsgrad*), indem Vierecke mit zwei, drei und vier gleich langen Seiten analysiert werden. Dabei bleibt der Gesamtumfang des Vierecks zunächst konstant.

[7] vgl. Schumann 1991, S. 89
[8] Schumann 1991, S. 225f

Variiert der Schüler nun die Lage und Form eines Vierecks mit vier nicht gleich langen Seiten, so wird der Flächeninhalt maximal, wenn das Viereck ein Sehnenviereck ist (Abbildung 4.21). Um diesen Sachverhalt für den Schüler augenfällig zu machen, ist die Figur so konstruiert, daß der Umkreis des Vierecks genau dann eingeblendet wird, wenn es ein Sehnenviereck ist. Durch systematisches

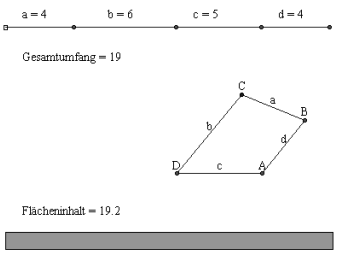

Abbildung 4.20: Für welches Viereck wird der Flächeninhalt maximal?

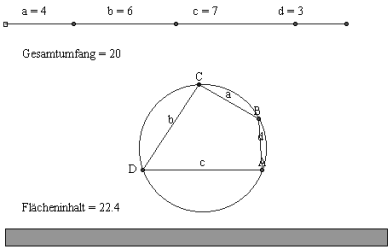

Abbildung 4.21: $ABCD$ ist ein Sehnenviereck.

Variieren kann der Schüler herausfinden, daß bei einem gleichseitigen Drachen mit zwei paarweise gleich langen Seiten der Drachen mit zwei rechten Gegenwinkeln maximalen Flächeninhalt besitzt (Abbildung 4.22). Sind die gegenüberliegenden Seiten paarweise gleich lang, so bildet das Viereck ein Parallelogramm. Der Flächeninhalt wird maximal, wenn das Parallelogramm zum Rechteck wird (Abbildung 4.23). Bei drei gleich langen Seiten wird der Flächeninhalt maximal,

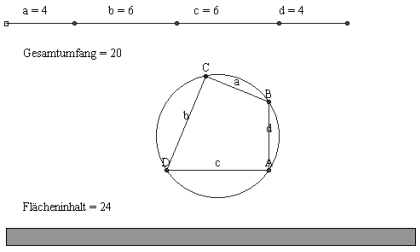

Abbildung 4.22: $ABCD$ ist ein Drachen mit zwei rechten Gegenwinkeln.

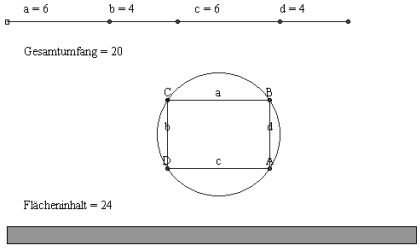

Abbildung 4.23: $ABCD$ ist ein Rechteck.

sobald das Viereck ein Trapez ist (Abbildung 4.24). Sind alle vier Seiten gleich lang, so entsteht eine Raute. Der Flächeninhalt ist maximal, wenn das Viereck zu einem Quadrat variiert wird. Das Quadrat besitzt den größten Flächeninhalt aller Vierecke mit konstantem Umfang (Abbildung 4.25). Der Schüler kann die gefundenen Sachverhalte noch einmal für weitere Viereckssklassen überprüfen, indem er den Gesamtumfang des Vierecks vergrößert oder verkleinert. Als Ergebnis kann schließlich festgehalten werden:

1. Der Flächeninhalt eines Vierecks mit vorgegebenen Seitenlängen wird genau dann maximal, wenn es ein Sehnenviereck ist.

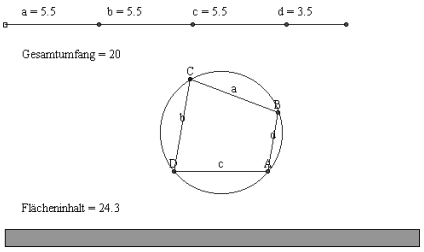

Abbildung 4.24: $ABCD$ ist ein gleichschenkliges Trapez.

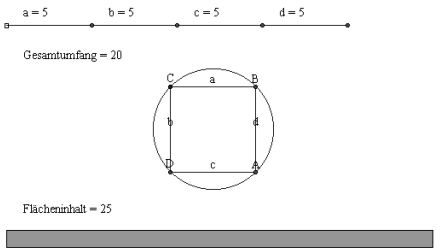

Abbildung 4.25: $ABCD$ ist ein Quadrat.

2. Unter allen umfangsgleichen Vierecken besitzt das Quadrat den größten Flächeninhalt.[9]

Setzt man das Ergebnis mit anderen Sätzen der Geometrie in Beziehung, so zeigt sich, daß die gefundenen Sachverhalte auf das isoperimetrische Problem für Dreiecke zurückgeführt werden können. Dort hat das gleichschenklige Dreieck ABC mit fester Seite AB den größten Flächeninhalt unter allen umfangsgleichen Dreiecken.[10] Das Ergebnis läßt sich auch auf Körper im Raum übertragen: Unter allen achteckigen, ebenflächig begrenzten Körpern mit konstanter Oberfläche besitzt der Würfel das größte Volumen.

Beispiel: Die Picksche Formel

Anhand des folgenden Lernbausteins soll demonstriert werden, wie der Schüler heuristische Aktivitäten bei der Satzfindung von funktionalen Zusammenhängen anwenden kann. Speziell zu diesem Beispiel muß jedoch angemerkt werden, daß das Entdecken der Pickschen Formel komplexe heuristische Tätigkeiten voraussetzt und demnach eine relativ starke Lenkung durch den Lehrenden erfordert.

Die folgenden Ausführungen orientieren sich an Schreiber[11]. Mit Hilfe der Pickschen Formel läßt sich der Flächeninhalt f von Gittervielecken durch die Anzahl i der Gitterpunkte innerhalb eines Vielecks und durch die Anzahl r der Gitterpunkte auf dem Rand eines Vielecks bestimmen. Die Aufgabe des Schülers ist es, einen funktionalen Zusammenhang $f = F(i, r)$ durch experimentelles Arbeiten mit der Figur zu entdecken. Der Lernbaustein enthält ein Dreieck, ein Viereck und ein Fünfeck, dessen Ecken auf den Gitterpunkten verschoben werden können. Zu jedem Vieleck werden mit Hilfe einer externen Klasse (Seite 381) die Werte für i, r und f berechnet und auf der Zeichenfläche angezeigt (Abbildung 4.26).

Ein erstes Experimentieren (*Variiere das Gegebene*) und Überprüfen von einfachen Beispielen (etwa das kleinste Dreieck oder einfache Quadrate) zeigt: Der Wert f ist entweder ganzzahlig oder ein Vielfaches von $\frac{1}{2}$. Anfangs ist jedoch noch nicht offensichtlich, ob die Anzahl der Eckpunkte den Flächeninhalt beeinflußt. Durch Betrachten von Sonderfällen für $i = 1$ und $r = 6$ läßt sich dies jedoch zurückweisen (Abbildung 4.26). Der Schüler könnte vermuten, daß f

[9] Ein Beweis findet sich in Claus 1992, S. 113f.

[10] vgl. Claus 1992, S. 106

[11] Schreiber 1999, http://www.uni-flensburg.de/mathe/zero/veranst/didmath/heuristik-_problemloesen/heuristik_problemloesen.html

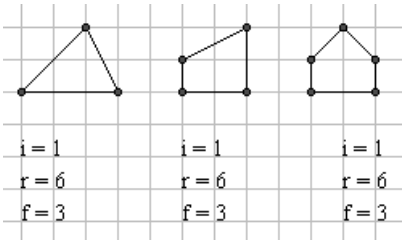

Abbildung 4.26: Verschiedene Vielecke mit gleichen Werten für i, r und f.

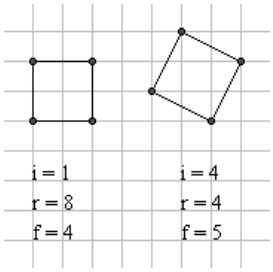

Abbildung 4.27: Wenn $i + r$ steigt, wird i. A. auch f größer. Gibt es Ausnahmen?

zunimmt, wenn $i + r$ wächst (*Probiere systematisch*). Jedoch läßt sich diese Vermutung durch Überprüfen von Sonderfällen widerlegen (Abbildung 4.27). Insgesamt läßt sich noch kein konkreter funktionaler Zusammenhang $f = F(i, r)$ erkennen, deshalb ist es sinnvoll, weiter systematisch vorzugehen (*Probiere systematisch*), indem einzelne Parameter variiert und andere konstant gehalten werden. Hält man zuerst den Wert i konstant und variiert r, dann zeigt sich für $i = 0$: Mit steigendem r steigt auch f (Abbildung 4.28). Ist $i = 1$, dann steigt

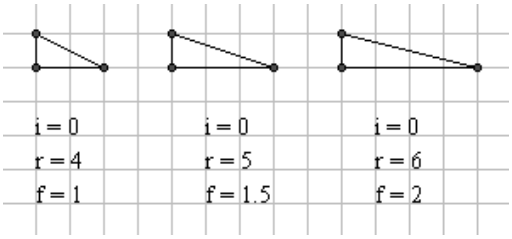

Abbildung 4.28: Wenn i konstant bleibt und r zunimmt, steigt f.

f ebenfalls mit wachsendem r (Abbildung 4.29). Ein linearer Zusammenhang läßt sich vermuten. Mit jedem neuen Randpunkt steigt der Flächeninhalt um $\frac{1}{2}$.

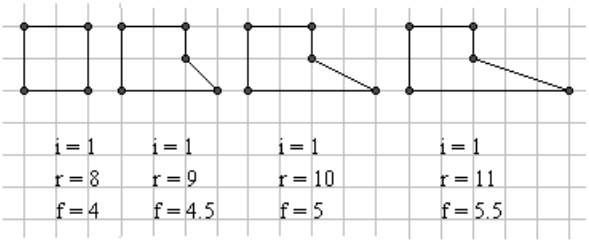

Abbildung 4.29: Ist $i = 1$ und r steigt, dann nimmt auch f zu.

Wenn man den Wert r konstant hält ($r \geq 4$) und $i = 3, 4, 5, \ldots$ betrachtet, deutet sich ebenfalls ein linearer Zusammenhang zwischen i und f an (Abbildung 4.30). Man beobachtet: Mit jedem neuen inneren Punkt scheint f um 1 zu steigen.

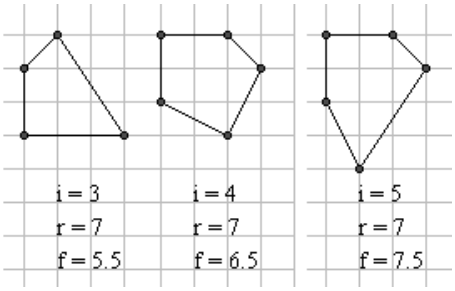

Abbildung 4.30: Ist r konstant und i steigt, dann nimmt f zu.

Ausgehend von den beobachteten linearen Zusammenhängen kann der Schüler versuchen, zu verallgemeinern, indem er annimmt, daß ein Ausdruck für den Flächeninhalt nach der Form $f = F(i, r) = ai + br + c$ ist. Die Variablen a und b sorgen dafür, daß f in einem bestimmten Verhältnis mit i und r wächst. Die Variable c bewirkt eventuell eine Korrektur der Summe $ai + br$. Wenn ein solcher funktionaler Zusammenhang in dieser Form existiert, können die Werte für a, b und c beispielsweise durch Lösen eines Gleichungssystems bestimmt werden. Eine solche Berechnung liefert als Ergebnis $a = 1$, $b = \frac{1}{2}$ und $c = -1$. Durch empirisches Überprüfen der gefundenen Formel kann der Schüler sich von der Richtigkeit überzeugen und als Ergebnis die folgende Formel festhalten: $f = i + \frac{r}{2} - 1$. Die Satzfindung ist damit abgeschlossen, wenngleich ein Beweis damit noch nicht gegeben ist.

4.2.2 Beweisfindung

Während es bei der Satzfindung darum geht, Aussagen über Sachverhalte zu gewinnen, steht bei der Beweisfindung die Sicherung der Allgemeingültigkeit von Aussagen im Mittelpunkt. Lernbausteine eignen sich für Aufgaben zur Beweisfindung deshalb, weil der Schüler beim Variieren der Figur im Zugmodus heuristische Strategien anwenden kann. Damit ist nicht gemeint, daß der Schüler den durch eine Figur vorgegebenen Sachverhalt lediglich empirisch verifiziert. Im Gegenteil: Er soll erkennen, daß die Gültigkeit einer Behauptung zu einer geometrischen Figur nicht bewiesen werden kann, indem man endlich viele Figurenzustände im Zugmodus überprüft. Dabei ist es eine bekannte Tatsache, daß viele Schüler keine Notwendigkeit sehen und wenig motiviert sind, einen Sachverhalt zu beweisen, der durch die Variation der Figur als offensichtlich empfunden wird.[12] Meiner Ansicht nach kann die Motivation eines Schülers durch folgende Vorgehensweisen gesteigert werden:

- Kontrastierende Beispielfiguren sollten den Schüler unterstützen, das Besondere vom Allgemeinen zu unterscheiden und damit den zu beweisenden Sachverhalt auch als etwas Besonderes zu empfinden.
- Die Figur sollte beim Variieren ein verblüffendes oder unerwartetes Verhalten zeigen. Ein solcher Effekt kann erzeugt werden, indem der Schüler angehalten wird, das Verhalten der Figur in einer bestimmten Situation

[12] vgl. Elschenbroich 1997, S. 58

vorauszusagen. Trifft seine Vermutung nicht zu, so kann sich bei ihm ein Überraschungsmoment einstellen, das die Frage aufwirft, warum das so ist.

- Ein Lernbaustein sollte vom Figurenautor so konzipiert werden, daß die Gültigkeit des zu beweisenden Sachverhalts für den Schüler nicht von vornherein evident ist.

Beim Lösungsprozeß von Beweisfindungsaufgaben lassen sich analog zu den Problemlösephasen bei der Satzfindung (Abschnitt 4.2.1) ebenfalls vier Stufen unterscheiden:[13]

1. **Verstehen des Beweisproblems** In dieser Phase geht es darum, den Zusammenhang zwischen Figur und Beweisproblem zu verstehen. Dazu muß man erkennen, welche Voraussetzungen gegeben sind und durch welche konstruktiven Abhängigkeiten und Bewegungseinschränkungen diese in der Figur ausgedrückt werden. Für die Behauptung der Beweisaufgabe ist zu prüfen, durch welche Variationsmöglichkeiten ihre Allgemeingültigkeit in der Figur ausgedrückt wird.

2. **Heuristische Tätigkeiten** In dieser Phase gilt es, die Behauptung aus den Voraussetzungen zu folgern. Dazu kann man folgende heuristische Tätigkeiten zur Beweisfindung anwenden:

   - *Arbeite vorwärts* Ausgehend von den Voraussetzungen lassen sich solange Folgerungen ziehen, bis aus diesen die Behauptung deduziert werden kann.
   - *Arbeite rückwärts* Bei dieser Strategie geht man von der Behauptung aus und sucht nach einem Satz, aus dem die Behauptung gefolgert werden kann. Falls dieser Satz nicht aus den Voraussetzungen direkt geschlossen werden kann, dient er als neuer Ausgangspunkt, von dem aus man rückwärts weiterarbeitet.
   - *Unterscheide Fälle* Durch das Treffen von Fallunterscheidungen zerlegt man das Beweisproblem in mehrere einzelne Probleme. Eventuell kann man die Behauptung für einen Spezialfall beweisen und den allgemeinen Fall auf den Spezialfall zurückführen.
   - *Variiere das Gegebene* Das Variieren der Figur im Zugmodus kann sich unterstützend beim Aufspüren einer Beweisidee auswirken. Indem man funktionale Abhängigkeiten ausmacht, invariante geometrische Relationen erkennt oder einfach nur verschiedene Ausprägungen der Figur aufnimmt, kann in – im besten Fall – eine Beweisidee geweckt werden.

3. **Dokumentation des Beweises** Nachdem eine Beweisidee inspiriert worden ist, soll der Beweis ausformuliert und dokumentiert werden. Dabei kann noch einmal überprüft werden, ob alle Beweisschritte auch gültig sind. Der Beweis muß von anderen Personen nachvollzogen und als korrekt anerkannt werden können. Bezeichnungen aus der Figur können beim Notieren verwendet werden.

[13] vgl. Holland 1996, S. 109-122

4. **Rückschau und Einordnung** In der letzten Phase geht es darum, sich noch einmal die entscheidenden Lösungsideen bewußt zu machen und zu überlegen, welche der heuristischen Tätigkeiten angewendet wurden. Man kann prüfen, ob sich der Beweis in einen allgemeinen Rahmen einfügen läßt. Gegebenenfalls läßt sich im Nachhinein noch eine einfachere Begründung oder eine anschaulichere Darstellungsform für den Beweis finden.

Beispiel: Beweisaufgabe zum Höhenfußpunktdreieck

Der im folgenden diskutierte Lernbaustein zeigt eine Aufgabe zur Beweisfindung, die in erster Linie durch die heuristischen Strategien *Arbeite vorwärts* und *Variiere das Gegebene* gelöst werden kann. Den Beweis hat Holland[14] beschrieben.

Gegeben ist ein Dreieck ABC, dessen Eckpunkte frei innerhalb der Zeichenfläche bewegt werden können. Zu zeigen ist, daß die Höhen des Dreiecks ABC zugleich die Winkelhalbierenden des Höhenfußpunktdreiecks PQR sind. Voraussetzung sind: $AP \perp BC$, $BQ \perp CA$, $CR \perp AB$, und die Höhen schneiden sich in dem Punkt H (Abbildung 4.31). Ein Variieren der Punkte A, B, C

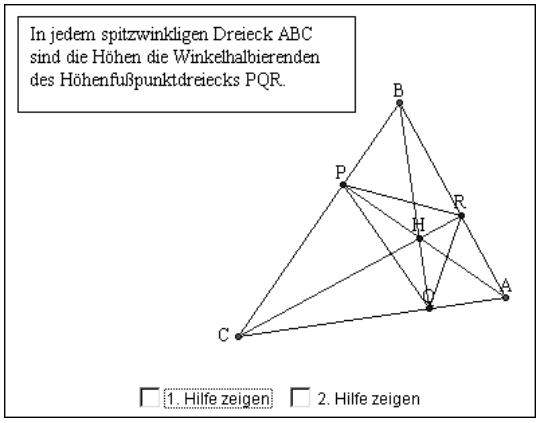

Abbildung 4.31: Beweisaufgabe zum Höhenfußpunktdreieck.

scheint die Behauptung zu bestätigen. Sie ist aber auf den ersten Blick keineswegs evident. Aus den Voraussetzungen kann gefolgert werden, daß die Vierecke $ARHQ$ und $RBPH$ Sehnenvierecke sind. Als Hilfestellung für den Schüler lassen sich die entsprechenden Kreise einblenden (Abbildung 4.32). Betrachtet man das Sehnenviereck $ARHQ$, so können nach dem Peripheriewinkelsatz mehrere gleich große Winkel gefunden werden, insbesondere ist $\angle RAH = \angle RQH$. Im Sehnenviereck $CQHP$ ist $\angle HCP = \angle HQP$. Wenn nun noch gezeigt werden könnte, daß $\angle HCP = \angle RAH$ ist, wäre die Behauptung für die Winkelhalbierende durch Q bewiesen. Um dieses zu zeigen, betrachtet man das Viereck $ARPC$, das ebenfalls ein Sehnenviereck ist. Die Punkte R und P liegen auf dem Thaleskreis der Strecke CA. Als Hilfestellung kann der Schüler sich den

[14] Holland 1996, S. 109f

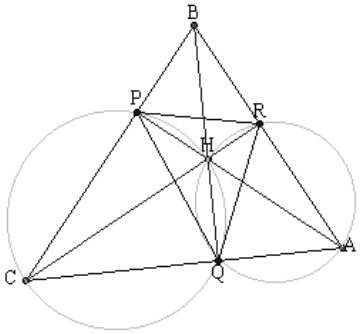

Abbildung 4.32: Die Vierecke $ARHQ$ und $RBPH$ sind Sehnenvierecke.

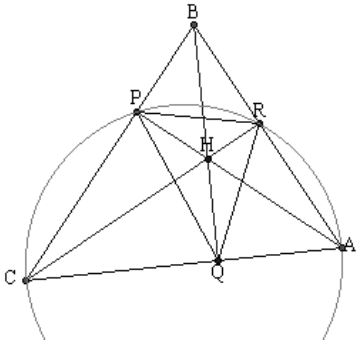

Abbildung 4.33: Das Viereck $ARPC$ ist ein Sehnenviereck.

entsprechenden Umkreis einblenden (Abbildung 4.33). Betrachtet man die Sehne PR des Umkreises, so ist nach dem Peripheriewinkelsatz $\angle HCP = \angle RAP$ und auch $\angle HQP = \angle RQH$. Die Beweisschritte sollen noch einmal ausführlich notiert werden:

1. $AP \perp BC$ (Voraussetzung).
2. $BQ \perp CA$ (Voraussetzung).
3. $CR \perp AB$ (Voraussetzung).
4. Die Höhen schneiden sich in dem Punkt H (Voraussetzung).
5. $ARHQ$ ist ein Sehnenviereck (2, 3).
6. $\angle RAH = \angle RQH$ (Peripheriewinkelsatz).
7. $CQHP$ ist ein Sehnenviereck (1, 3).
8. $\angle HCP = \angle HQP$ (Peripheriewinkelsatz).
9. $ARPC$ ist ein Sehnenviereck (1, 2, Thalessatz).
10. $\angle HCP = \angle RAP$ (Peripheriewinkelsatz).
11. $\angle HQP = \angle RQH$ (Peripheriewinkelsatz).
12. QB ist die Winkelhalbierende von $\angle RQP$.

Für die beiden Höhen AP und CR ist der Beweis analog zu führen.

Beispiel: Beweisaufgabe zur Lotsumme im gleichseitigen Dreieck

Der folgende Lernbaustein ist ein Beispiel für eine Aufgabe zur Beweisfindung, die in erster Linie durch die heuristischen Strategien *Unterscheide Fälle* und *Variiere das Gegebene* gelöst werden kann. Die Figur orientiert sich an einer Konstruktion von Elschenbroich[15].

Gegeben ist ein gleichseitiges Dreieck ABC, das in Größe und Lage variiert werden kann. Innerhalb des Dreiecks kann der Punkt P verschoben werden. Ausgehend von dem Punkt P sind die drei Lote x, y, z zu den Dreiecksseiten eingezeichnet (Abbildung 4.34). Mit Hilfe zweier Schaltelemente kann der Schüler zusätzliche Hilfslinien einblenden. Diese verlaufen durch den Punkt P und sind jeweils paarweise parallel zu einer Dreiecksseite. Die zugehörige Auf-

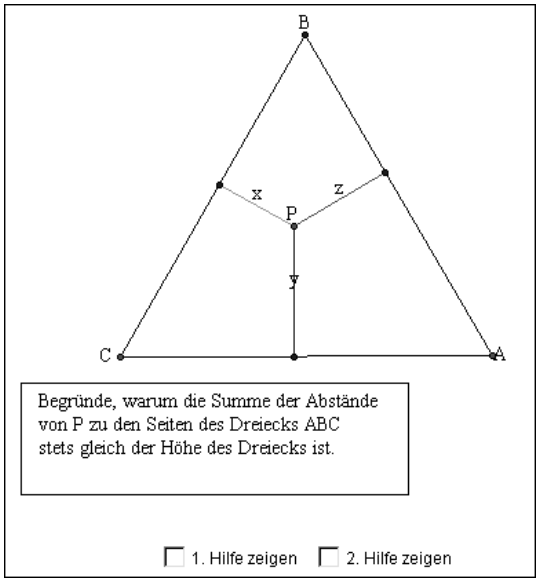

Abbildung 4.34: Beweisaufgabe zur Lotsumme im gleichseitigen Dreieck.

gabenstellung lautet: Begründe, warum die Summe der Abstände von P zu den Seiten des Dreiecks stets gleich der Dreieckshöhe h ist. Verwende die Hilfslinien, um einen Beweis zu finden.

Die Figur ist mit Absicht so konstruiert, daß die Gültigkeit dieses Satzes für den Schüler nicht evident ist. Indem er sich klarmacht, warum der Satz gilt, entwickelt der Schüler einen Beweis. Durch die Variation des Punkts P sieht man, daß die Lote x, y, z sich verändern. Ob dabei die Gesamtlänge konstant bleibt, kann man nur durch Abschätzen vermuten. Ein erster Anhaltspunkt zur Lösung bietet sich dem Schüler, indem er eine augenfällige Grenzlage herstellt. Inzidiert der Punkt P mit B, dann wird die Strecke y zur Symmetrieachse des Dreiecks und $y = h$ (mit $x = 0$ und $z = 0$). Für diesen Spezialfall ist die Behauptung offensichtlich wahr (Abbildung 4.35).

Ein weiterer Spezialfall ergibt sich, wenn P auf einer Dreiecksseite, etwa auf AC, liegt. Hier lohnt es sich, die Hilfslinien einzublenden und mit in die Be-

[15] Elschenbroich 1999, S. 61f

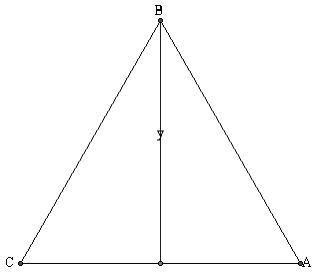

Abbildung 4.35: Spezialfall $y = h$.

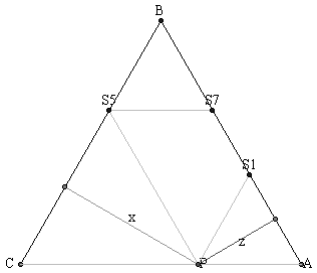

Abbildung 4.36: Spezialfall $x + z = h$.

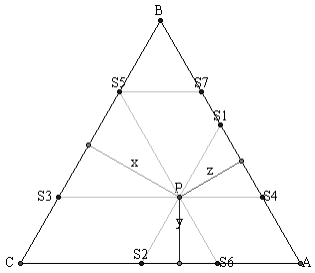

Abbildung 4.37: Allgemeiner Fall $x + y + z = h$.

trachtung einzubeziehen. Es lassen sich drei Dreiecke erkennen: CPS_5, PAS_1 und S_5S_7B. Diese sind gleichseitig, weil die Hilfslinien jeweils parallel zu einer der Dreiecksseiten von ABC verlaufen. Außerdem ist PAS_1 kongruent zu S_5S_7B. Es gilt also $x + z = h$ (mit $y = 0$). Analog gilt dies auch, falls P mit der Seite AB oder BC inzidiert (Abbildung 4.36).

Von diesem speziellen Fall kann man zum allgemeinen Fall gelangen, indem die Hilfslinien erneut betrachtet werden (Abbildung 4.37). In dem Dreieck S_3S_4B liegt P wieder auf der Grundseite und der Schüler kann erkennen, daß die Lotsumme für dieses Dreieck gleich $x+z$ ist. Für das Ausgangsdreieck ABC ist dann die Lotsumme $x + y + z = h$. Damit ist eine Beweisidee gefunden, die nun nur noch besser ausgearbeitet werden muß.
Voraussetzung: ABC ist gleichseitig, und P liegt innerhalb des Dreiecks.
Behauptung: $x + y + z = h$.

1. Beweis des Spezialfalls "P liegt auf AB": Aus PAS_1 kongruent zu S_5S_7B und $y = 0$, folgt: $x + z = h$.

2. Beweis des allgemeinen Falls: Den allgemeinen Fall kann man auf den Spezialfall zurückführen. Betrachtet man das gleichseitige Dreieck S_3S_4B, dann gilt für dieses Dreieck der Spezialfall $h' = x + z$. Somit folgt für den allgemeinen Fall $h = h' + y = x + y + z$.

4.2.3 Begriffsbildung

Die (geometrische) Begriffsbildung ist eine fundamentale Aufgabe des Geometrieunterrichts. Um die Frage zu beantworten, welche Rolle Lernbausteine dabei spielen, soll skizziert werden, was ich unter Begriffsbildung verstehe.

Bender & Schreiber[16] beschreiben das Prinzip der operativen Begriffsbildung, indem sie davon ausgehen, daß Begriffe durch Ideation gebildet werden. Ideation bedeutet das Hineinsehen von Eigenschaften in ein Ding. Die Autoren formulieren das Prinzip der operativen Begriffsbildung wie folgt: "Geometrische Begriffe sind operativ zu bilden, d. h.: Von bestimmten Zwecken ausgehend werden Normen zur Herstellung von Formen entwickelt, die jene Zwecke erfüllen. Die Normen, zumeist Homogenitätsforderungen, werden in Handlungsvorschriften zu ihrer exhaustiven Realisierung umgesetzt und sind damit inhaltliche Grundlage der ihnen entsprechenden Begriffe."

Bezogen auf das Prinzip der operativen Begriffsbildung spielen Lernbausteine im wesentlichen keine große Rolle, denn sie liefern weder einen Beitrag zur Zweckanalyse, noch läßt sich ein Realisat durch eine Figur herstellen. Vor diesem Hintergrund möchte ich Begriffsbildung weiter fassen (wie es auch Bender & Schreiber tun):

1. Verschiedene Ausprägungen und Darstellungsformen eines Begriffs kennenlernen.
2. Den Begriff in ein Begriffssystem einbetten und dabei die Beziehungen zu anderen Begriffen herausstellen.
3. Das Begriffssystem durch Sätze über den Begriff weiterentwickeln.
4. Probleme im Zusammenhang mit dem Begriff lösen.

Diese Aktivitäten dienen dazu, das Verständnis in einen bereits gebildeten Begriff zu vertiefen und ihn mit dem bisherigen Wissen in Beziehung zu setzen. Hier lassen sich Lernbausteine sinnvoll einsetzen. Der Schüler kann einen durch eine Figur dargestellten Begriff selbsttätig variieren und durch die unterschiedlichen Lagen der Figur verschiedene Ausprägungen und Repräsentanten kennenlernen. Durch alternative Darstellungsformen einer Figur (visuell, numerisch, symbolisch) kann ein Begriff auf verschiedene Arten veranschaulicht werden. Die Figur kann dabei so konstruiert sein, daß sie bei Variation stets ein positives Beispiel für einen Begriff ist. Sie kann aber auch so konzipiert sein, daß sie nur in speziellen Lagen ein Repräsentant eines Begriffs ist. Durch Texteinblendungen während der Variation kann der Schüler auf besondere Ausprägungen eines Begriffs aufmerksam gemacht werden.

Weil eine Figur stets aus einer gewissen Anzahl von Objekten besteht, ist der durch die Figur repräsentierte Begriff gleichzeitig in ein Begriffssystem eingebettet. Wie bereits in Abschnitt 4.2.1 erwähnt, eignen sich Lernbausteine zum Arbeiten mit und Finden von geometrischen Sätzen. Außerdem kann das Lösen von Problemen mit Hilfe einer Figur zur Begriffsbildung beitragen. Lernbausteine unterstützen die Tätigkeiten des Schülers vor allem durch Möglichkeiten wie:

[16] Bender & Schreiber 1985, S. 26f

- das Realisieren von kinematischen Zusammenhängen (etwa in der Abbildungsgeometrie, bei Simulationen oder durch Ortsspuren abhängiger Punkte),
- das Darstellen von funktionalen Zusammenhängen oder
- das Aufzeigen von Invarianten bei der Variation im Zugmodus.

Dabei ist zu beachten, daß die Figur in einem Lernbaustein immer ein Modell darstellt, das zwischen der idealen Geometrie und der Realität vermittelt. Deshalb sollte insbesondere beim Einsatz von Lernbausteinen zur Begriffsbildung immer darauf geachtet werden, daß die Unterschiede zwischen idealer Geometrie, geometrischem Modell und Wirklichkeit herausgestellt werden.

Beispiel: Der Begriff Ableitung

Der folgende Lernbaustein soll dem Schüler die visuellen Aspekte des Begriffs Ableitung vermitteln. Die Figur entspricht in Teilen einem Java-Applet, das von Embacher entwickelt und in einem Artikel[17] diskutiert wurde. In dem Lernbaustein ist der Graph der Funktion $f(x) = \frac{x^5}{300} - \frac{11x^3}{60} + \frac{3x}{2}$ dargestellt (Abbildung 4.38). Auf dem Graphen kann ein Punkt A und die durch A verlaufende Tan-

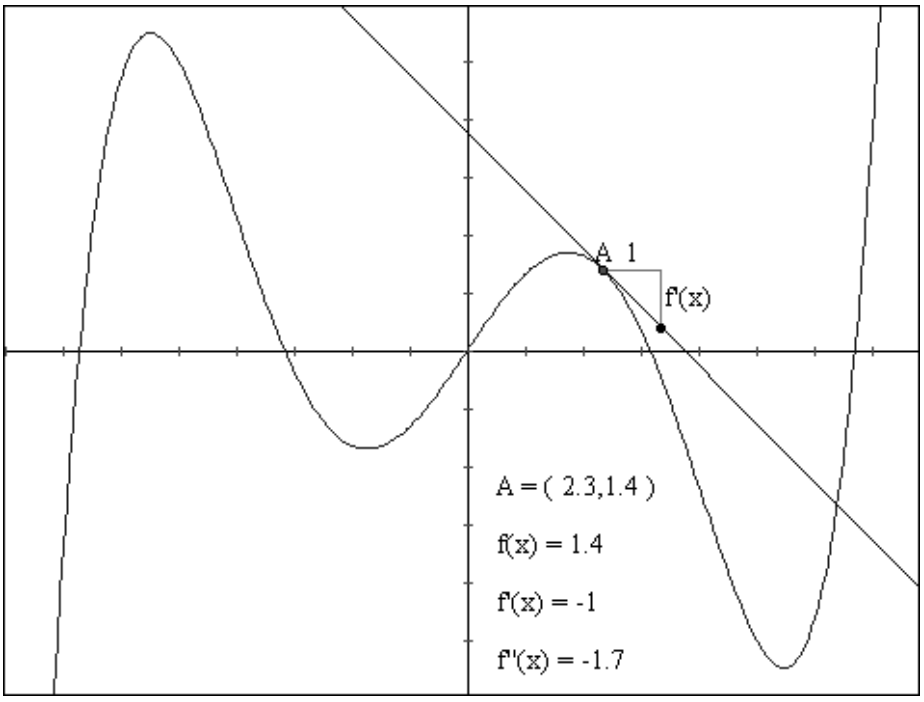

Abbildung 4.38: Lernbaustein zum Begriff der Ableitung.

gente verschoben werden. Der Begriff Ableitung soll als Anstieg der Tangente an einen Graphen eingeführt werden. Der Anstieg kann in dem Koordinatensystem

[17] Embacher 1998, S. 71f

nach der Regel "1 nach rechts, k hinauf oder $-k$ hinunter" abgelesen werden. In der Figur ist diese Strecke mit $f'(x)$ bezeichnet. Als numerische Werte werden u. a. die Koordinaten von A und der Wert $f'(x)$ angezeigt.

Für den Schüler bietet der Lernbaustein die Möglichkeit, den Punkt A zu verschieben und dabei verschiedene Werte der Ableitung zu beobachten. Der Wert der Ableitung ist negativ, wenn $f(x)$ mit zunehmenden x abfällt. Der Wert der Ableitung ist positiv, wenn $f(x)$ mit zunehmenden x ansteigt. Beim Übergang zwischen diesen beiden Zuständen nimmt die Ableitung den Wert 0 an. Offensichtlich ist $f'(x)$ genau an den Stellen des Graphen gleich 0, an denen ein lokales Maximum oder ein lokales Minimum vorliegt. Der Schüler wird auf diesen Sachverhalt durch das Einblenden eines Textfensters aufmerksam gemacht (Abbildung 4.39).

Betrachtet man den Zusammenhang zwischen der Lage der Tangente und dem Wert der Ableitung, so ist augenfällig, daß die Tangente genau dann horizontal verläuft, wenn $f'(x) = 0$ ist. Dieser Sachverhalt, der durch die Figur schon beinahe trivial erscheint, ist jedoch ein wichtiger Schritt, um das algebraische Verfahren zum Lösen von Extremwertaufgaben zu verstehen. Um den Schüler problemorientiert an den Begriff heranzuführen, können ihm beispielsweise folgende Aufgaben gestellt werden:

- Den Wert der Ableitung beispielsweise an den Stellen $x = 0$, $x = 1$ und $x = 2$ bestimmen. Dabei kann der Schüler den Wert der Ableitung an der numerischen Darstellung von $f'(x)$ ablesen, oder er kann die Größe der Ableitung mit Hilfe des Koordinatenrasters abschätzen.

- Alle Stellen des Graphen zu finden, an denen die Ableitung den Wert 1 hat. Zur Lösung verschiebt der Schüler den Punkt A auf dem Funktionsgraphen und verfolgt die stetige Lageveränderung der Tangente sowie die Änderung des Ableitungswerts.

- Alle Stellen des Graphen zu finden, an denen die Ableitung den Wert 0 besitzt. Der Schüler kann erkennen, daß die Ableitung genau dann den Wert 0 annimmt, wenn die Tangente horizontal verläuft. An diesen Stellen besitzt die Funktion ein lokales Minimum oder Maximum (Abbildung 4.40).

- Genau die Intervalle zu bestimmen, in denen die Ableitung mit wachsendem x steigt oder fällt. Beim Lösen dieser Aufgabe verschiebt der Schüler den Punkt A so, daß der x-Wert steigt. Dabei beobachtet er den Wert der Ableitung. Der Begriff Wendepunkt wird ihm bei den entsprechenden Figurenzuständen durch ein Textfenster angezeigt (Abbildung 4.39). Durch diese Übung kann der Schüler den Zusammenhang zwischen Krümmung des Funktionsgraphen und dem Ansteigen oder Abfallen des Ableitungswerts erkennen.

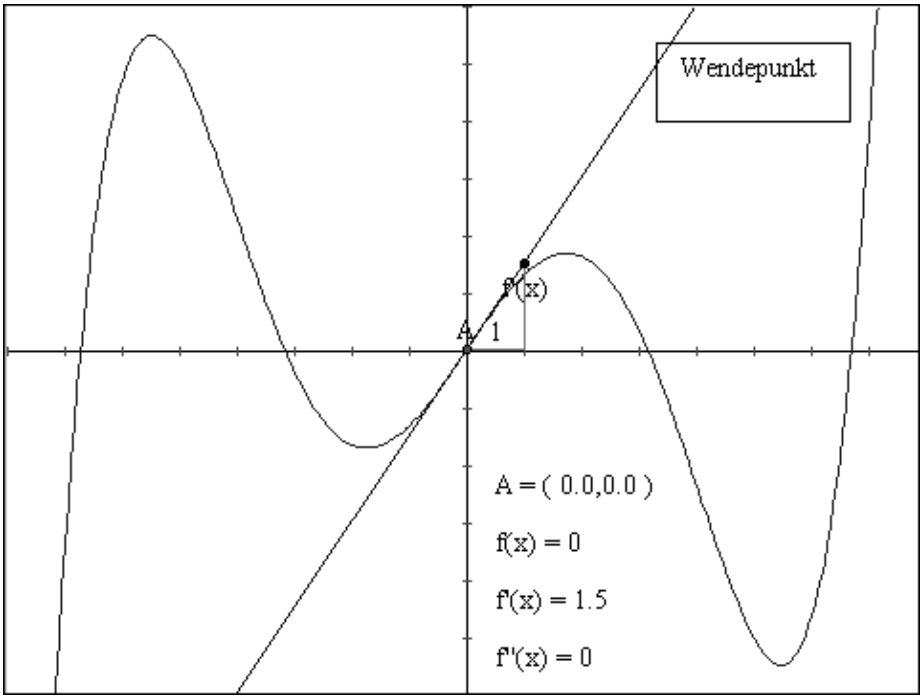

Abbildung 4.39: Texteinblendungen bei Wendepunkten.

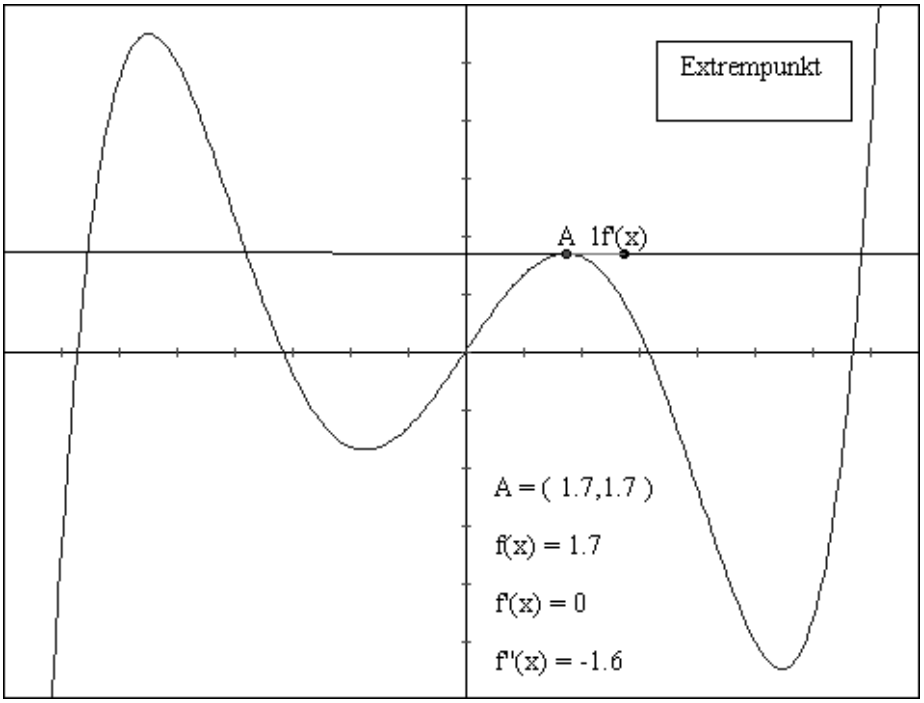

Abbildung 4.40: Texteinblendungen bei Extremwerten.

Beispiel: Begriff Evolute

Im folgenden werden drei Lernbausteine zum Begriff der Evolute vorgestellt. Dazu ist als Figur einmal

$$\text{eine Parabel}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} t \\ at^2 + bt + c \end{pmatrix},$$

$$\text{eine Ellipse}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a\cos t \\ b\sin t \end{pmatrix} \text{ und}$$

$$\text{eine Zykloide}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} rt - a\sin t \\ r - a\cos t \end{pmatrix} \text{ gegeben.}$$

Zu jedem dieser Kurvenobjekte lassen sich die Kurvenparameter durch Schieberegler auf spezielle Werte einstellen, so daß jeweils eine Kurvenschar betrachtet werden kann. Auf jeder Kurve kann ein Punkt T bewegt werden, zu dem die Tangente, die Normale sowie der entsprechende Krümmungskreis angezeigt wird. Mit Hilfe eines Ortslinienobjekts ist die Bahn des Mittelpunkts des Krümmungskreises dargestellt. Der Schüler lernt durch Variation der folgenden Figuren verschiedene Ausprägungen von Evoluten kennen. Dabei wird der Begriff Evolute eng mit den Begriffen Tangente, Normale und Krümmungskreis verbunden.

Betrachtet man die Evoluten von Parabeln mit unterschiedlichen Parameterwerten, so kann man erkennen, daß nur der Parameter a die Form der Evolute beeinflußt. Je kleiner der Betrag von a ist, desto spitzer verläuft die Evolute (Abbildungen 4.41 und 4.42). Die Parameter b und c bewirken lediglich eine Lageverschiebung im Koordinatensystem.

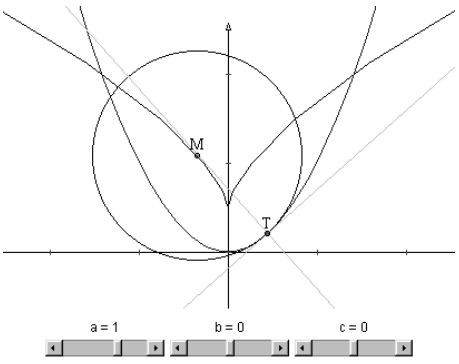

Abbildung 4.41: Evolute bei einer Normalparabel.

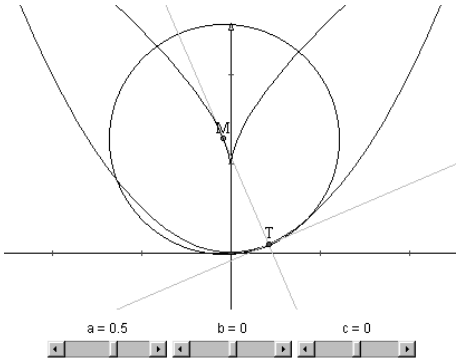

Abbildung 4.42: Evolute bei einer Parabel der Form $y = \frac{1}{2}x^2$.

Untersucht man die Evoluten von verschiedenen Ellipsen, so läßt sich vermuten, daß es sich stets um eine Astroide handelt. Dieser Sachverhalt kann durch spätere analytische Behandlung verifiziert werden (Abbildungen 4.43 und 4.44). Für $a = b$ stellt die Ellipse einen Kreis dar, und die Evolute entartet zum Kreismittelpunkt (Abbildung 4.45).

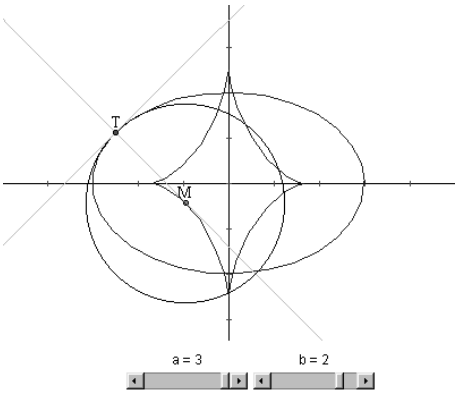

Abbildung 4.43: Die Evolute einer Ellipse ist eine Astroide.

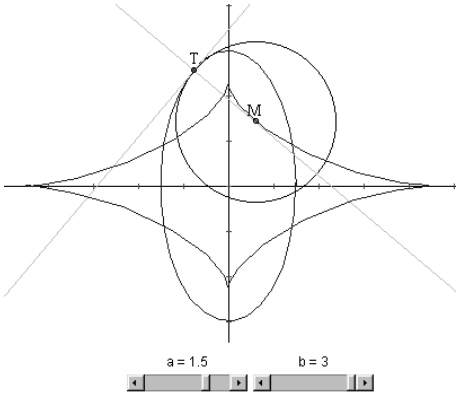

Abbildung 4.44: Evolute einer weiteren Ellipse.

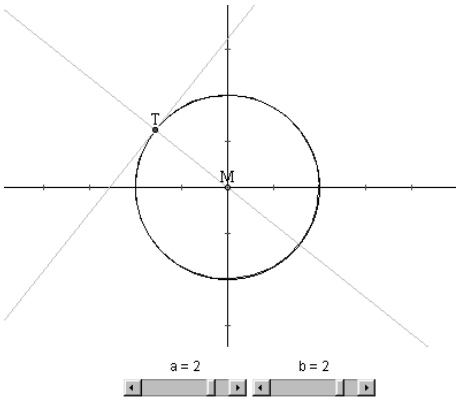

Abbildung 4.45: Beim Kreis entartet die Evolute zum Punkt.

Die Evoluten einer gestreckten ($a < r$), einer spitzen ($a = r$) und einer geschlungenen ($a > r$) Zykloide sind stets zur Ausgangskurve kongruent und lediglich um den Vektor $\binom{\pi r}{2r}$ verschoben (Abbildungen 4.46 - 4.48). Dieser Sachverhalt läßt sich durch eine analytische Untersuchung bestätigen.[18]

[18] vgl. Schupp & Dabrock 1995, S. 193f

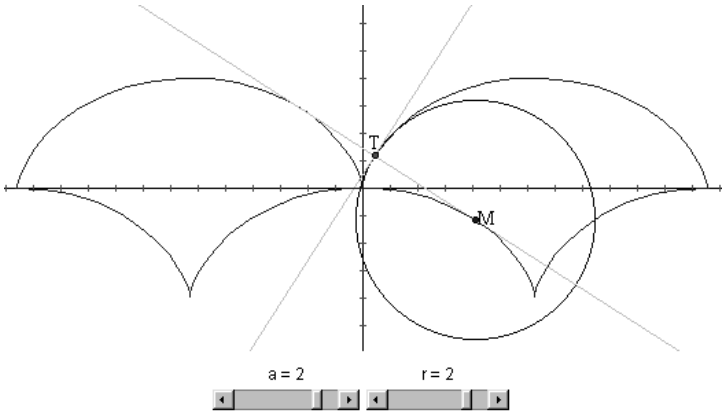

Abbildung 4.46: Evolute einer spitzen Zykloide.

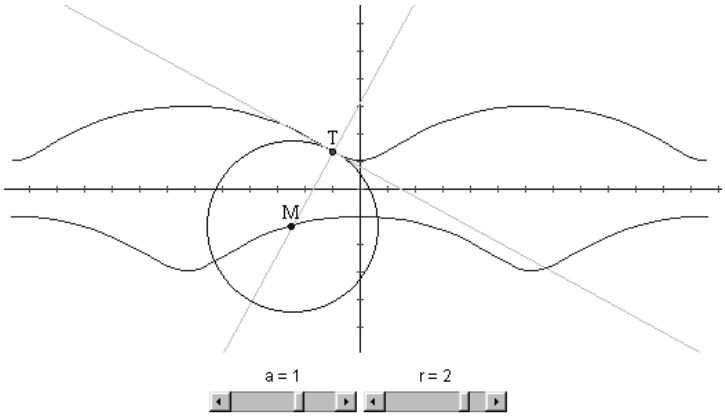

Abbildung 4.47: Evolute einer gestreckten Zykloide.

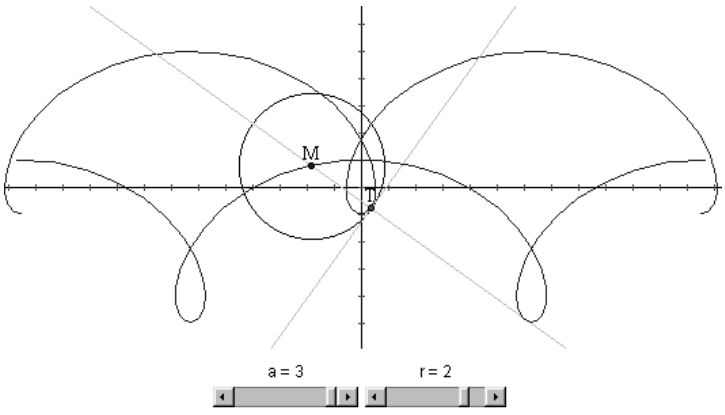

Abbildung 4.48: Evolute einer geschlungenen Zykloide.

4.3 Lernbausteine zur Selbstkontrolle

Lernbausteine zur Selbstkontrolle sollen dem Schüler die Möglichkeit geben, eingestreut innerhalb eines interaktiven Lehrtexts oder zum Abschluß eines Online-Kurses im Internet seinen Wissensstand zu testen. Der Schüler kann dadurch eigene Wissensdefizite erkennen und diese durch gezieltes Nacharbeiten beseitigen. Neben der kontrollierenden Funktion haben Lernbausteine zur Selbstkontrolle noch eine weitere Aufgabe: Sie lockern den Inhalt eines Lehrtexts auf, indem sie eine kleine Pause beim Durcharbeiten bieten und die Konzentration auf den Inhalt fördern. Die erfolgreich bewältigten Aufgaben motivieren den Schüler weiterzuarbeiten.

Im folgenden unterscheide ich – nach zunehmenden Komplexitätsgrad – drei Typen von Lernbausteinen zur Selbstkontrolle: Verständnisfragen (Abschnitt 4.3.1), einfache Variationsaufgaben (Abschnitt 4.3.2) und komplexe Variationsaufgaben (Abschnitt 4.3.3).

4.3.1 Verständnisfragen

Bei diesem Lernbausteintyp wird dem Schüler eine Verständnisfrage zu einer Figur und dem damit verbundenen geometrischen Sachverhalt gestellt. Der Schüler überlegt sich die Antwort und läßt sich die Aufgabenlösung anzeigen. Eine automatische Bewertung seiner Antwort wird bei dieser Form von Selbstkontrolle nicht vorgenommen. Hier muß der Schüler ehrlich sich selbst gegenüber sein und prüfen, ob seine Lösung richtig oder falsch war.

Beispiel: Satz von Varignon

Das folgende Beispiel zeigt einen Lernbaustein mit einer Verständnisfrage. Er stammt aus dem Online-Skript zur Elementargeometrie (Kapitel 5). Gegeben ist ein bewegliches Parallelogramm (Abbildung 4.49). Zu dem Satz von Varignon wird folgende Verständnisfrage gestellt: Wieviele Ausgangsvierecke gibt es zu einem gegebenen Varignon-Parallelogramm? Der Schüler überlegt sich zu

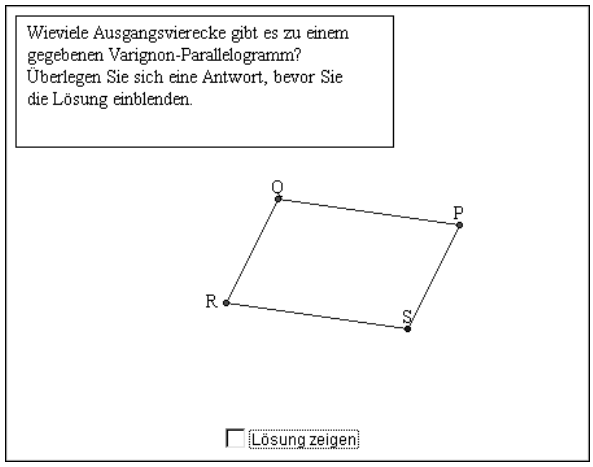

Abbildung 4.49: Lernbaustein mit Verständnisfrage.

der Frage eine Antwort und blendet durch Anklicken der Checkbox "Lösung zeigen" den Lösungskommentar und die Lösungsfigur ein (Abbildung 4.50). Die

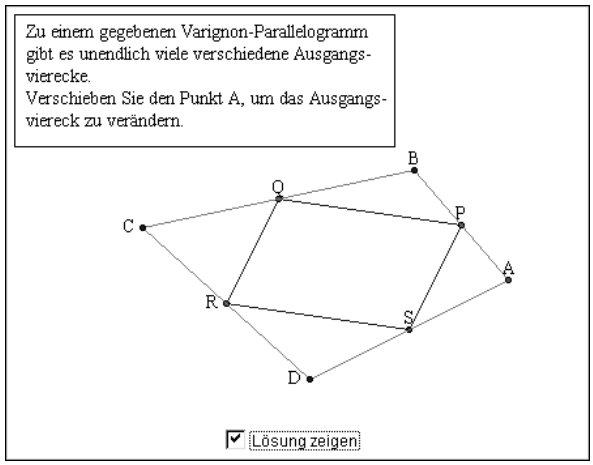

Abbildung 4.50: Lösungsfigur mit Schlußkommentar.

Lösung lautet in diesem Beispiel: Zu einem gegebenen Varignon-Parallelogramm gibt es beliebig viele Ausgangsvierecke. Der Schüler kann diese Aussage prüfen, indem er das Ausgangsviereck so variiert, daß das Varignon-Parallelogramm unverändert bleibt.

Ein weiteres Beispiel für einen Lernbaustein mit einer Verständnisfrage ist die Aufgabe zum Verhältnis zweier Quadratflächen (Seite 139).

4.3.2 Einfache Variationsaufgaben

Unter einer einfachen Variationsaufgabe verstehe ich eine Aufgabe mit einer Antwortanalyse, bei der eine Ein-Punkt-Inzidenz herzustellen ist. Der Schüler soll die gegebene Figur so variieren, daß ein spezieller Punkt bestimmte Eigenschaften erfüllt. Inhaltlich kann dies etwa bedeuten: einen Punkt auf einer speziellen Position im Koordinatensystem zu plazieren, einen besonderen Kurvenpunkt zu bestimmen oder die Position eines Schiebereglers auf einen speziellen numerischen Wert zu justieren.

Um die Antwort auf eine Aufgabe zu kontrollieren und zu bewerten, klickt der Schüler den Button "Auswertung" mit der Computermaus an und löst damit das Verfahren der in Abschnitt 3.1 beschriebenen Antwortanalyse aus. Der Antwort des Schülers wird dadurch ein Antwortwert zugewiesen und dem Schüler wird der zum Antwortwert zugehörige Kommentar angezeigt. Wenn für eine Aufgabe eine maximale Gesamthöchstzahl von Antwortversuchen festgelegt worden ist und diese erreicht wird, bekommt der Schüler einen Schlußkommentar und ggf. eine Lösungsfigur angezeigt. Ist keine Gesamthöchstzahl vorgegeben, kann der Schüler die Antwortanalyse unbegrenzt oft aufrufen.

Beispiel: Teilverhältnis

Der folgende Lernbaustein ist ein Beispiel für eine einfache Variationsaufgabe zum Begriff Teilverhältnis mit der Möglichkeit zur Selbstkontrolle. In dem Lernbaustein ist eine Gerade durch zwei Punkte A und B gegeben. Ein dritter Punkt T kann auf der Geraden bewegt werden. Mit Hilfe einer externen Funktion (Seite 383) wird das Teilverhältnis $t = \frac{AT}{TB}$ berechnet. Die Aufgabe des Schülers besteht darin, den Punkt T so zu verschieben, daß $\frac{AT}{TB} = -4$ ist (Abbildung 4.51). Der Kommentar zur Schülerantwort erscheint in einem separaten Fenster. Bei der Antwortanalyse können grundsätzlich beliebig viele falsche oder teilweise richtige Antworten definiert werden, die erkannt werden sollen. In diesem Beispiel sind drei Arten von speziellen Falschantworten definiert. Im ersten Fall liegt T zwischen A und B (Abbildung 4.51 und 4.52). Im zweiten Fall ist $|AT| < |TB|$ (Abbildung 4.53 und 4.54). Bei der dritten Falschantwort ist $t = -5$. Trifft dieser Fall zu, so hat der Schüler das Streckenverhältnis falsch berechnet (Abbildung 4.55 und 4.56). Die Lösungsfigur der Aufgabe und den Lösungskommentar zeigen die Abbildungen 4.57 und 4.58.

Ein weiteres Beispiel für eine einfache Variationsaufgabe ist der Lernbaustein zur Abstandssumme in der Taxi-Metrik (Seite 167).

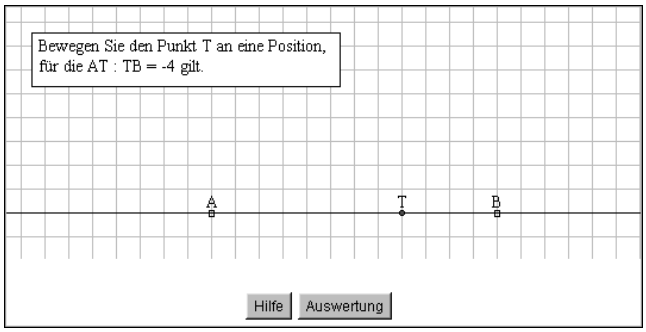

Abbildung 4.51: Einfache Variationsaufgabe zum Begriff Teilverhältnis.

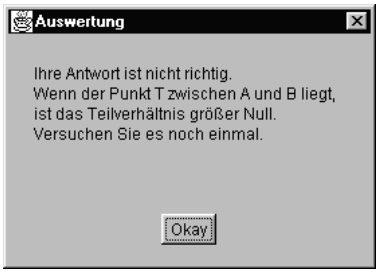

Abbildung 4.52: Antwortkommentar zur Figur in Abbildung 4.51.

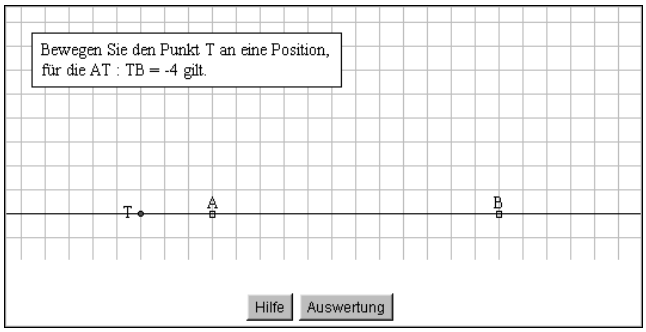

Abbildung 4.53: Falschantwort: $|AT| < |TB|$.

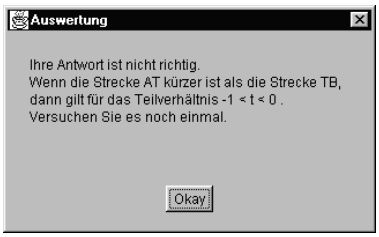

Abbildung 4.54: Antwortkommentar zur Figur in Abbildung 4.53.

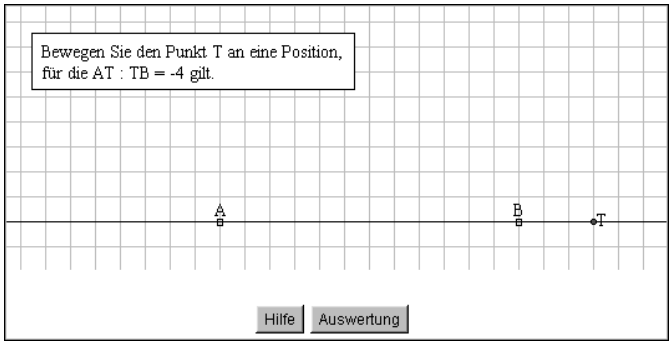

Abbildung 4.55: Falschantwort: $\frac{AT}{TB} = -5$.

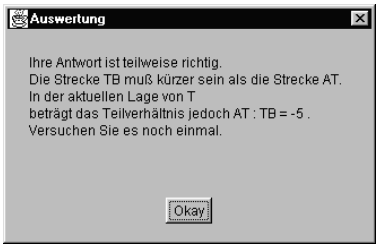

Abbildung 4.56: Antwortkommentar zur Figur in Abbildung 4.55.

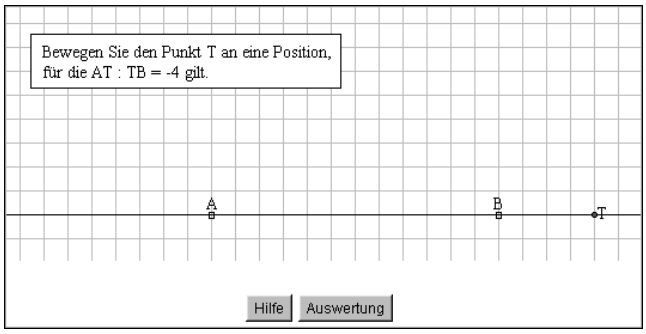

Abbildung 4.57: Lösungsfigur.

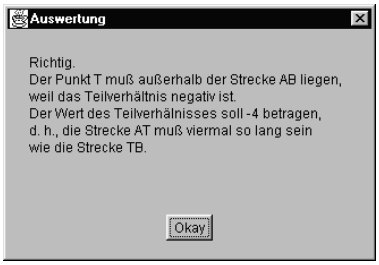

Abbildung 4.58: Antwortkommentar zur Figur in Abbildung 4.57.

4.3.3 Komplexe Variationsaufgaben

Einfache und komplexe Variationsaufgabe unterscheiden sich durch die Größe ihres Zustandsraums. Um eine komplexe Variationsaufgabe zu lösen, muß der Schüler mehrere Ein-Punkt-Inzidenzen, spezielle Typen von Polygonen oder andere geometrische Relationen herstellen. Die Aufgabe ist für den Schüler daher schwieriger zu lösen. Das Aufrufen der Antwortanalyse und ihr Ablauf erfolgt jedoch wie bei einfachen Variationsaufgaben. Ebenso lassen sich beliebig viele falsche und teilweise richtige Antworten erkennen und kommentieren.

Beispiel: Satz von Ceva

Im Zusammenhang mit dem Satz von Ceva geht es darum, die Ecktransversalen in einem gegebenen Dreieck auf drei spezielle Teilverhältnisse einzustellen. Zur Kontrolle kann der Schüler seine Antwort auswerten und kommentieren lassen. Die Abbildungen 4.59 und 4.60 zeigen eine falsche Antwort und den entsprechenden Kommentar. Die Abbildungen 4.61 und 4.62 zeigen die Lösung und den Schlußkommentar.

Weitere Beispiele für komplexe Variationsaufgaben sind: Würfelnetze (Seite 133), Haus der Vierecke (Seite 135), gleichseitiges Dreieck im Quadrat (Seite 137), Geradenspiegelung (Seite 145), Drehstreckung eines Dreiecks (Seite 146), Mengensprache (Seite 170). Auch das auf Seite 174 beschriebene Problem der acht Damen kann als komplexe Variationsaufgabe angesehen werden.

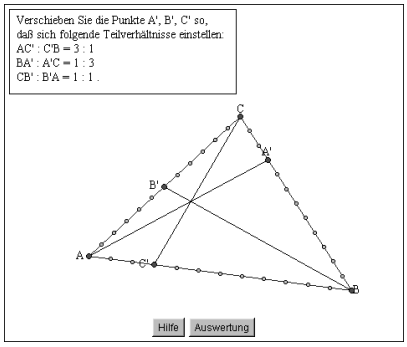

Abbildung 4.59: Komplexe Variationsaufgabe zum Satz von Ceva.

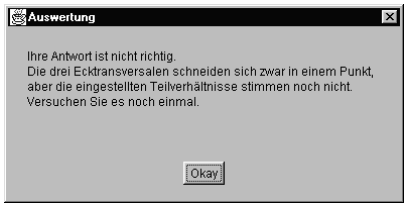

Abbildung 4.60: Antwortkommentar zur Figur in Abbildung 4.59.

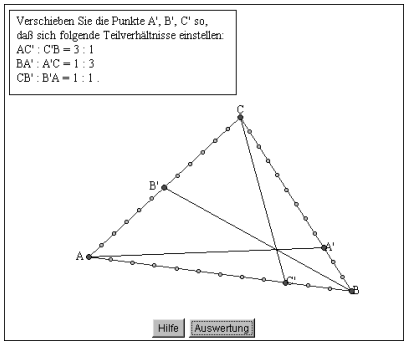

Abbildung 4.61: Lösungsfigur.

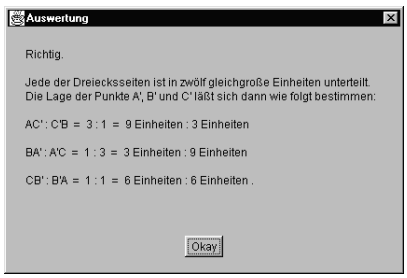

Abbildung 4.62: Schlußkommentar zur Figur in Abbildung 4.61.

4.4 Thematische Beispielsammlung

Mit dieser thematisch gegliederten Sammlung von Lernbausteinen möchte ich an ausgewählten Beispielen zeigen, wie breit das Spektrum an Inhalten ist, zu dem geometrische Lernbausteine entwickelt werden können. Etwas problematisch bei der Präsentation einer solchen Beispielsammlung ist, daß auf diese Weise die Figuren nur isoliert ohne konkreten Lernkontext dargeboten werden. Aus diesem Grunde werde ich, wenn es passend ist, einen didaktischen Zusammenhang zumindest nennen, in dem ein Lernbaustein eingesetzt werden könnte. Neben dem inhaltlichen Umfang sollen mit dieser Beispielsammlung auch besondere Funktionen von *Geometria* demonstriert werden. Darauf werde ich an geeigneter Stelle hinweisen. Die Skripte zu allen Lernbausteinen sind im Anhang C (Seite 298-380) wiedergeben.

4.4.1 Elementargeometrie

Geometrische Lernbausteine sind vor allem auf den Bereich der Elementargeometrie zugeschnitten. Da bereits viele Vorschläge für Figuren und interaktive Arbeitsblätter in der Literatur (z. B. Schumann 1991, Henn & Jock 1992, King & Schattschneider 1997, Lugon, Chastellain & Atzbach 1991, Shaffer 1995) veröffentlicht wurden, werde ich mich auf einige wenige beschränken. Außerdem sei auf das Online-Skript zur Elementargeometrie verwiesen (Kapitel 5), das mehr als 50 Lernbausteine enthält.

Zerlegung von Dreieck und Quadrat

Der berühmte Rätselerfinder H. E. Dudeney hat in seiner ersten Rätselsammlung "The Canterbury Puzzles" (1907) eine Aufgabe veröffentlicht, die er das Kurzwarenhändler-Problem nannte. Dabei geht es darum, ein gleichseitiges Dreieck in vier Stücke zu zerteilen und aus diesen ein Quadrat zu bilden. In dem in Abbildung 4.63 dargestellten Lernbaustein wird das Problem in einer etwas abgewandelten Form präsentiert. Es sind vier Polygone vorgegeben, und die Aufgabe des Schülers besteht darin, die Vielecke durch Drehen und Verschieben einmal zu einem Quadrat und einmal zu einem gleichseitigen Dreieck zusammenzusetzen (Abbildung 4.63 - 4.65). Dieser Lernbaustein wurde auf der Website

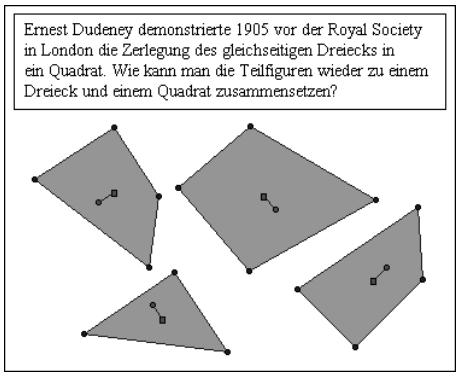

Abbildung 4.63: Zerlegung von Dreieck und Quadrat nach H. E. Dudeney.

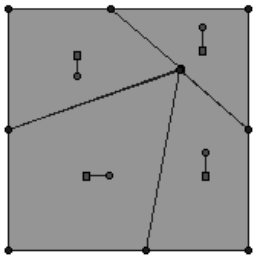

Abbildung 4.64: Lösungsfigur Quadrat.

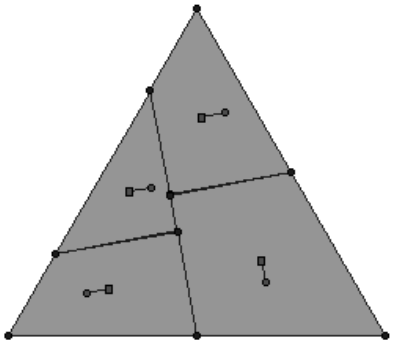

Abbildung 4.65: Lösungsfigur Dreieck.

des Instituts für Mathematik und ihre Didaktik der Universität Flensburg als Denkzettel Nr. 10 veröffentlicht.[19]

Satz von Holditch

Im folgenden soll ein Lernbaustein beschrieben werden, der den Satz von Holditch visualisiert.[20] Gegeben ist eine Eilinie, d. h. eine geschlossene konvexe Kurve K, in deren Innerem eine Stange s so bewegt werden kann, daß ihre beiden Endpunkte O und S die Kurve K berühren. Ist s genügend klein, so ist durch s und K ein geschlossener Bewegungsvorgang erklärt, bei dem die beiden Stangenendpunkte die Eilinie jeweils einmal durchlaufen.

Die in der Abbildung 4.66 dargestellte Figur visualisiert den beschriebenen Sachverhalt. Bewegt man den Endpunkt O auf der Eilinie, so verschiebt sich die Stange s entsprechend mit. Die Länge der Stange kann durch Verlängern oder Verkürzen der Strecke $O'S'$ im oberen Teil der Zeichenfläche variiert werden. Die Bahnkurve (Ortslinie) des Punkts X auf der Stange s, mit $OX = x$ und $XS = y$, bildet eine geschlossene Kurve. Im Lernbaustein kann die Bahn von X durch das Aufzeichnen der Ortsspur interaktiv generiert werden (Abbildung 4.67). Überraschenderweise ist der Flächeninhalt F des Ringgebiets zwischen der Ortslinie von X und der Kurve K unabhängig von der Form und Größe der Eilinie. Der Satz von Holditch besagt:

$$F = \pi xy.$$

Der Flächeninhalt des Ringgebiets hängt also lediglich von den Entfernungen x und y des Punkts X von den Stangenenden ab. Einen Beweis findet man in Blaschke & Müller (1956, S. 120).

Diese Figur ist ein Beispiel für das Verwenden einer (statischen) Punktmenge als Bezugsobjekt für einen ziehbaren Punkt (Abschnitt 3.3.9).

[19] http://www.uni-flensburg.de/mathe/dzettel/0010/dz_0010.html
[20] Blaschke & Müller 1956, S. 120

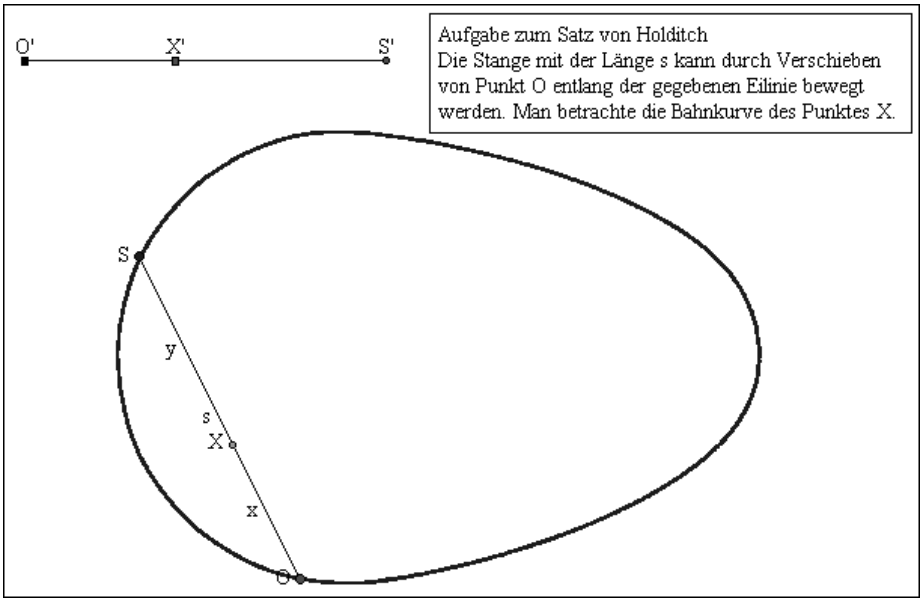

Abbildung 4.66: Die Stange s wird entlang der Eilinie bewegt.

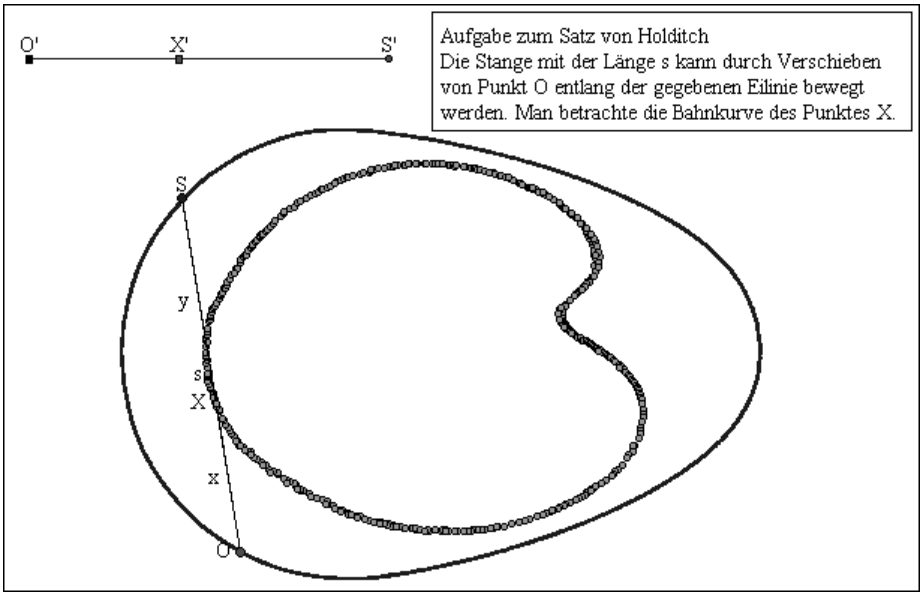

Abbildung 4.67: Wird s verschoben, so umschließt die Ortsspur von X eine nicht-konvexe Fläche.

Würfelnetze

Zum Thema Würfelnetze enthält das Geometrie-Lernprogramm "Elly für Windows" von J. Ingold eine interessante Aufgabe mit Antwortanalyse. In Anlehnung daran habe ich einen Lernbaustein entwickelt, der ähnliches leistet.

Gegeben sind sechs Quadrate, die in einem Gitternetzraster auf der Zeichenfläche verschoben werden können. Werden die Quadrate so aneinandergelegt, daß es keine Überlappungen gibt und daß jedes Quadrat mit mindestens einer Seite die Seite eines anderen Quadrats berührt, dann läßt sich die Gesamtfläche als ein Würfelnetz interpretieren. Die Aufgabe des Schülers besteht nun darin, das ein oder andere Würfelnetz zu finden. Nach dem Aufrufen der Antwortanalyse erhält der Schüler Rückmeldung, ob ein gültiges Würfelnetz vorliegt (Abbildung 4.68 und 4.69) oder ob das Netz sich nicht zu einem Würfel falten läßt (Abbildung 4.70 und 4.71). Ferner wird erkannt, ob überhaupt ein Netz mit einer zusammenhängenden Fläche erstellt worden ist (Abbildung 4.72 und 4.73).

Bei der Definition der Aufgabe werden die Prüfschlüssel für die einzelnen Antwortwerte mit Hilfe der speziell entwickelten und durch *GeoScript* eingebundenen Java-Klasse `Functional_Wuerfelnetz` (Seite 384) realisiert. Durch diese Klasse können gültige Würfelnetze erkannt werden. Der Prüfalgorithmus untersucht dazu, ob die sechs Quadratmittelpunkte ein Muster bilden, das mit einem der möglichen elf Würfelnetze übereinstimmt und liefert einen entsprechenden Wert zurück.

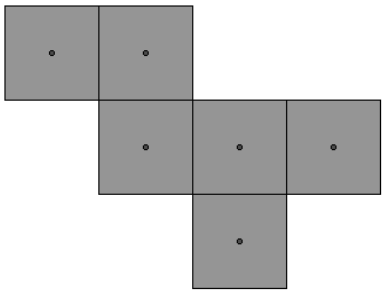

Abbildung 4.68: Sechs verschiebbare Quadrate können als ein Würfelnetz aufgefaßt werden.

Abbildung 4.69: Antwortkommentar zum Würfelnetz in Abbildung 4.68.

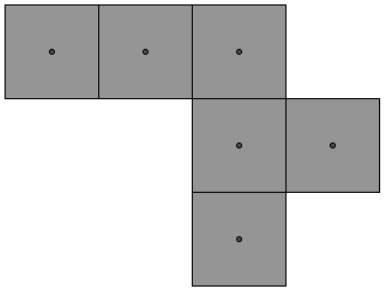

Abbildung 4.70: Ein Netz, das nicht zum Würfel gefaltet werden kann.

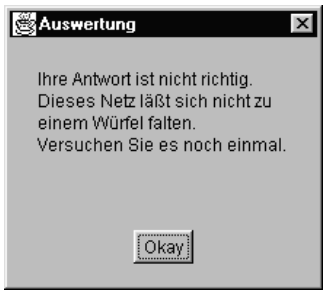

Abbildung 4.71: Antwortkommentar zum Netz in Abbildung 4.70.

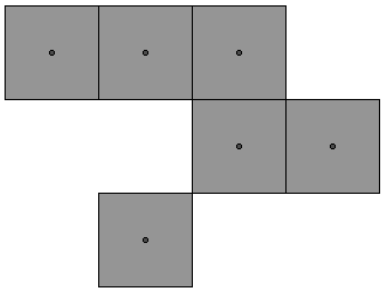

Abbildung 4.72: Ein nicht zusammenhängendes Netz.

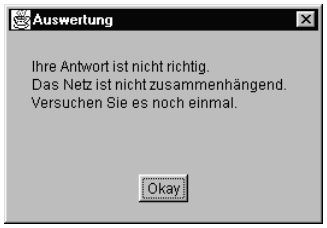

Abbildung 4.73: Antwortkommentar bei einem nicht zusammenhängenden Netz.

Das Haus der Vierecke

Das Haus der Vierecke ist ein Thema im Geometrieunterricht, an dem logisches Ordnen und Klassifizieren geübt werden kann. In einem Artikel zum Thema Haus der Vierecke beschreibt Neubrand[21] mathematische Prozesse, die bei der Begriffsbildung auftreten können. Ausgehend von den bekannten Viereckstypen wie etwa Quadrat, Raute, Drachen, Rechteck, ... läßt sich ein Beziehungsnetz aufstellen, in dem eine Lücke offen bleibt (Abbildung 4.74). Bei der Suche nach

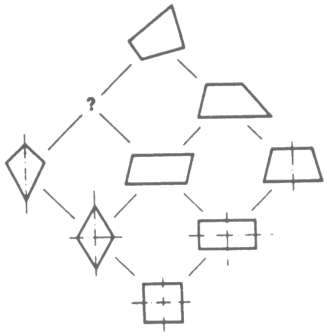

Abbildung 4.74: Welcher Viereckstyp paßt in die Lücke?

einem Viereck, das in die Lücke passen könnte, muß der Schüler die Systematik der Hierarchie berücksichtigen. Dabei gilt es, einen Oberbegriff für ein Parallelogramm und einen Drachen zu finden, der mit dem Trapez jedoch nur den Sonderfall des Parallelogramms gemeinsam hat.[22] Die Suche nach einem solchen Viereckstyp ist gleichzeitig die Suche nach einem neuen Begriff.

Entsprechend dieser Aufgabenstellung habe ich den folgenden Lernbaustein entwickelt (Abbildung 4.75). In der Zeichenfläche ist das Haus der Vierecke abgebildet. Von den neun hierarchisch angeordneten Vierecken kann das farblich hervorgehobene Viereck $ABCD$ bewegt werden. Der Schüler hat dadurch die Möglichkeit, bei der Suche nach dem speziellen Viereckstyp experimentell vorzugehen. Als Hilfestellung kann er sich die Diagonalen und ihre Mittelpunkte anzeigen lassen oder die vier Innenwinkel einblenden. Der Rasterfangmodus ist eingeschaltet, um die Figur einfacher zu positionieren. Aufgabe ist es, das Viereck so zu variieren, daß es dem gesuchten Viereckstyp entspricht. Als Lösung für diese Aufgabe nennen Graumann et al. zwei Viereckstypen: einen schrägen Drachen und einen schiefen Drachen.

Schräger Drachen In einem Parallelogramm halbieren sich die Diagonalen gegenseitig, dagegen wird beim Drachen nur eine Diagonale durch den Schnittpunkt der anderen Diagonalen halbiert. Allerdings stehen die Diagonalen senkrecht aufeinander. Ein Viereck, bei dem ein Diagonalenschnittpunkt die andere Diagonale halbiert, wäre eine Verallgemeinerung von Parallelogramm und Drachen (Abbildung 4.76).

[21] Neubrand 1981, S. 37-50
[22] Graumann et al. 1996, S. 212f

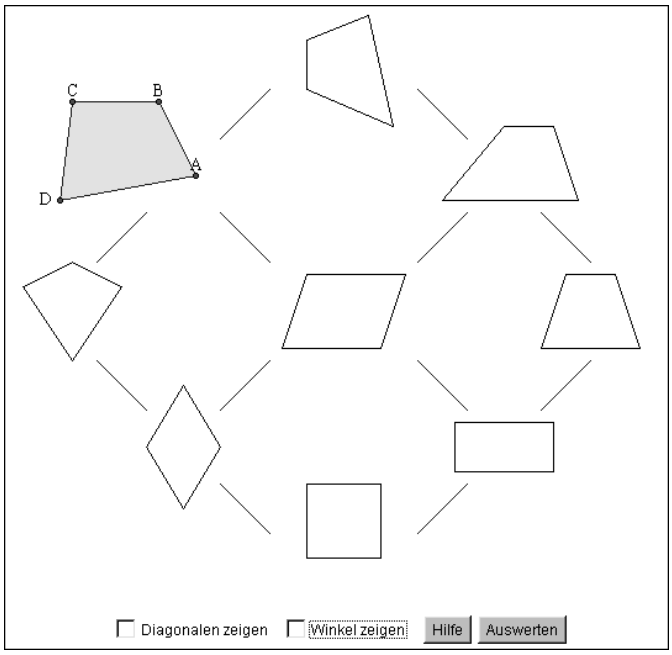

Abbildung 4.75: Lernbaustein zur Begriffsbildung beim Haus der Vierecke.

Schiefer Drachen In einem Parallelogramm liegen paarweise gleich große Winkel einander gegenüber, dagegen hat ein Drachenviereck nur ein gegenüberliegendes gleiches Winkelpaar. Zusätzlich besitzt es allerdings noch zwei paarweise gleich lange Seiten. Eine Verallgemeinerung von Parallelogramm und Drachen wäre daher ein Viereck mit nur einem Paar gegenüberliegender gleicher Winkel (Abbildung 4.77).

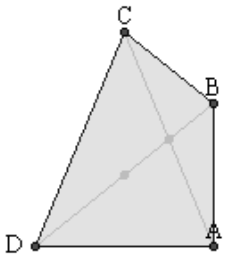

Abbildung 4.76: Schräger Drachen.

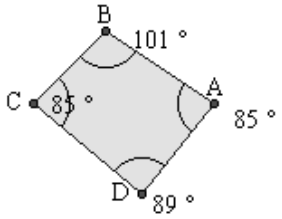

Abbildung 4.77: Schiefer Drachen.

Nachdem der Schüler mit dem beweglichen Viereck experimentiert hat, kann er seine Lösung kontrollieren lassen, indem er die Antwortanalyse aufruft. Die Antwortwerte sind dabei so belegt, daß eine Antwort als richtig eingestuft wird, wenn $ABCD$ ein schiefes oder schräges Drachenviereck ist. Als falsche Antworten werden die fest vorgegebenen acht Viereckstypen erkannt und entsprechend kommentiert. Die Abbildungen 4.78 und 4.79 zeigen die Antwortkommentare für

die beiden richtigen Viereckstypen. Die Definition der Aufgabe mit *GeoScript*

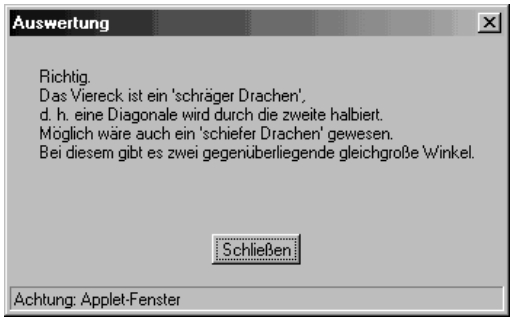

Abbildung 4.78: Antwortkommentar zur Figur in Abbildung 4.76.

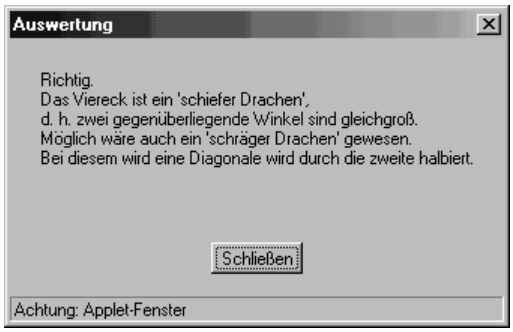

Abbildung 4.79: Antwortkommentar zur Figur in Abbildung 4.77.

greift bei der Beschreibung der Prüfschlüssel für die Antwortwerte auf die Eigenschaften eines Polygonobjekts zu. Wie bereits in Abschnitt 3.3.8 erwähnt, ist in der Klasse `PolygonElement` eine automatische Viereckanalyse implementiert, die zu jedem Viereck die beste Viereckklasse liefert.

Gleichseitiges Dreieck im Quadrat

Eine spezielle Klasse von Aufgaben, die gut durch Lernbausteine dargestellt werden können, sind Extremwertaufgaben. Einzelne Parameter einer Figur lassen sich durch gezieltes Variieren so verändern, daß bestimmte Werte ihre Extrema annehmen. Wird die Figur in einen solchen Zustand versetzt, so kann dieser durch eine Antwortanalyse kontrolliert, bewertet und kommentiert werden.

Exemplarisch möchte ich einen Lernbaustein mit einer Extremwertaufgabe beschreiben. In der Zeichenfläche ist ein Quadrat vorgegeben, dem ein gleichseitiges Dreieck ABC so einbeschrieben werden soll, daß der Flächeninhalt von ABC möglichst groß ausfällt (Abbildung 4.80). Als numerische Rückmeldung wird dem Schüler die Größe der Dreiecksfläche angezeigt, außerdem ist eine Antwortanalyse vorbereitet. Diese erkennt neben den richtigen Antworten (Abbildung 4.81 und 4.82) auch Figurenzustände, in denen die Fläche nicht maximal ist (Abbildung 4.83 und 4.84).

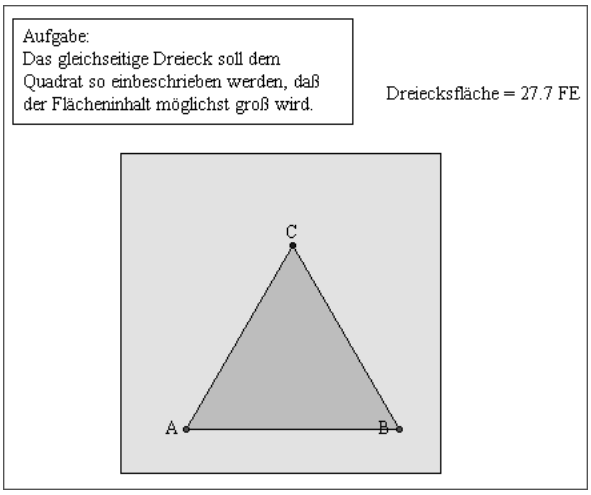

Abbildung 4.80: Bewegliches Dreieck in Quadratfläche.

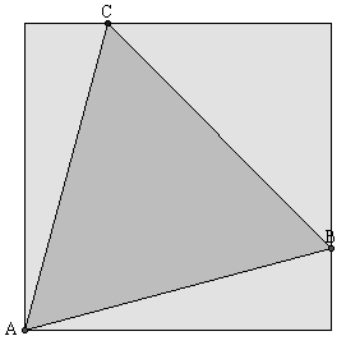

Abbildung 4.81: Gleichseitiges Dreieck in einem Quadrat mit maximaler Fläche.

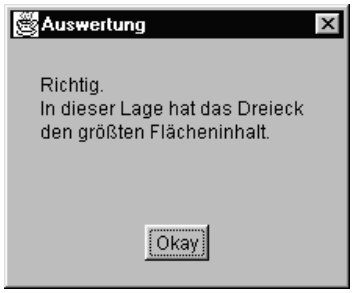

Abbildung 4.82: Antwortkommentar zur Figur in Abbildung 4.81.

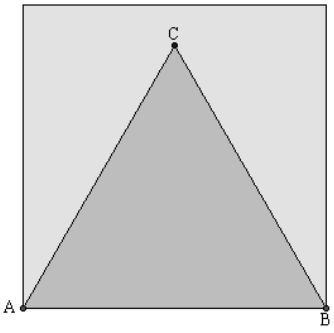

Abbildung 4.83: Gleichseitiges Dreieck in einem Quadrat mit nicht maximaler Fläche.

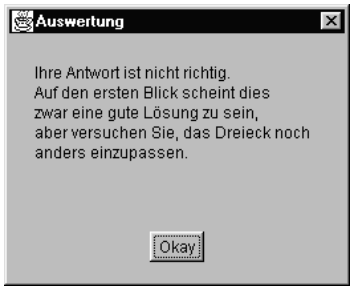

Abbildung 4.84: Antwortkommentar zur Figur in Abbildung 4.83.

Eine technische Besonderheit: Weder die ziehbaren Punkte A und B noch der nicht-ziehbare Punkt C lassen sich aus der Quadratfläche herausbewegen. Bei einer ähnlichen Figur mit einem aktuellen DG-System sind solche Beschränkungen des Zustandsraums nicht möglich. Die Punktobjekte lassen sich dort stets in der gesamten Zeichenfläche verschieben, was bei dieser Aufgabenstellung nicht angemessen wäre.

Weitere Beispiele für Extremwertaufgaben, die sich auch durch Lernbausteine realisieren ließen, beschreiben Schupp[23] sowie Danckwerts & Vogel[24]. Schumann stellt in einem Artikel Möglichkeiten der dynamischen Behandlung von geometrischen Extremwertaufgaben vor.[25] Ein Beispiel für diesen Aufgabentyp habe ich bereits mit der Figur "Schachtelvolumen" auf Seite 101 gegeben.

4.4.2 Maße

Aufgaben zur Bestimmung von Maßen besitzen in der Schulgeometrie als Unterrichtsgegenstand eine lange Tradition. Sie werden z. B. häufig in schriftlichen Prüfungen eingesetzt. Bei einer Maßbestimmungsaufgabe geht es darum, aus gegebenen Größen und Größenverhältnissen einer geometrischen Figur eine oder mehrere unbekannte Größen oder Größenverhältnisse zu bestimmen. Um eine Maßbestimmungsaufgabe zu lösen, ist heuristisches sowie geometrisches Wissen erforderlich, und arithmetisch-algebraische Fertigkeiten werden vorausgesetzt.

Ein großer Anteil herkömmlicher Maßbestimmungsaufgaben[26] kann durch Lernbausteine dargestellt werden. Am besten geeignet sind jedoch Maßbestimmungsaufgaben, bei denen die Variation der Figur im Zugmodus zur Problemlösung beiträgt. Dieses möchte ich an einem Beispiel aus der Schulgeometrie demonstrieren.

Verhältnis zweier Quadratflächen

Die Idee für den folgenden Lernbaustein stammt von einer Aufgabe aus dem Swarthmore-Forum "Geometry Problem of the week"[27] vom November 1998.

Der Lernbaustein enthält in der Zeichenfläche ein Quadrat Q_1, dessen Eckpunkte A und B ziehbar sind. Dem Quadrat Q_1 ist ein Kreis k einbeschrieben, der die vier Seitenmitten von Q_1 berührt. Dem Kreis wiederum ist ein Quadrat Q_2 einbeschrieben, dessen vier Eckpunkte auf k liegen. Der Eckpunkt R von Q_2 kann auf der Kreislinie verschoben werden. Q_2 läßt sich auf diese Weise um den Kreismittelpunkt drehen (Abbildung 4.85). Zu der Figur wird die folgende Aufgabe gestellt: In welchem Verhältnis stehen die Flächeninhalte des kleinen und des großen Quadrats zueinander? Der Schüler kann die Figur nun so variieren, daß für ihn sichtbar wird: Die Diagonale des inneren Quadrats ist gleich der Seite des äußeren Quadrats (Abbildung 4.86). Daraus läßt sich eine Lösungsidee ableiten. Kennt man die Diagonale eines Quadrats, so kann man auch seine Seitenlänge bestimmen. Mit der Seitenlänge läßt sich wiederum der Flächeninhalt berechnen.

[23] Schupp 1992
[24] Danckwerts & Vogel 1998, S. 74-79
[25] Schumann 1999
[26] Eine Sammlung von ca. 300 Maßbestimmungsaufgaben hat Paul Eigenmann (1981) zusammengestellt.
[27] http://forum.swarthmore.edu/geopow/archive.html

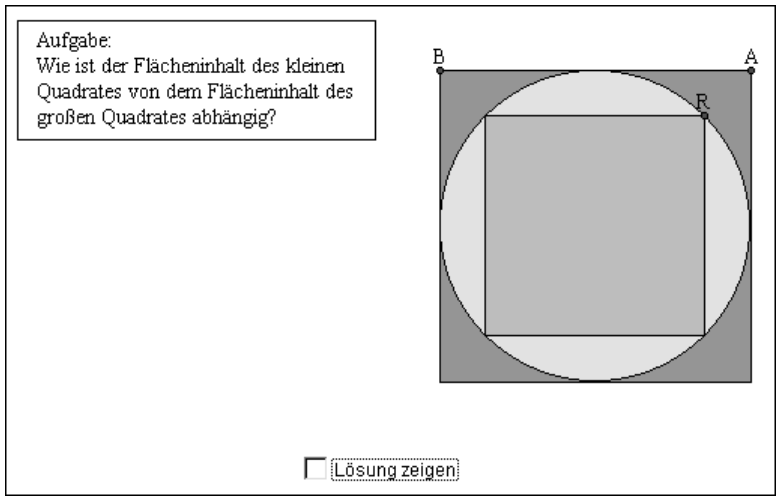

Abbildung 4.85: Wie verhalten sich die beiden Quadratflächen zueinander?

Der Flächeninhalt A_1 des äußeren Quadrats beträgt $|AB|^2$. Die Länge der Diagonalen d_2 des inneren Quadrats ist $|AB|$. Die Seitenlänge s_2 von Q_2 kann man durch den pythagoräischen Lehrsatz bestimmen: ${s_2}^2 + {s_2}^2 = {d_2}^2$. Daraus folgt:

$$A_2 = {s_2}^2 = \frac{{d_2}^2}{2} = \frac{|AB|^2}{2} = \frac{A_1}{2}.$$

Ohne arithmetisch-algebraisches Kalkül läßt sich die Lösung aber noch einfacher finden. Denkt man sich die Diagonalen des inneren Quadrats eingezeichnet und Q_2 so positioniert, daß die Eckpunkte mit den Seitenmitten inzidieren, so ist leicht zu sehen, daß der Flächeninhalt des äußeren Quadrats doppelt so groß ist wie der des inneren. Zur Kontrolle kann sich der Schüler die Lösungsfigur anzeigen lassen (Abbildung 4.87).

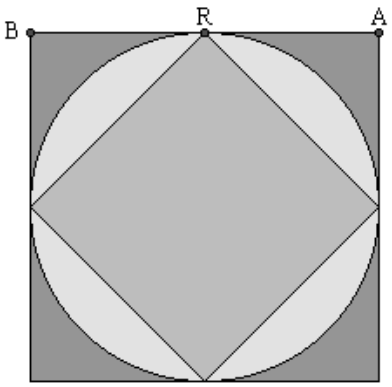

Abbildung 4.86: Die Seite des äußeren Quadrats ist so lang wie die Diagonale des inneren.

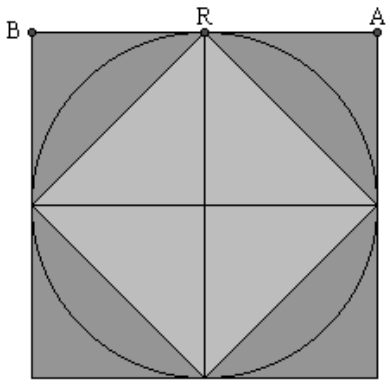

Abbildung 4.87: Lösungsfigur: Die Flächeninhalte stehen im Verhältnis 2:1.

Sangaku - Japanische Tempelgeometrie

Anstatt weitere Beispiele von Maßbestimungsaufgaben aus dem Bereich der Schulgeometrie zu nennen, möchte ich lieber auf eine Aufgabenklasse hinweisen, die im deutschsprachigen Raum bislang wenig beachtet wurde. Als Sangaku oder japanische Tempelgeometrie werden geometrische Aufgaben bezeichnet, die auf kunstvoll gezeichneten, kolorierten Holztafeln unter die Dächer von Tempeln gehängt wurden. Sie entstanden seit der Zeit der japanischen Isolation (1639-1854), in der es in Japan nahezu keinen Austausch von wissenschaftlichen Erkenntnissen mit der westlichen Welt gab. Die Sangaku-Tafeln wurden vermutlich als eine Form der religiösen Ehrung aufgestellt und sollten die Gläubigen herausfordern, die Aufgaben zu lösen. Inhaltlich werden auf den Holztafeln spezielle Anordnungen von Kreisen, Ellipsen, Polygonen oder Kugeln dargestellt, zu denen bestimmte Maßverhältnisse zu berechnen sind. Auffällig ist, daß dabei Kreise und Ellipsen fast immer eine Rolle spielen.

Für Lernbausteine eignet sich diese Aufgabenklasse vor allem deshalb, weil die ästhetischen Figuren relativ leicht nachgebildet werden können. Dabei betont der Zugmodus noch zusätzlich die Allgemeingültigkeit der zu zeigenden Maßverhältnisse.

Im folgenden möchte ich vier Lernbausteine mit typischen Sangaku-Aufgaben vorstellen. Sie sind beschrieben in der Sammlung mit über 200 Aufgaben von Fukagawa & Pedoe[28]. Der Artikel von Rothman[29] sei zur Einführung in das Thema Sangaku empfohlen.

Dreieck mit Gerade und zwei Kreisen Gegeben ist ein gleichseitiges Dreieck ABC. Durch den Punkt C verläuft eine im Zugmodus um C drehbare Gerade g. Der Kreis k_1 berührt g, AB und AC, und der Kreis k_2 berührt g, AB und BC (Abbildung 4.88). Zu berechnen ist die Summe der beiden Radien r_1 und r_2. Um eine Lösung zu finden, kann der Schüler die Figur nun variieren. Für den Spezialfall, daß die Gerade durch C parallel zur Geraden AB verläuft, läßt sich die Lösung leicht bestimmen (Abbildung 4.89). Beide Kreisdurchmesser sind gleich groß, und die Summe der beiden Radien ist gleich der Höhe im gleichseitigen Dreieck. Variiert man die Gerade durch C langsam von der besonderen Lage in eine allgemeine Lage, so läßt sich erkennen, daß r_1 im gleichen Maße zunimmt, wie r_2 abnimmt. Die Summe der beiden Radien bleibt also konstant. Für den allgemeinen Fall gilt daher:

$$r_1 + r_2 = \frac{\sqrt{3}}{2} AB.$$

Dreieck mit einbeschriebenen Kreisen und Quadraten Die Abbildung 4.90 zeigt ein rechtwinkliges Dreieck ABC, das in Größe und Lage variabel ist. Dem Dreieck sind drei Quadrate und drei Kreise einbeschrieben. Die Aufgabe des Schülers besteht darin, das Verhältnis der Kreisradien r_i (mit $i = 1, 2, 3$) zueinander zu bestimmen. Das Ergebnis ist verblüffend einfach: $r_1 r_3 = {r_2}^2$.

[28] Fukagawa & Pedoe 1989
[29] Rothman 1998

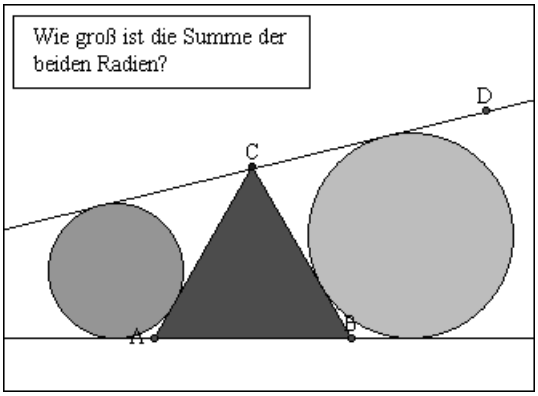

Abbildung 4.88: Dreieck mit Gerade und zwei Kreisen.

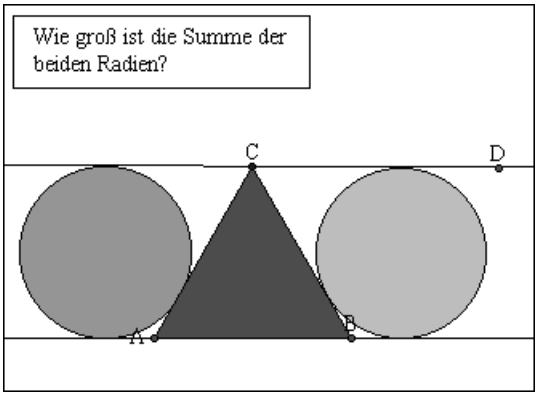

Abbildung 4.89: Die Gerade durch C verläuft parallel zur Geraden AB.

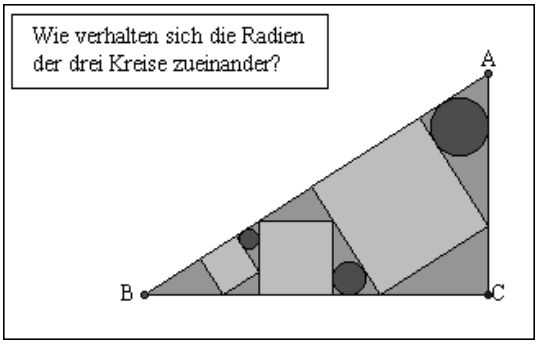

Abbildung 4.90: Dreieck mit einbeschriebenen Kreisen und Quadraten.

Drei Kreise auf einer Geraden Die Abbildung 4.91 zeigt eine Figur, in der drei Kreise so angeordnet sind, daß sie eine gegebene Gerade berühren. Man soll zeigen, daß stets

$$\frac{1}{\sqrt{r_3}} = \frac{1}{\sqrt{r_1}} + \frac{1}{\sqrt{r_2}}$$

gilt.

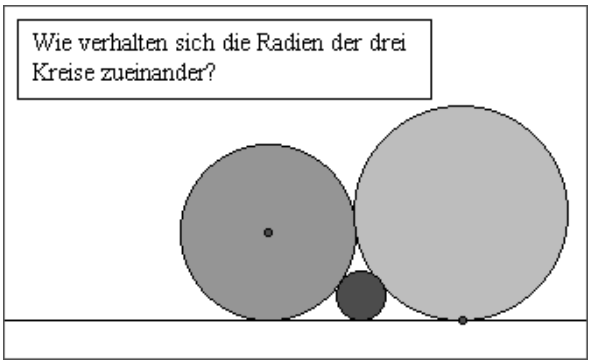

Abbildung 4.91: Drei Kreise auf einer Geraden.

Ellipse mit Tangente und Normale Die Abbildung 4.92 zeigt eine Ellipse auf der ein Punkt P verschoben werden kann. Durch P ist die Tangente an die Ellipse gezeichnet und senkrecht zur Tangente verläuft die Normale durch P. Diese schneidet die Ellipse in dem Punkt Q. Die Aufgabe besteht darin, den kleinsten Wert für PQ zu berechnen. Das Ergebnis, mit a und b als Ellipsenparameter, lautet:

$$PQ = \frac{\sqrt{27}a^2b^2}{(a^2+b^2)^{\frac{3}{2}}}.$$

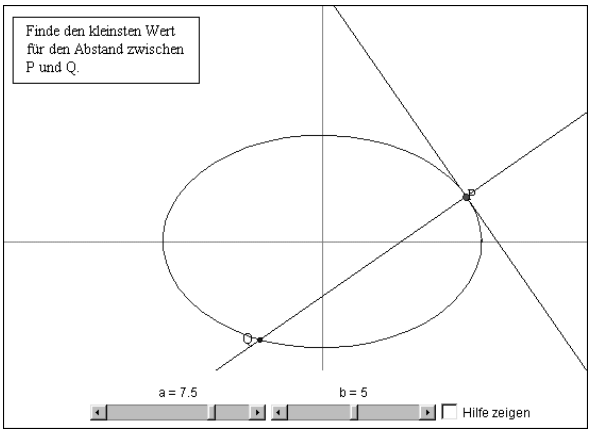

Abbildung 4.92: Ellipse mit Tangente und Normale.

4.4.3 Abbildungen

Das Thema Abbildungen ist ein weiteres Gebiet, für das sich Lernbausteine besonders eignen. Dabei kommen vor allem Abbildungen in Form von funktionalen Abhängigkeiten ($f\colon \mathbb{R} \to \mathbb{R}$) und in Form von geometrischen Transformationen ($f\colon \mathbb{R}^2 \to \mathbb{R}^2$) in Betracht. Eine Sequenz von Lernbausteinen, in denen funktionale Abhängigkeiten untersucht werden, beschreibe ich unter der Überschrift "Kurven" in Abschnitt 4.4.4. Im folgenden möchte ich Lernbausteine mit geometrischen Transformationen vorstellen. Dazu lassen sich neben den Kongruenzabbildungen auch nahezu beliebige affine und nicht-affine Abbildungen definieren. Ein Beispiel für eine affine Abbildung habe ich bereits durch die Figur auf Seite 95 gegeben.

Doppelspiegelung eines Fünfecks

In dem folgenden Lernbaustein geht es um das Untersuchen einer Verkettung von zwei Geradenspiegelungen. Dazu wird ein bewegliches Fünfeck $ABCDE$ nacheinander an zwei Geraden g_1 und g_2 gespiegelt. Die Bildfünfecke sind mit $A'B'C'D'E'$ und $A''B''C''D''E''$ bezeichnet. Die Aufgabe des Schülers besteht darin, die Geraden so zu verschieben, daß $ABCDE$ und $A''B''C''D''E''$ identisch sind (Abbildung 4.93).[30] Beim Variieren der Figur kann der Schüler die

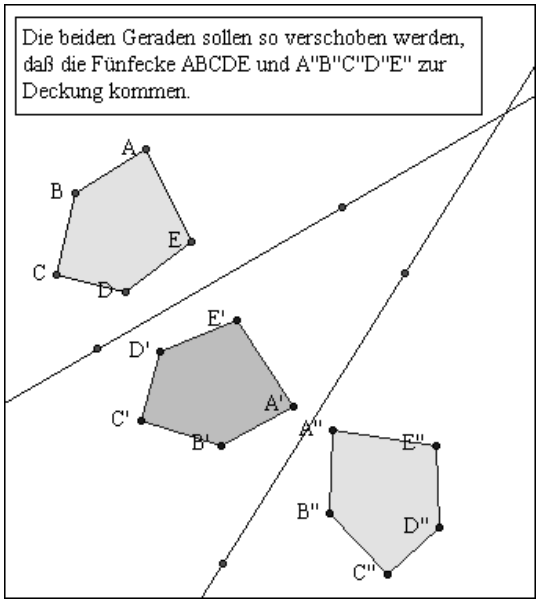

Abbildung 4.93: Doppelspiegelung eines Fünfecks.

Eigenschaften der Verkettung von zwei Geradenspiegelungen studieren. Sind die beiden Geraden zueinander parallel, so wird $ABCDE$ um den doppelten Geradenabstand verschoben. Schneiden sich die beiden Geraden, so findet eine Drehung um den Geradenschnittpunkt statt. Der Drehwinkel ist doppelt so groß

[30] Die Idee für diesen Lernbaustein stammt von einer Beispielaufgabe mit *Geonet*.

wie der Schnittwinkel zwischen g_1 und g_2. Zur Deckung kommen Ausgangs- und Bildfünfeck genau dann, wenn die beiden Geraden genau übereinander liegen.

Aufgabe zur Geradenspiegelung

Die folgende Aufgabe stammt aus einer Vorlesung[31] zur Abbildungsgeometrie. Gegeben ist ein Dreieck ABC, dessen Lage in der Zeichenfläche nicht veränderbar ist. Ebenfalls fest vorgegeben ist ein Punkt A'. Die zugehörige Aufgabe besteht darin, die bewegliche Gerade s und die beiden ziehbaren Punkte B' und C' so zu variieren, daß das Dreieck ABC durch eine Spiegelung an s in das Dreieck $A'B'C'$ übergeht. Zur Kontrolle der Lösung ist eine Antwortanaly-

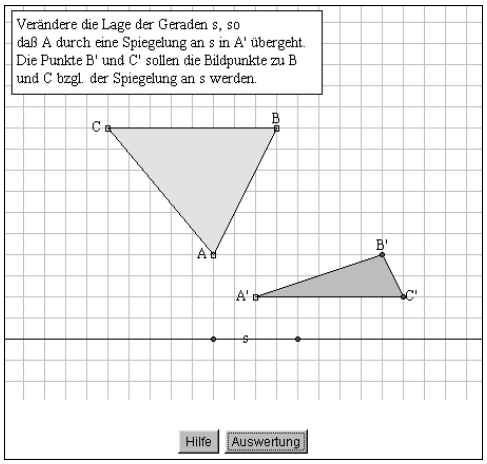

Abbildung 4.94: Geradenspiegelung eines Dreiecks.

se definiert. Dabei wird nicht nur die richtige Lösung erkannt, sondern es wird auch eine entsprechende Rückmeldung gegeben, falls die Lage der Geraden oder einzelner Bildpunkte nicht korrekt ist (Abbildung 4.95 und 4.96).

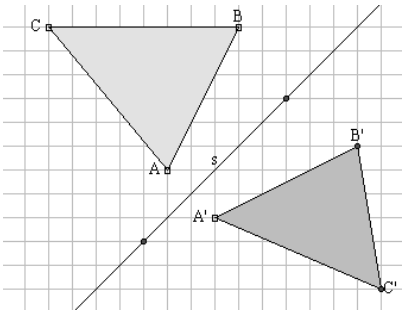

Abbildung 4.95: Diese Antwort ist nur teilweise richtig.

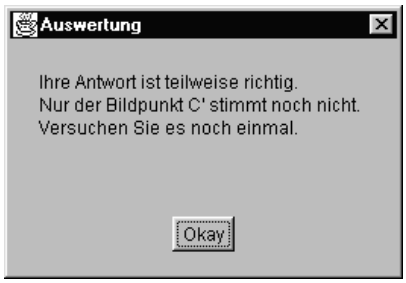

Abbildung 4.96: Bewertung des Figurenzustands aus Abbildung 4.95.

[31] Reimers: "Abbildungsgeometrie" gehalten an der Universität Flensburg im Wintersemester 1993/94.

Drehstreckung eines Dreiecks

Der folgende Lernbaustein enthält eine Übungsaufgabe zur Drehstreckung. Gegeben sind ein Dreieck ABC und ein Punkt Z, beide ortsfest. Die Aufgabe des Schülers ist es, die Eckpunkte des Dreiecks $A'B'C'$ so zu verschieben, daß das Dreieck ABC durch eine Drehung um 90° um das Drehzentrum Z mit einem Streckfaktor von $1,5$ in das Dreieck $A'B'C'$ übergeht (Abbildung 4.97). Analog

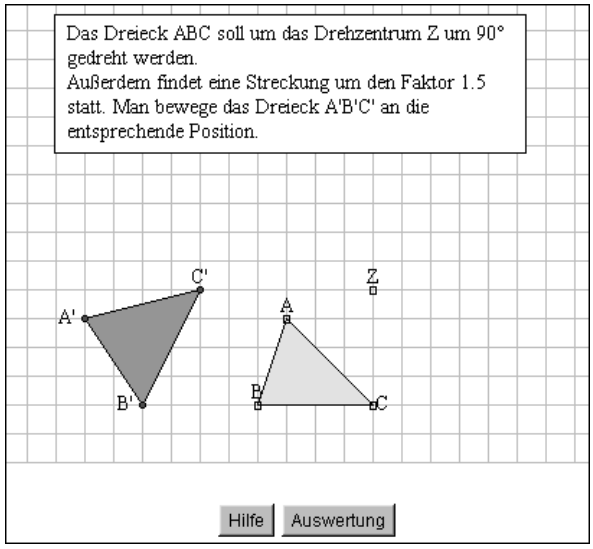

Abbildung 4.97: Drehstreckung eines Dreiecks.

zum vorangegangenen Beispiel ist ebenfalls eine Antwortanalyse definiert, die auch Lösungen kommentiert, die nur teilweise richtig sind. In der Abbildung 4.98 sind die Bildpunkte A' und C' korrekt plaziert, lediglich die Lage von B' stimmt noch nicht. Einen entsprechenden Kommentar zeigt die Abbildung 4.99.

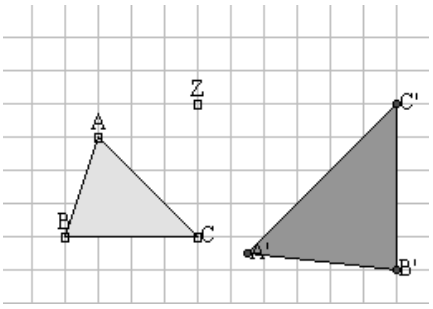

Abbildung 4.98: Diese Antwort ist nur teilweise richtig.

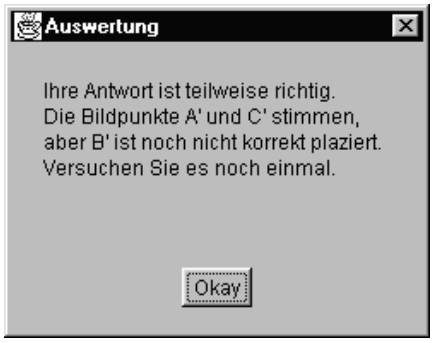

Abbildung 4.99: Bewertung des Figurenzustands aus Abbildung 4.98.

Eine nicht-affine Abbildung

Der folgende Lernbaustein zeigt exemplarisch eine nicht-affine Abbildung. Darin wird ein Dreieck ABC nach der Funktionsvorschrift $f(x, y) := (|x|, 3|y|)$ abgebildet. In der Zeichenfläche ist das Dreieck ABC innerhalb des gegebenen Koordinatensystems beweglich. Verschiebt der Schüler einen Eckpunkt, so verändert sich das Bilddreieck $A'B'C'$ entsprechend mit. Aufgabe ist es, die Funktionsvorschrift $f(x, y)$ herauszuarbeiten. Dabei können die in Abschnitt 4.2.1 beschriebenen heuristischen Strategien angewendet werden (Abbildung 4.100).

Dieser Lernbaustein ist auch ein Beispiel dafür, daß sich mit *GeoScript* nahezu beliebige Abbildungsvorschriften definieren lassen.

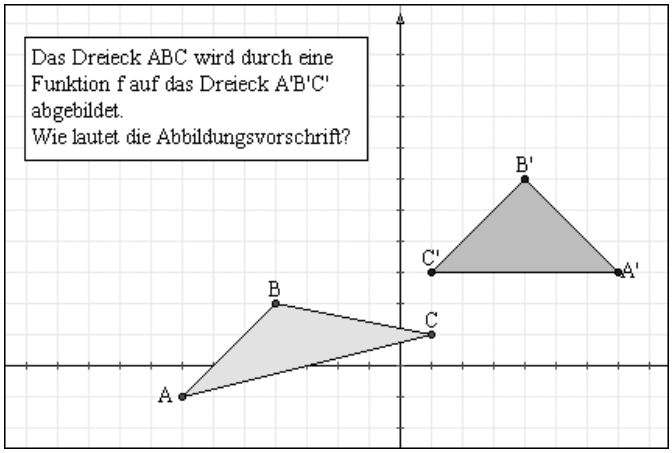

Abbildung 4.100: Durch systematisches Variieren soll die Abbildungsvorschrift gefunden werden.

4.4.4 Kurven

Im herkömmlichen Mathematikunterricht werden Aufgaben und Übungen zum Thema Kurven im allgemeinen in statischer Form behandelt. Neue Möglichkeiten ergeben sich durch den Zugmodus. Zu dem Thema Kurven zähle ich auch das interaktive Experimentieren mit Funktionsgraphen. In Anlehnung an die von Schumann[32] vorgestellten Phasen zur dynamischen Behandlung elementarer Funktionen werde ich im ersten Unterabschnitt eine Sequenz von Lernbausteinen vorstellen. In den daran anschließenden Unterabschnitten gebe ich einige Beispiele zu Lernbausteinen mit parametrisierten Kurven.

Während man mit aktuellen DG-Systemen Kurven durch Ortslinienobjekte darstellen muß, arbeitet *Geometria* mit einer eigenen Kurven-Klasse (Abschnitt 3.3.6). Aus diesem Grund habe ich zwischen Lernbausteinen zum Thema Kurven und zum Thema Ortslinien unterschieden.

[32] Schumann 1998, S. 172-188

Lernbaustein-Sequenz zur Behandlung von Funktionen

In dem oben erwähnten Artikel zeigt Schumann am Beispiel der quadratischen Funktion $f(x) = ax^2 + bx + c$, wie diese mit *Cabri-Géomètre II* interaktiv untersucht werden kann.[33] Ich möchte diese Behandlungsweise aufgreifen und mit der folgenden Lernbaustein-Sequenz ein Beispiel für die gebrochen-rationale Funktionenschar $f(x) = ax + \frac{b}{x} + c$ bringen.

Interaktives Generieren eines Funktionsgraphen Am Anfang einer Untersuchung einer Funktionenschar sollte der Schüler einen oder mehrere spezielle Funktionsgraphen interaktiv generieren. Dazu kann der Schüler einen Punkt auf der Abszisse $(x, 0)$ bewegen und die Ortsspur des Punkts $(x, f(x))$ aufzeichnen. Es wird dadurch verdeutlicht, wie der Graph einer Funktion zustande kommt, und ein erster Eindruck vermittelt, welche Form ein spezieller Funktionsgraph besitzt.

In der Abbildung 4.101 wurde die Ortsspur des Punkts $(x, f(x))$ mit $a = 1$, $b = -1$ und $c = 1$ aufgezeichnet.

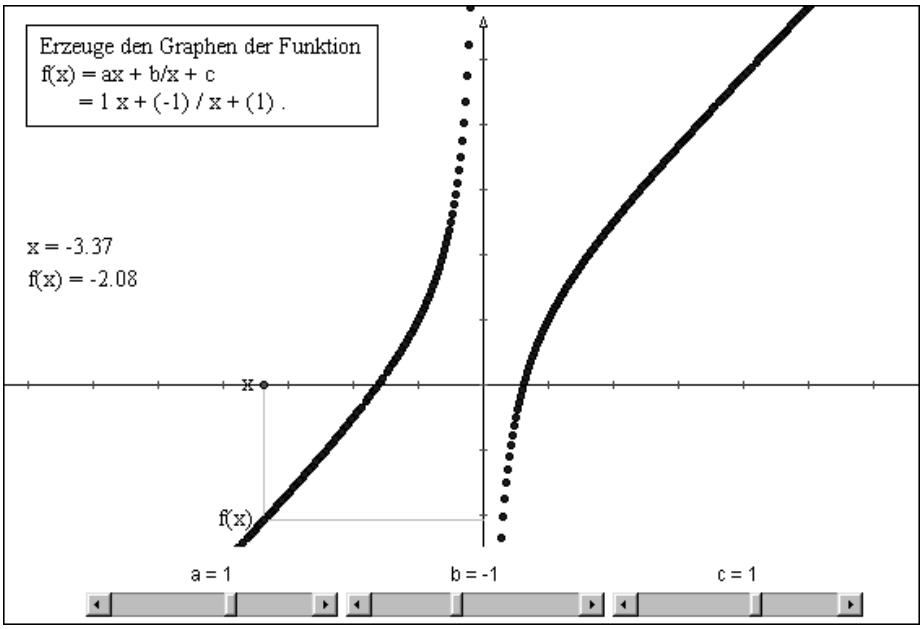

Abbildung 4.101: Interaktives Generieren eines Funktionsgraphen.

[33] Dabei wird die quadratische Funktion mit *Cabri-Géomètre II* durch ein Kegelschnittobjekt realisiert. Andere Funktionsklassen sind dagegen nicht so einfach zu erzeugen. Dazu müßte der Figurenautor mit einem Ortslinienobjekt arbeiten und auf umständliche Art Stützpunkte für die Ortslinie konstruieren. Dieses Problem tritt beim Erstellen von Lernbausteinen nicht auf, weil hier eine eigene Kurvenklasse verfügbar ist, die im Prinzip beliebige Funktionsdefinitionen interpretieren und darstellen kann (Abschnitt 3.3.6).

Interaktives Untersuchen besonderer Punkte Um besondere Punkte des Funktionsgraphen zu untersuchen, ist in dem folgenden Lernbaustein der Graph durch ein Kurvenobjekt realisiert, auf dem ein ziehbarer Punkt P verschoben werden kann. Schumann[34] nennt diesen Punkt einen Funktionscursor, mit dessen Hilfe besondere x- und y-Werte (etwa Nullstellen, Extremstellen) näherungsweise bestimmt werden können. In Abbildung 4.102 ist der Funktionscursor dargestellt, um ein lokales Minimum abzulesen. Durch die Variation der Funktionsparameter a, b, c können beliebige Funktionen der Form $f(x) = ax + \frac{b}{x} + c$ untersucht werden. Zu einer solchen Funktion lassen sich dann Extremwertaufgaben graphisch-experimentell lösen. Zur Kontrolle kann der Schüler die gefundenen Werte auch rechnerisch durch algebraische Umformungen der Funktionsgleichung bestätigen. Auch der umgekehrte Zugang ist möglich: Berechnete Extrempunkte kann der Schüler auf Richtigkeit überprüfen.

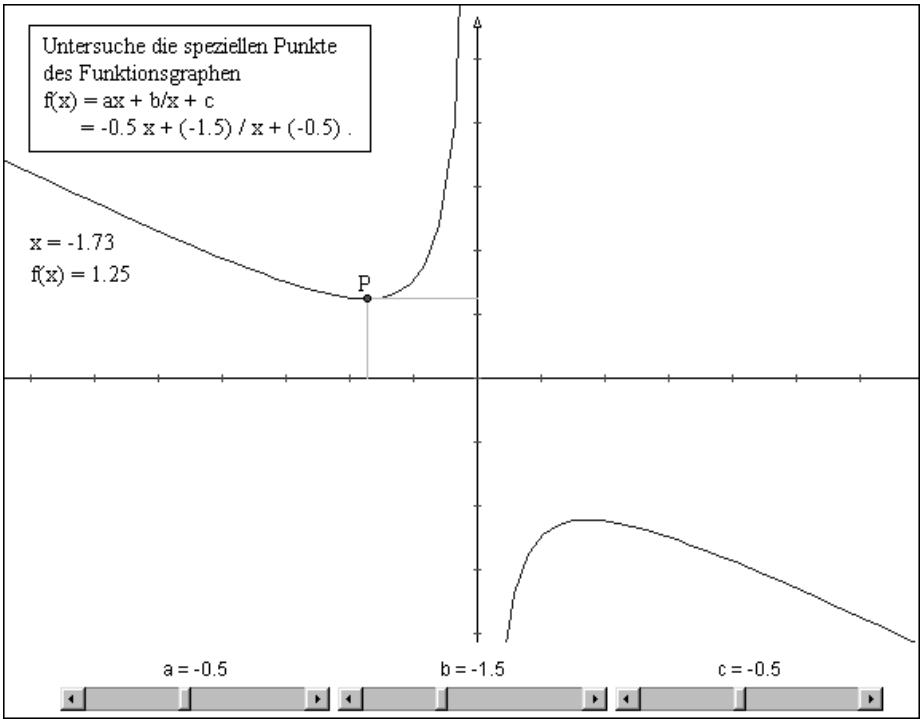

Abbildung 4.102: Interaktives Untersuchen besonderer Punkte.

[34] Schumann 1998, S. 174

Interaktives Untersuchen der Funktionsgleichung durch Verändern des Graphen Bei der folgenden Aufgabenstellung kann der Schüler die Form und Lage des Funktionsgraphen variieren und dabei beobachten, wie sich die Parameter der Funktionsgleichung ändern. Dazu sind drei frei ziehbare Referenzpunkte gegeben, durch die der Graph verläuft. Als zusätzliche Aufgabe können spezielle Kurven in der Zeichenfläche vorgegeben werden, deren Funktionsgleichung dann interaktiv zu bestimmen ist.

Die Abbildung 4.103 zeigt die ziehbaren Punkte P_1, P_2, P_3 und den durch die Punkte verlaufenden Graphen, der in der aktuellen Lage die Parameterwerte $a = -2$, $b = -12$ und $c = 12$ besitzt. Zusätzlich sind drei feste Graphen vorgegeben, deren Parameterwerte zu bestimmen sind.

Erwähnenswert ist die Realisation des interaktiven Schaubilds durch Referenzpunkte. Aus den Koordinaten der Referenzpunkte werden durch Lösen eines Gleichungssystems mit Hilfe einer externen Funktion die Funktionsparameter berechnet (Seite 389). Mit diesen Werten läßt sich ein – durch eine externe Klasse berechnetes – Kurvenobjekt erzeugen (Seite 390).

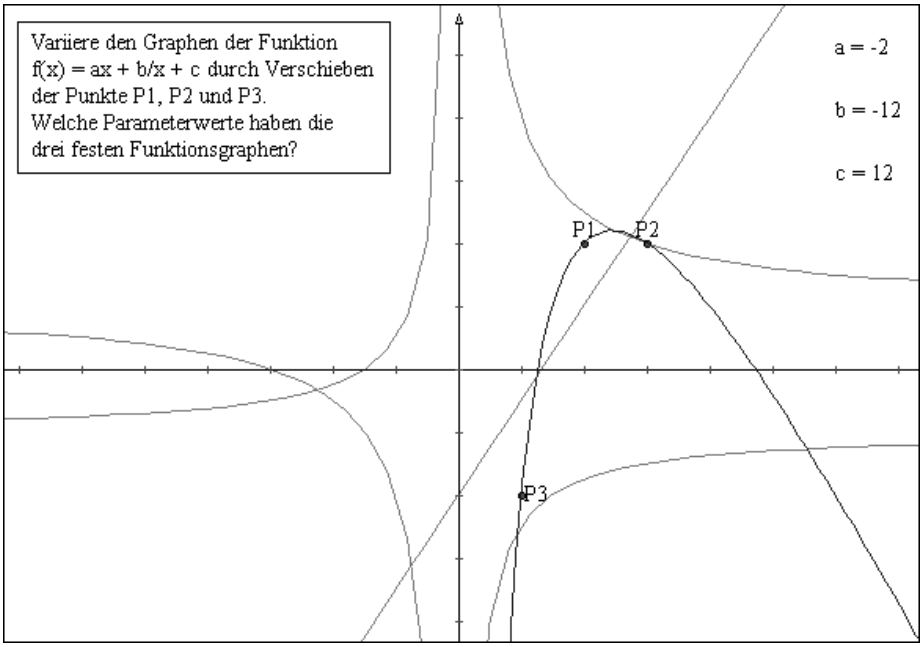

Abbildung 4.103: Interaktives Untersuchen der Funktionsgleichung durch Verändern der Form und Lage des Graphen.

Interaktives Untersuchen des Graphen durch Verändern der Funktionsgleichung Durch Variieren der Parameterwerte kann der Schüler untersuchen, wie sich der Funktionsgraph in Form und Lage verändert. Der Schüler kann dabei herausfinden, ob bestimmte Parameter beispielsweise eine horizontale oder vertikale Verschiebung bestimmen, ob sie für den Grad einer Öffnung verantwortlich sind oder wie sich die Schnittpunkte des Graphen mit den Achsen verschieben. Er kann prüfen, wie der Graph verläuft, wenn bestimmte Parameter den Wert 0, 1 oder -1 annehmen. Eventuell lassen sich auch bestimmte Kurvenscharen klassifizieren. Die Zusatzaufgabe aus dem vorherigen Lernbaustein läßt sich auch hier stellen: Zu einem vorgegebenen Graphen sollen die entsprechenden Scharparameter bestimmt werden (Abbildung 4.104).

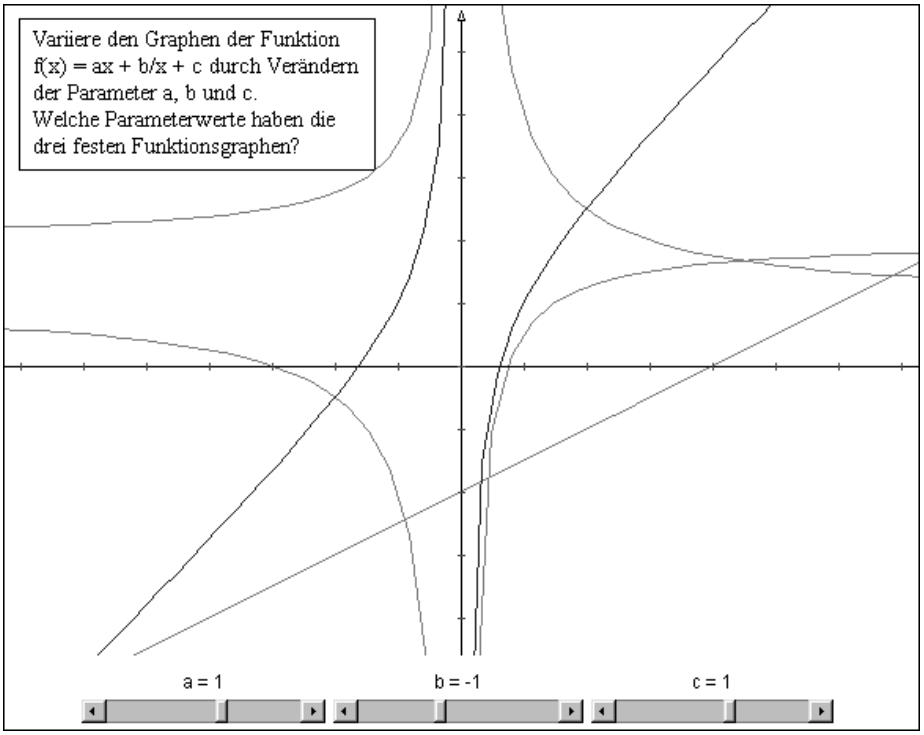

Abbildung 4.104: Interaktives Untersuchen des Graphen durch Verändern der Funktionsgleichung.

Modulares Generieren des Funktionsgraphen Mit dem folgenden Lernbaustein kann der Schüler experimentell nachvollziehen, wie sich die obige Funktion additiv zusammensetzen läßt. Addiert man die Funktionen $f_1(x) = ax$ und $f_2(x) = \frac{b}{x} + c$ so erhält man die ursprüngliche Funktionsgleichung. Die Abbildung 4.105 zeigt den auf der x-Achse verschiebbaren Punkt P, zu dem die Punkte (x, ax), $(x, \frac{b}{x} + c)$ und $(x, ax + \frac{b}{x} + c)$ dargestellt sind. Der Schüler kann

nun den entsprechenden Funktionsgraphen durch Aufzeichnen der Ortsspur von P punktweise generieren.

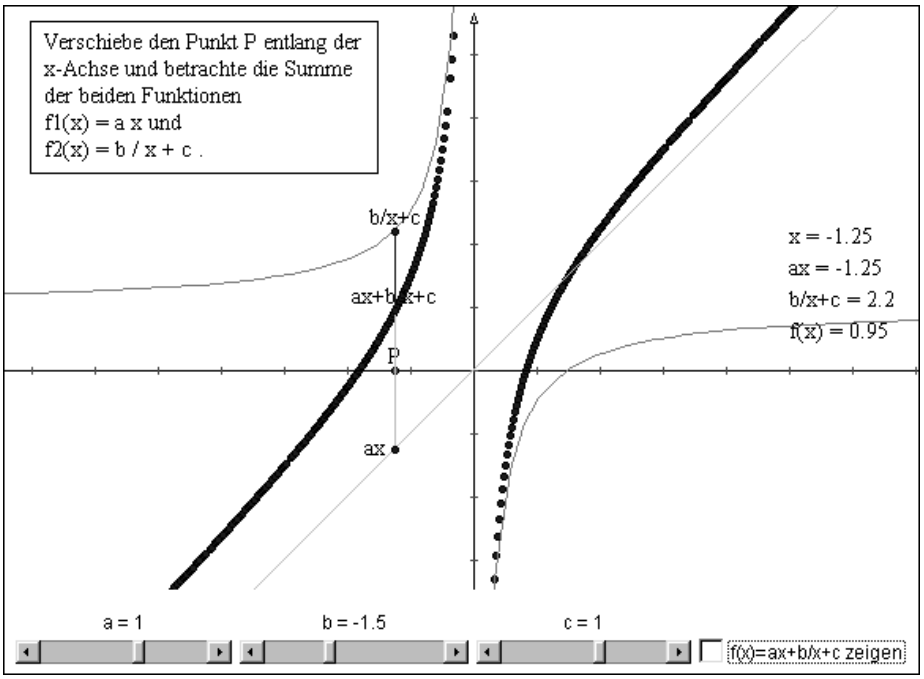

Abbildung 4.105: Modulares Generieren des Funktionsgraphen.

Interaktives Generieren der Umkehrfunktion Den Graphen der Umkehrfunktion kann man punktweise generieren, indem man einen ziehbaren Punkt auf einem Funktionsgraphen bewegt und ihn dabei an der ersten Winkelhalbierenden spiegelt. Die Ortsspur des Bildpunkts liefert den Graphen der Umkehrfunktion. Die Abbildung 4.106 zeigt einen Lernbaustein, in dem der Punkt P auf dem Graphen gezogen werden kann. Gleichzeitig wird dabei die Ortsspur von P' aufgezeichnet.

Interaktives Untersuchen von Tangenten Die Eigenschaften eines Funktionsgraphen lassen sich noch weiter untersuchen, indem die Tangente durch einen auf dem Graphen ziehbaren Punkt betrachtet wird. Dabei kann der Schüler beispielsweise prüfen, wie sich die Tangentensteigung bei zunehmendem x-Wert verhält, wann die Tangente fällt und wann sie steigt. Er kann näherungsweise die Punkte bestimmen, durch die die Tangente horizontal verläuft (Extrem- oder Wendepunkte). Die Abbildung 4.107 zeigt die Tangente an den Funktionsgraphen durch den Punkt P. Als numerischer Parameter wird zusätzlich die Tangentensteigung $f'(x)$ angezeigt.

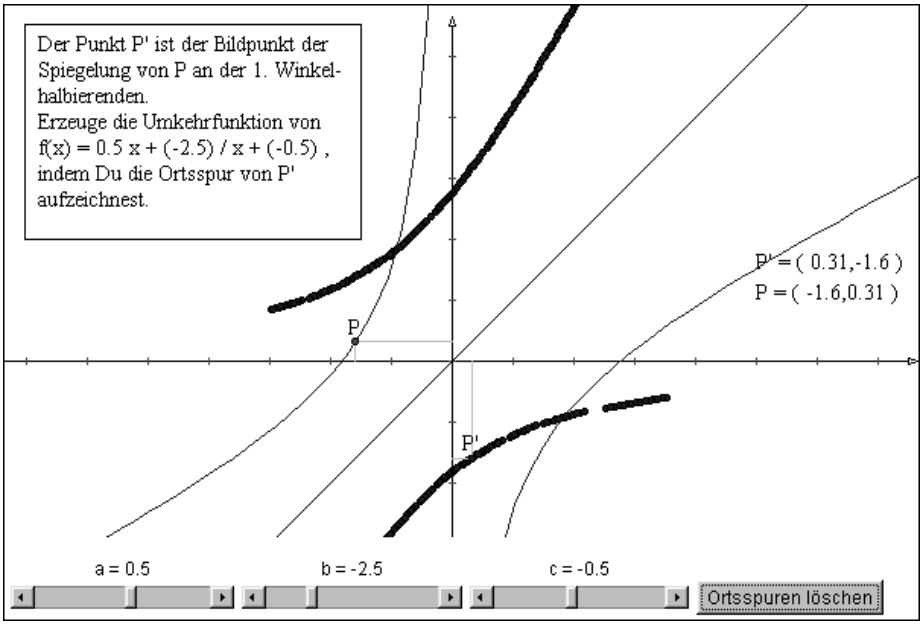

Abbildung 4.106: Interaktives Generieren der Umkehrfunktion.

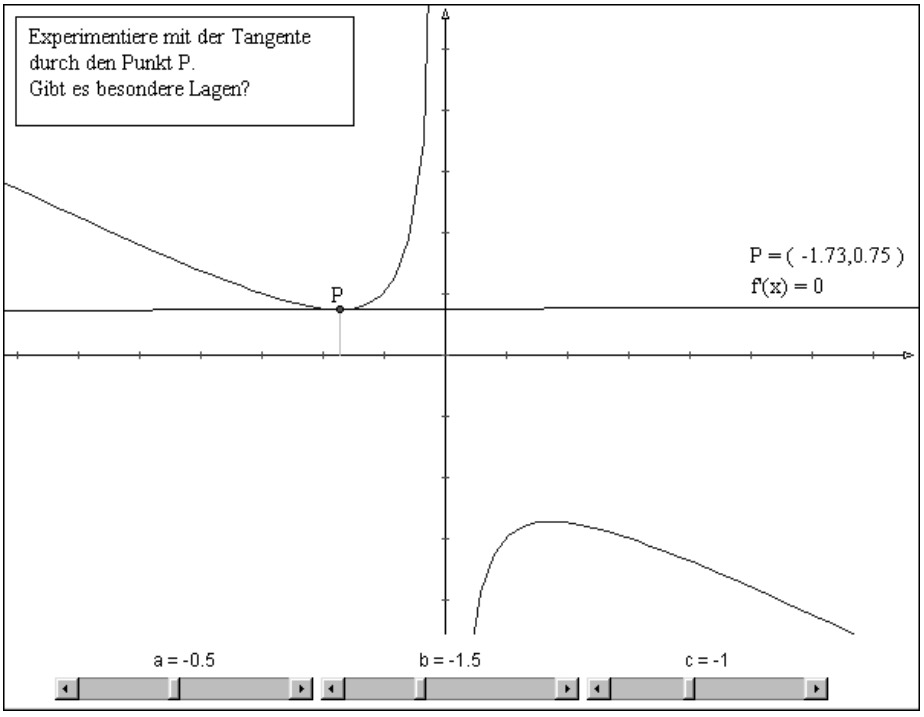

Abbildung 4.107: Interaktives Untersuchen von Tangenten.

Eine algebraische Kurve

Im folgenden möchte ich an einem Beispiel zeigen, wie Lernbausteine eingesetzt werden können, um algebraische Kurven zu diskutieren. Die folgenden Ausführungen beziehen sich auf die von Schupp & Dabrock[35] durchgeführte Diskussion der Kurve $y^2 = ax^4 + bx^2$ mit $a, b \in \mathbb{R}$. Dazu habe ich einen Lernbaustein entwickelt, in dem die algebraische Kurve durch eine externe Kurvenklasse realisiert ist (Seite 391). Durch zwei Schieberegler können die Parameter a, b eingestellt werden, dabei ändern sich Form und Lage der Kurve entsprechend (Abbildung 4.108-4.112). Für eine detaillierte Diskussion verweise ich auf die obige Arbeit, hier möchte ich nur die wichtigsten Ergebnisse wiedergeben, die durch interaktives Untersuchen erarbeitet werden können. Abhängig von den Parameterwerten kann man vier Fälle unterscheiden:

1. $a \geq 0$ und $b \geq 0$
   Es lassen sich wiederum drei Fälle unterscheiden. Ist $a = 0$ und $b = 0$, so entspricht die Kurve der x-Achse (Abbildung 4.108). Ist $a > 0$ und $b = 0$, dann ergibt sich eine Doppelparabel $y = \pm\sqrt{a}x^2$ (Abbildung 4.109). Wenn $a = 0$ und $b > 0$ ist, entsteht eine Doppelgerade $y = \pm bx$ (Abbildung 4.110).

2. $a > 0$ und $b < 0$
   Hier besteht die Kurve aus zwei getrennten, zueinander symmetrischen Ästen und dem Koordinatenursprung (Abbildung 4.111).

3. $a < 0$ und $b = 0$
   Die Kurve entartet zu einem Punkt, der im Koordinatenursprung liegt (ohne Abbildung).

4. $a < 0$ und $b > 0$
   Die Kurve verläuft schleifenförmig, wie in Abbildung 4.112 dargestellt.

Hält man $a > 0$ konstant und variiert b, so kann man erkennen, wie sich die Kurve von einer Doppelgeraden, über eine geschwungene X-Form, zu einer Doppelparabel verändert und schließlich eine hyperbelähnliche Form annimmt. Eine weitere Metamorphose ergibt sich, wenn man $b > 0$ konstant hält und a variiert.

Flächeninhalt unter einer Kurve

Mit Hilfe einer speziell entwickelten, externen Java-Klasse kann der Flächeninhalt unter einer Kurve berechnet werden.[36] Diese Klasse wird durch *GeoScript* aufgerufen, wobei als Parameter ein Kurvenobjekt, die Intervallgrenzen und die Anzahl der Stützpunkte zu übergeben sind. Als Beispiele möchte ich zwei Lernbausteine vorstellen, in denen interaktiv der Flächeninhalt unter einer Kurve untersucht werden kann. Eine solche Aufgabe könnte etwa im Anschluß an eine Kurvendiskussion (wie zuvor beschrieben) erfolgen. Die Lernbausteine eignen sich dazu, rechnerische Ergebnisse selbsttätig anhand der Figur zu überprüfen.

[35] Schupp & Dabrock 1995, S. 362-366

[36] Dabei handelt es sich um die Klasse `Functional_Integral` (Seite 393). Bei deren Programmierung habe ich die Java-Bibliothek *JavaSci* von Mark Hale u. a. verwendet, die zahlreiche mathematische Funktionen bereitstellt, darunter auch Funktionen zur numerischen Integration.

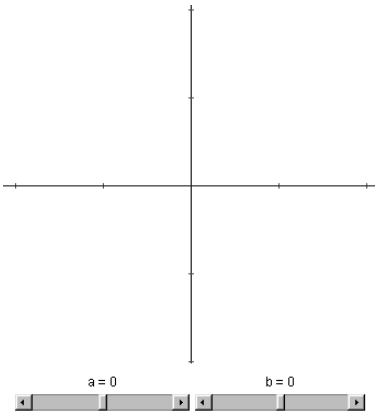

Abbildung 4.108: Gerade $y = 0$.

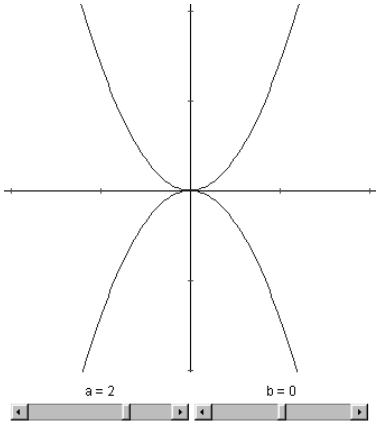

Abbildung 4.109: Doppelparabel.

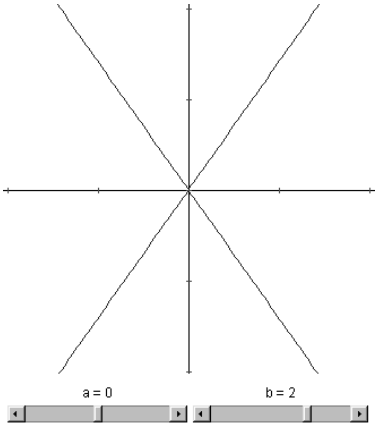

Abbildung 4.110: Doppelgerade.

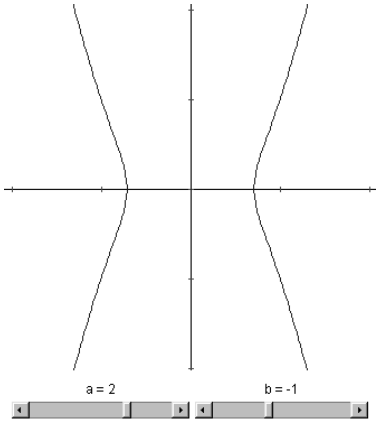

Abbildung 4.111: Hyperbelähnliche Form.

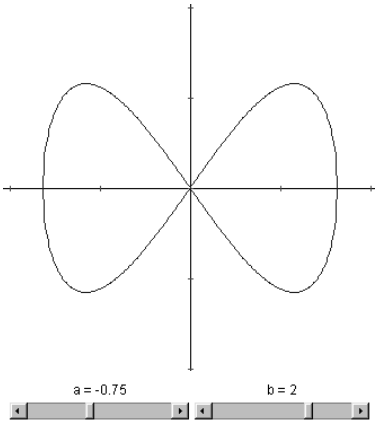

Abbildung 4.112: Schleifenform.

Umgekehrt können auch an der Figur gemessene Werte durch schriftliche Berechnungen bestätigt werden.

Das erste Beispiel enthält ein Kurvenobjekt, das den Graphen der Exponentialfunktion $f(x) = e^{ax+b}$ darstellt. Die Parameter a und b können durch Schieberegler eingestellt werden. Zwei auf der x-Achse verschiebbare Punkte P_1 und X legen die Intervallgrenzen des zu berechnenden Integrals fest. Der Wert des Integrals ist in der Zeichenfläche angezeigt und ändert sich entsprechend, wenn ein Parameter verändert wird. Zusätzlich wird in dem Lernbaustein der Graph der Funktion

$$F(X) = \int_{P_1}^{X} f(x)\ dx$$

dargestellt (Abbildung 4.113).

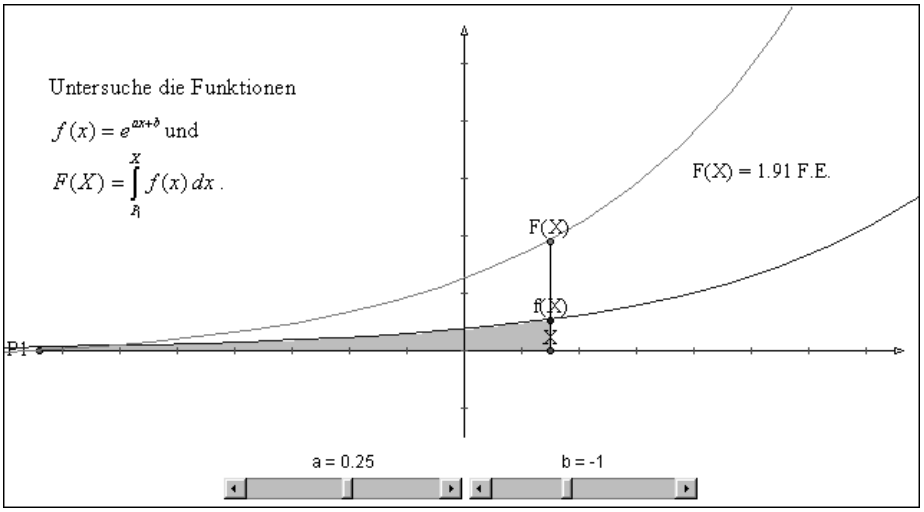

Abbildung 4.113: Integration von $f(x) = e^{ax+b}$.

Der zweite Lernbaustein zur Integration stellt den Graphen von

$$f(x) = |a - x^2| - \ x \ + \ b$$

dar. Wie in dem ersten Beispiel können die Parameter a, b durch Schieberegler und die Intervallgrenzen durch zwei ziehbare Punkte P_1 und P_2 variiert werden (Abbildung 4.114).

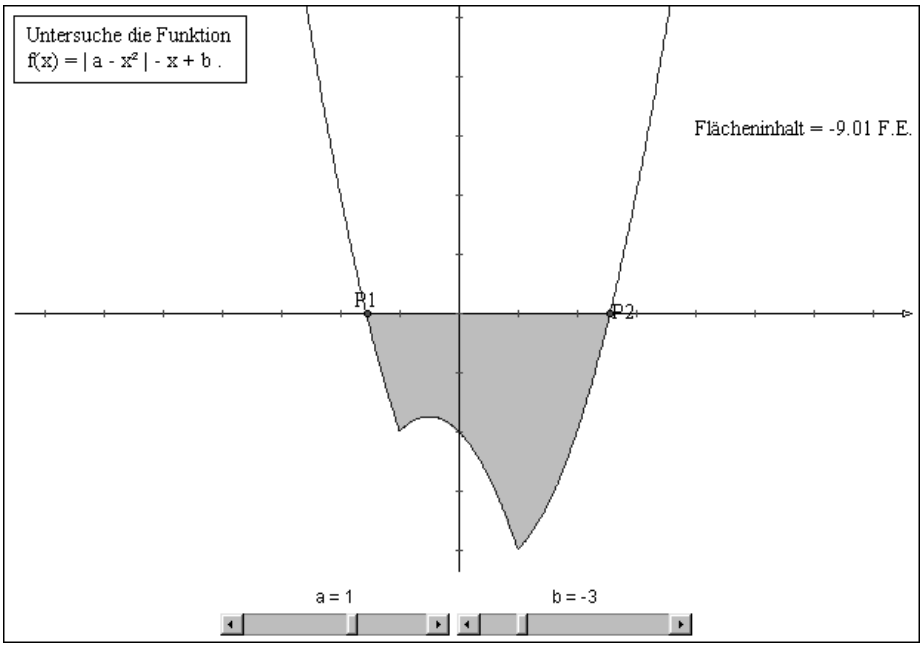

Abbildung 4.114: Integration von $f(x) = |a - x^2| - x + b$.

Spiralen

Spiralen sind eine Klasse von ästhetisch anmutenden Kurven, die leicht in Lernbausteinen dargestellt werden können. Definiert werden Spiralen durch eine Gleichung in Polarkoordinaten in der Form $r = f(\varphi)$. Im folgenden möchte ich vier Arten von Spiralen vorstellen: die Archimedische Spirale $r = a\varphi$ (Abbildung 4.115), die Logarithmische Spirale $r = c^{a\varphi}$ (Abbildung 4.116), die Hyperbolische Spirale $r = \frac{a}{\varphi}$ (Abbildung 4.117) und die Fermatsche Spirale $r = a\sqrt{\varphi}$ (Abbildung 4.118).

Wie schon bei den zuvor beschriebenen Kurven lassen sich die jeweiligen Parameter durch Schieberegler variieren. Dabei kann man gleichzeitig verfolgen, wie sich die Form und Lage der Spirale verändert.

Für eine Beschreibung der didaktisch-methodischen Behandlung dieser Kurvenklasse möchte ich auf zwei Arbeiten verweisen. Eine ausführliche Diskussion der Eigenschaften von Spiralen geben Schupp & Dabrock[37], hier findet man Anregungen für eine analytische Untersuchung. Einen handlungsorientierten Zugang zu diesem Thema beschreibt Gardner[38], in dem er Verfahren und mechanische Vorrichtungen zeigt, mit denen man Spiralen zeichnen kann.

[37] Schupp & Dabrock 1995, S. 212-227
[38] Gardner 1971, S. 90-100

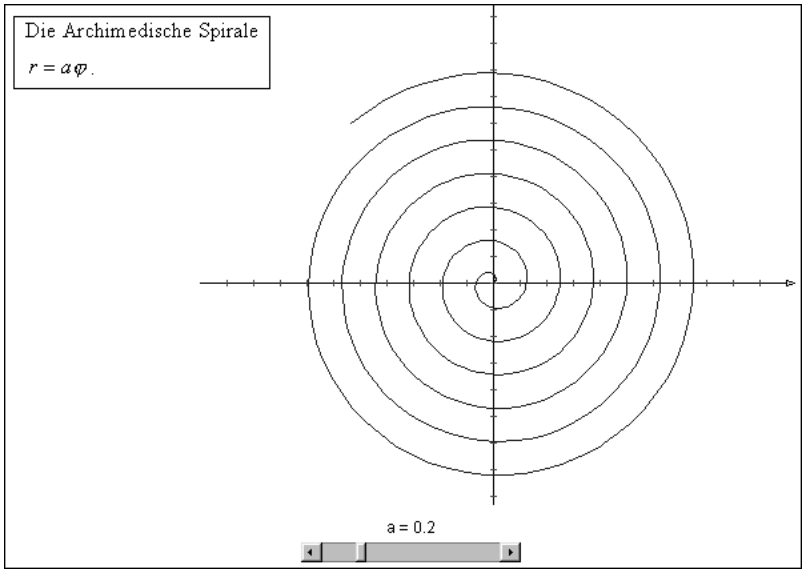

Abbildung 4.115: Die Archimedische Spirale.

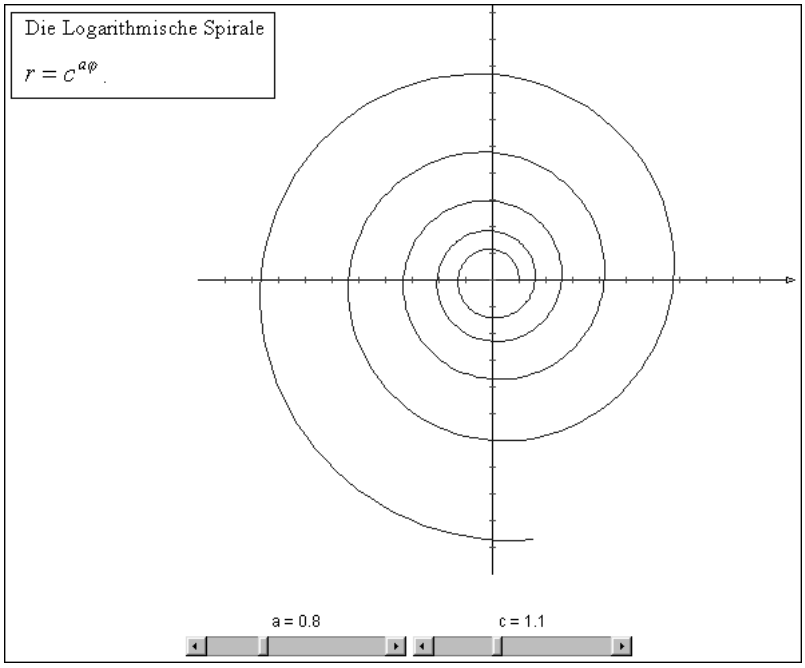

Abbildung 4.116: Die Logarithmische Spirale.

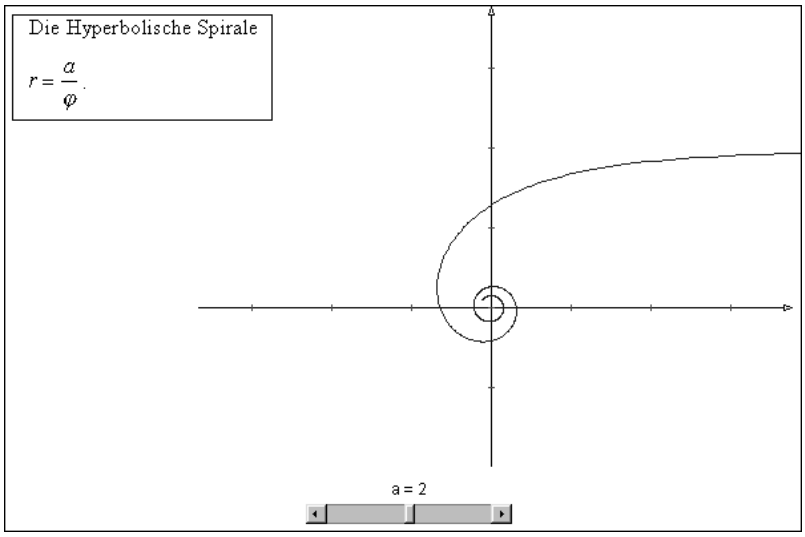

Abbildung 4.117: Die Hyperbolische Spirale.

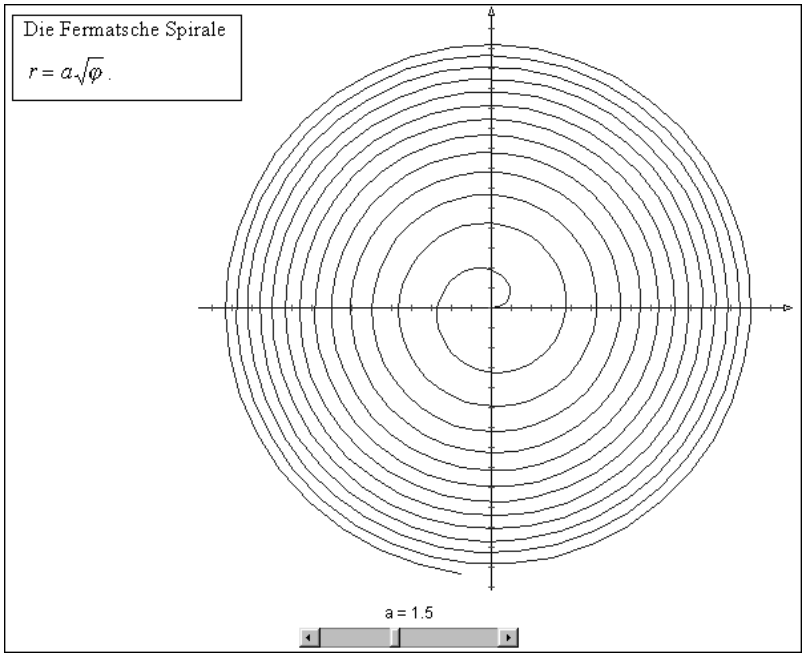

Abbildung 4.118: Die Fermatsche Spirale.

4.4.5 Ortslinien

Die Möglichkeit im Zugmodus Ortslinien abhängiger Punkte zu untersuchen, zählt zu den besonderen Eigenschaften von DG-Systemen. Während bei den zuvor beschriebenen Kurven in erster Linie numerische Werte als Eingangsparameter dienten, sind es beim Arbeiten mit Ortslinien vor allem Punktobjekte, die auf bestimmten Bahnen bewegt werden.

In der Literatur (z. B. Weth 1991, Wurm 1996, Hölzl & Schneider 1996, Hölzl 1999, Henn 1995, Henn 1996, Weigand 1996, Schwarze 1996) sind bereits zahlreiche Vorschläge zum Arbeiten mit Ortslinien gemacht worden. Im folgenden möchte ich deshalb nur zwei Beispiele für das interaktive Untersuchen von Ortslinien geben.

Kissoide

Man erhält eine Kissoide nach der folgenden Konstruktion. Gegeben seien zwei Kurven k_1 und k_2 sowie ein Punkt P. Eine Gerade l durch P schneidet k_1 in B und k_2 in G. Dreht man die Gerade l um P, verändern sich die Schnittpunkte B und G. Die Strecke BG wird in P entlang der Geraden l in beide Richtungen abgetragen. Die Streckenendpunkte werden mit A_1 und A_2 bezeichnet. Die Ortslinie, die A_1 (oder A_2) beschreibt, wenn B auf k_1 bewegt wird, heißt Kissoide.

In der folgenden Figur ist für die Kurve k_1 ein Kreis und für die Kurve k_2 eine Gerade gewählt. Der Punkt P liegt auf k_1. Die Strecke PA wird in Richtung BG abgetragen. Die Ortslinie von A wird als Kissoide des Diokles bezeichnet (Abbildung 4.119). Verschiebt man nun die Gerade k_2 in Richtung auf P, so ändert sich die Ortslinie von A. Die Abbildungen 4.119-4.123 zeigen einige Ausprägungen der Kissoide.

Eine ausführliche Kurvendiskussion der Kissoide wird von Brieskorn[39] gegeben. Eine ähnliche *Cabri*-Konstruktion haben bereits Schumann & Green[40] vorgestellt.

Bézier-Kurven

In den 60er Jahren entwickelten von de Casteljau und Bézier, zwei französische Ingenieure aus dem Automobilbau, ein Verfahren, um Kurven durch möglichst wenige Punkte zu beschreiben.[41]

Um solche sog. Bézier-Kurven zu generieren, benötigt man im einfachsten Fall drei Punkte A, B, C. Ein Punkt P_1 teilt AB im Teilverhältnis $t = \frac{AP_1}{P_1B}$. Auf der Strecke BC ist durch t ein Punkt P_2 definiert, mit $t = \frac{BP_2}{P_2C}$, ebenso ist durch t auf P_1P_2 ein Punkt P_3 festgelegt (Abbildung 4.125). Verschiebt man P_1 auf der Strecke AB, so ist die Ortslinie von P_3 die Bézier-Kurve von A, B, C (Abbildung 4.126). Diese Konstruktion läßt sich auch auf mehr als drei Ausgangspunkte anwenden. Die Abbildungen 4.127 und 4.128 zeigen eine Bézier-Kurve mit vier Ausgangspunkten, und die Abbildungen 4.129 und 4.130 eine mit sechs Ausgangspunkten. Durch Variation können nun komplexe geschwungene Kurvenprofile interaktiv untersucht werden.

[39] Brieskorn 1981
[40] Schumann & Green 1997, S. 79-87
[41] Meyer 1997, S. 90-95

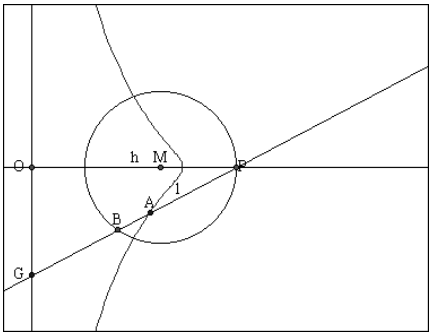

Abbildung 4.119: Bewegt man den Punkt B entlang der Kreisbahn, so heißt die Ortslinie von A Kissoide.

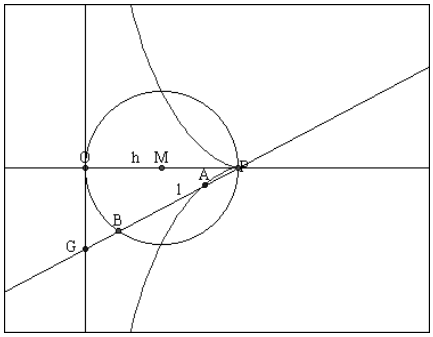

Abbildung 4.120: Berührt die Gerade k_2 den Kreis k_1 in O, so besitzt die Kissoide eine Spitze in P.

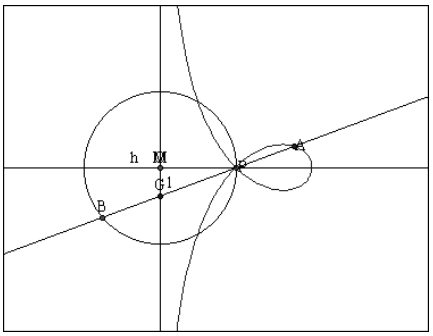

Abbildung 4.121: Verläuft die Gerade k_2 durch den Kreismittelpunkt M, wird eine Newtonsche Strophodie generiert.

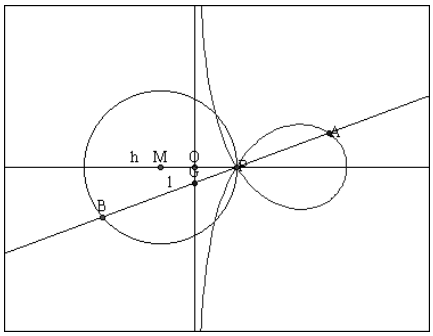

Abbildung 4.122: Wandert die Gerade k_2 weiter in Richtung P, so vergrößert sich die Schlaufe.

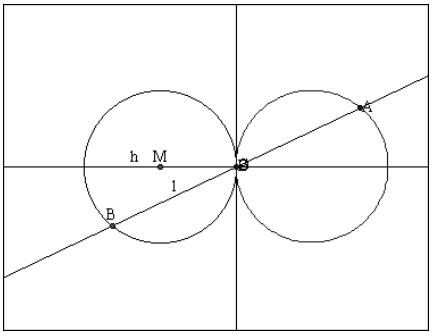

Abbildung 4.123: Verläuft die Gerade k_2 durch P, wird die Kissoide zum Spiegelbild des Kreises k_1.

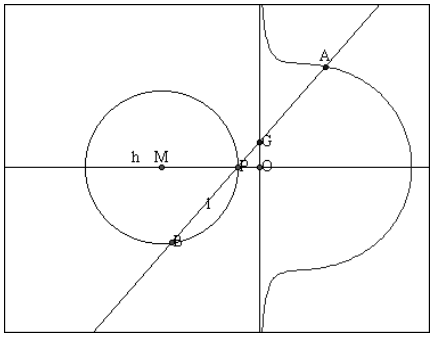

Abbildung 4.124: Wandert die Gerade k_2 über P hinaus, so entsteht eine beulenartige Form.

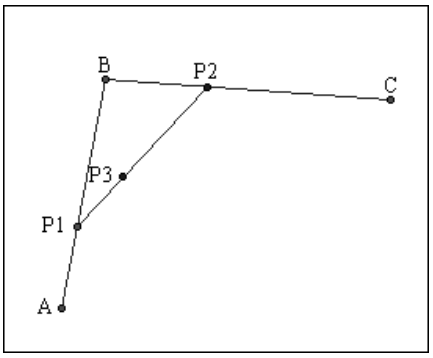

Abbildung 4.125: Der Punkt P_1 kann auf der Strecke AB verschoben werden. Für die Teilverhältnisse gilt: $\frac{AP_1}{P_1B} = \frac{BP_2}{P_2C} = \frac{P_1P_3}{P_3P_2}$.

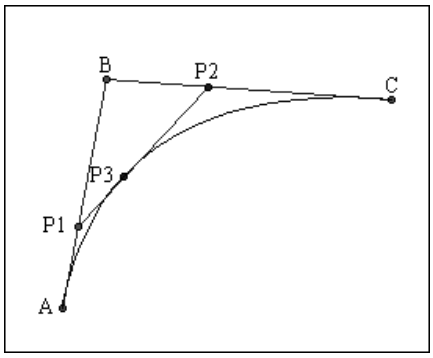

Abbildung 4.126: Die Ortslinie von P_3 ist die Bézier-Kurve zu ABC.

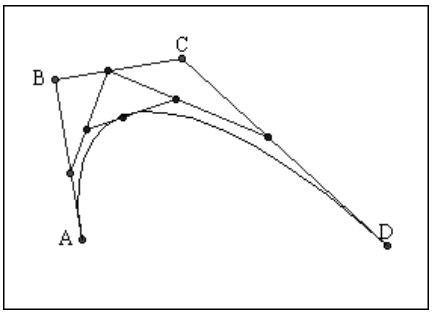

Abbildung 4.127: Konstruktion einer Bézier-Kurve aus vier frei ziehbaren Punkten.

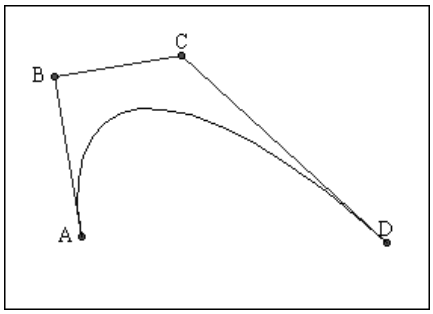

Abbildung 4.128: Bézier-Kurve aus Abbildung 4.127 ohne Hilfskonstruktion.

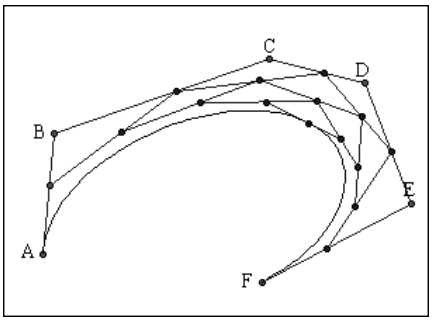

Abbildung 4.129: Konstruktion einer Bézier-Kurve aus sechs frei ziehbaren Punkten.

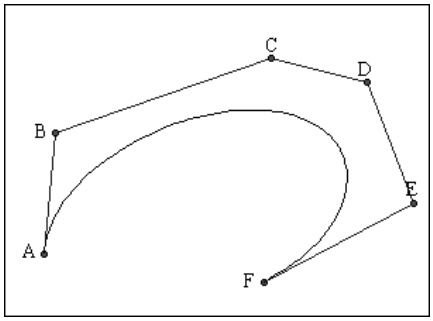

Abbildung 4.130: Bézier-Kurve aus Abbildung 4.129 ohne Hilfskonstruktion.

4.4.6 Fraktale Geometrie

Die fraktale Geometrie ist ein relativ junger Bereich der Mathematik, der durch den Einsatz des Computers deutlich vorangetrieben wurde.[42] Die besondere Ästhetik von Fraktalen, die vor allem in der Selbstähnlichkeit begründet liegt, läßt sich auch durch entsprechende Lernbausteine nachempfinden. In der Beschreibung der Klasse `CurveElement` in Abschnitt 3.3.6 habe ich bereits erwähnt, daß mit Hilfe von speziell entwickelten externen Java-Klassen Kurven im weiteren Sinne realisiert werden können. Durch solche Klassen lassen sich auch Fraktale erzeugen (Seite 394, 396 und 398). Im folgenden werde ich dies an drei Beispielen zeigen.

Koch-Kurve

Als Koch-Kurve wird, nach dem schwedischen Mathematiker H. von Koch, die Grenzkurve einer Folge bezeichnet, die sich nach dem folgenden Algorithmus ergibt: Eine durch zwei Endpunkte festgelegte Strecke (Abbildung 4.131) wird gedrittelt. Anschließend wird der mittlere Streckenabschnitt gelöscht und durch zwei Schenkel eines gleichseitigen Dreiecks ersetzt (Abbildung 4.132). Die Kurve besteht jetzt aus vier Teilstrecken, auf die jeweils das Verfahren erneut angewendet wird. Die Abbildungen 4.133 und 4.134 zeigen die dritte und siebte Stufe der Iteration.

Zu dem Lernbaustein ist hervorzuheben, daß nicht nur die beiden Endpunkte im Zugmodus verschoben werden können. Im Unterschied zu Demo-Figuren mit *Cabri-Géomètre II* und *Sketchpad* läßt sich auch die Anzahl der Iterationsstufen interaktiv verändern.

Sierpinski-Dreieck

Ein Sierpinski-Dreieck (nach dem polnischen Mathematiker W. Sierpinski) erhält man, indem ein gleichseitiges Dreieck in vier kongruente Teildreiecke unterteilt wird (Abbildungen 4.135 und 4.136). Dieser Teilungsprozeß wird anschließend auf die drei äußeren Dreiecke erneut angewendet (Abbildung 4.137). Die Abbildung 4.138 zeigt die siebte Stufe dieses Iterationsprozesses.

Durch das interaktive Variieren der Iterationsstufen läßt sich gut nachvollziehen, wie die Figurenfolge gegen das Sierpinski-Dreieck strebt.

Fraktaler Baum

Fraktale Bäume oder Farne lassen sich durch verschiedene Verfahren erzeugen (etwa als Lindenmayer-Systeme[43]). Im folgenden Lernbaustein wird ein fraktaler Baum durch fünf Punkte definiert (Abbildung 4.139). Die Strecke AE stellt dabei den Stamm und die Strecken EB, EC und ED die Äste dar. Mit jeder neuen Iterationsstufe wachsen pro Ast drei neue Äste. Die Abbildungen 4.140 und 4.141 zeigen den Baum in der dritten und sechsten Stufe.

Anders als bei den vorherigen Fraktalen können durch Verschieben der fünf Ausgangspunkte nicht nur die Größe und Lage des Baums variiert werden. Auch seine Form läßt sich verändern, da die vorhandenen Streckenverhältnisse und Winkel diese beeinflussen.

[42] Peitgen 1992, S. 1-19
[43] Heigl 1998

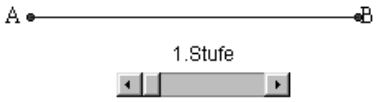

Abbildung 4.131: Ausgangsstrecke zur Konstruktion einer Koch-Kurve.

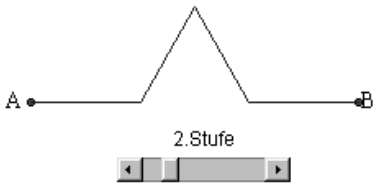

Abbildung 4.132: Die Ausgangsstrecke wird gedrittelt, das mittlere Drittel gelöscht und durch zwei Dreiecksschenkel ersetzt.

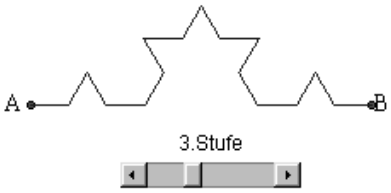

Abbildung 4.133: Die dritte Stufe einer Koch-Kurve.

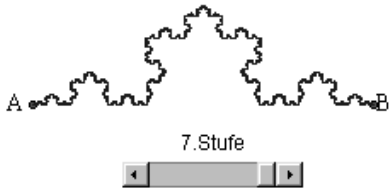

Abbildung 4.134: Die siebte Stufe einer Koch-Kurve.

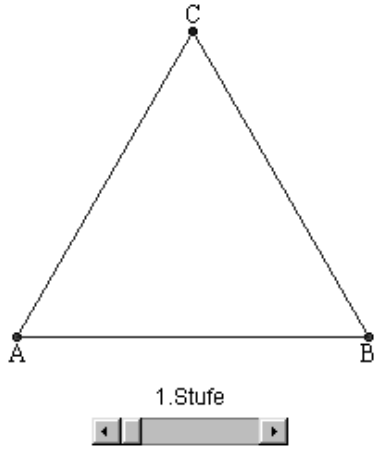

Abbildung 4.135: Ausgangsdreieck zur Konstruktion eines Sierpinski-Dreiecks.

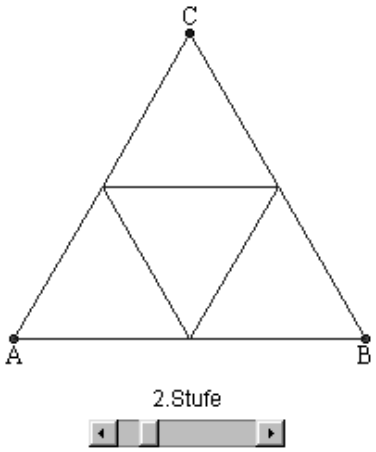

Abbildung 4.136: Das Ausgangsdreieck wird in vier kongruente Teildreiecke unterteilt.

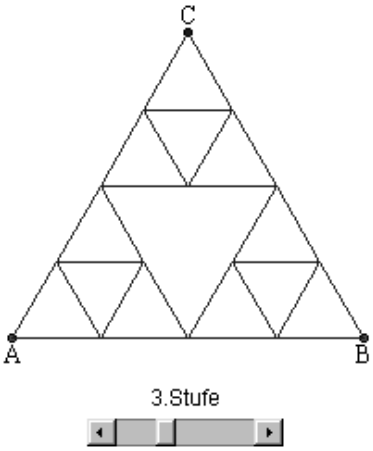

Abbildung 4.137: Die dritte Stufe eines Sierpinski-Dreiecks.

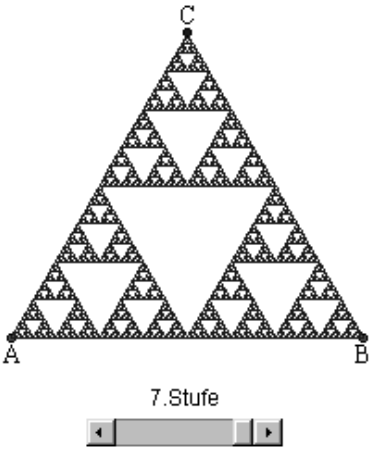

Abbildung 4.138: Die siebte Stufe eines Sierpinski-Dreiecks.

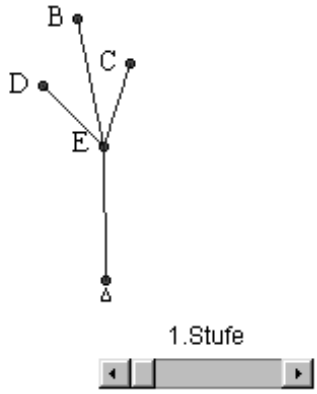

Abbildung 4.139: Fraktaler Baum definiert durch fünf Punkte.

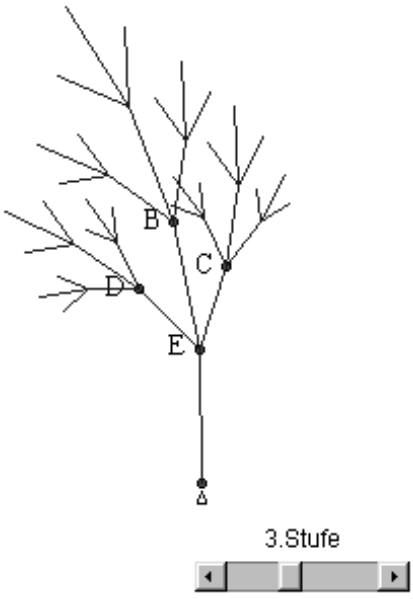

Abbildung 4.140: Mit jeder Stufe wachsen pro Ast drei neue Äste.

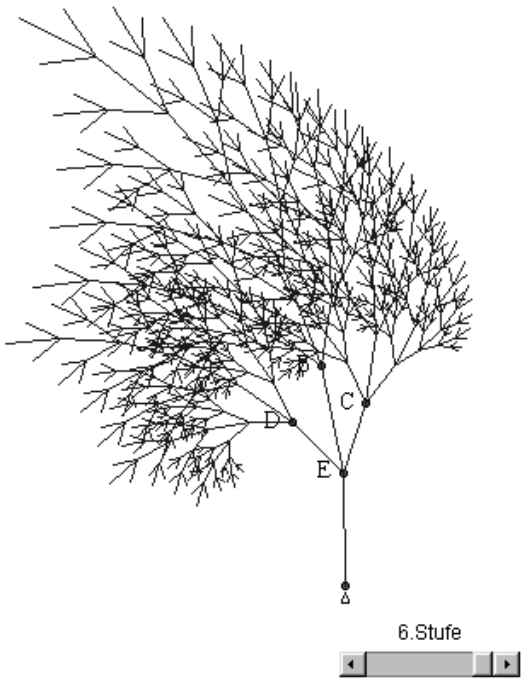

Abbildung 4.141: Fraktaler Baum nach der sechsten Iterationsstufe.

4.4.7 Metriken

Ein weiteres lohnenswertes Thema für Lernbausteine sind ebene metrische Räume, in denen der Begriff des Abstands zweier Punkte neu definiert wird. Durch die Möglichkeit, eigene Funktionale mit *GeoScript* zu definieren, lassen sich im Prinzip beliebige Abstandsfunktionen aufstellen. Dadurch können metrische Räume simuliert werden, in denen der Schüler untersuchen kann, welche Ausprägungen Begriffe annehmen, wenn man sie aus der euklidischen in eine andere Metrik überträgt. Hierzu kann man experimentieren, wie etwa ein Kreis, eine Ellipse, eine Parabel, ein Lot, eine Mittelsenkrechte in einer bestimmten Metrik aussieht. Auch lassen sich Extremwertaufgaben stellen, in denen kürzeste Wege zu bestimmen sind.

Im folgenden beschreibe ich exemplarisch vier Lernbausteine zum Thema Metriken. Die Aufgabenideen stammen aus einem Artikel von Claus[44] über Extremwertaufgaben in metrischen Räumen.

Ellipse in der Taxi-Metrik

Bei dem folgenden Lernbaustein geht es darum, die Form einer Ellipse in der Taxi-Metrik zu bestimmen. Dazu ist in der Zeichenfläche ein Gitternetz eingeblendet, auf dem die Eckpunkte $ABCDEF$ eines Sechsecks verschoben werden können. Auf dem Polygon-Kantenzug kann ein Punkt P bewegt werden. Ferner sind zwei feste Punkte M_1 und M_2 vorgegeben. Das Sechseck soll nun so variiert

[44] Claus 1982, S. 27-58

werden, daß es in der Taxi-Metrik eine Ellipse mit den beiden Brennpunkten M_1 und M_2 bildet. Mit Hilfe des Punkts P und einem Funktional, das die Abstände PM_1 und PM_2 mißt, kann der Schüler experimentell vorgehen und anschließend seine Lösung selbst kontrollieren (Abbildung 4.142).

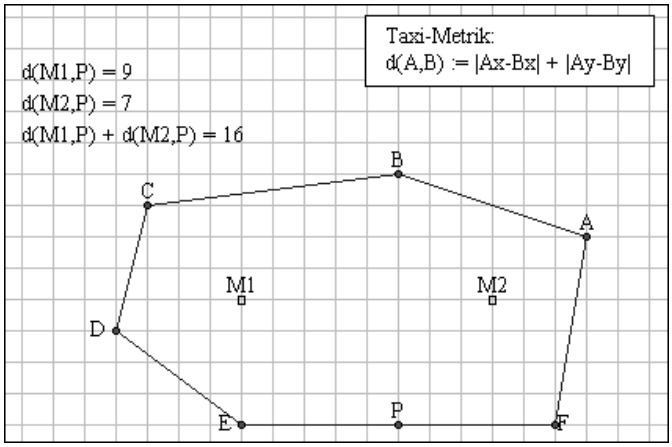

Abbildung 4.142: Das Sechseck $ABCDEF$ soll zu einer Ellipse mit den Brennpunkten M_1 und M_2 werden. Mit dem Punkt P kann der Abstand kontrolliert werden.

Abstandssumme in der Taxi-Metrik

Der folgende Lernbaustein enthält eine einfache Variationsaufgabe zur Taxi-Metrik. In der Zeichenfläche sind fünf ortsfeste Punkte $P_1, \ldots, P_5$ vorgegeben. Ein Punkt T soll auf dem Gitternetz so verschoben werden, daß die Summe aller Abstände TP_i $(i = 1, \ldots, 5)$ in der Taxi-Metrik minimal wird (Abbildung 4.143).

Um die Lösung der Aufgabe selbsttätig kontrollieren zu können, ist eine Antwortanalyse vorbereitet. Dabei werden neben den falschen Antworten auch Figurenzustände erkannt, bei denen der Punkt T um einen Rasterschritt von der richtigen Lösung abweicht, da er vielleicht nur versehentlich falsch plaziert wurde.

Kreis in der Maximum-Metrik

Ein Kreis ist definiert als die Menge aller Punkte, die von einem gegebenen Punkt einen konstanten Abstand besitzen. Die Frage, wie ein Kreis in der Maximum-Metrik aussieht, kann man mit dem folgenden Lernbaustein experimentell erarbeiten. Als Mittelpunkt ist ein fester Punkt M vorgegeben. Das Viereck $ABCD$ soll so verschoben werden, daß alle Punkte auf dem Kantenzug zu M einen konstanten Abstand besitzen. Mit Hilfe eines Punkts P, der auf dem Viereck $ABCD$ verschoben werden kann und zu dem der Maximum-Abstand von PM angezeigt wird, läßt sich die Lösung selbsttätig kontrollieren (Abbildung 4.144).

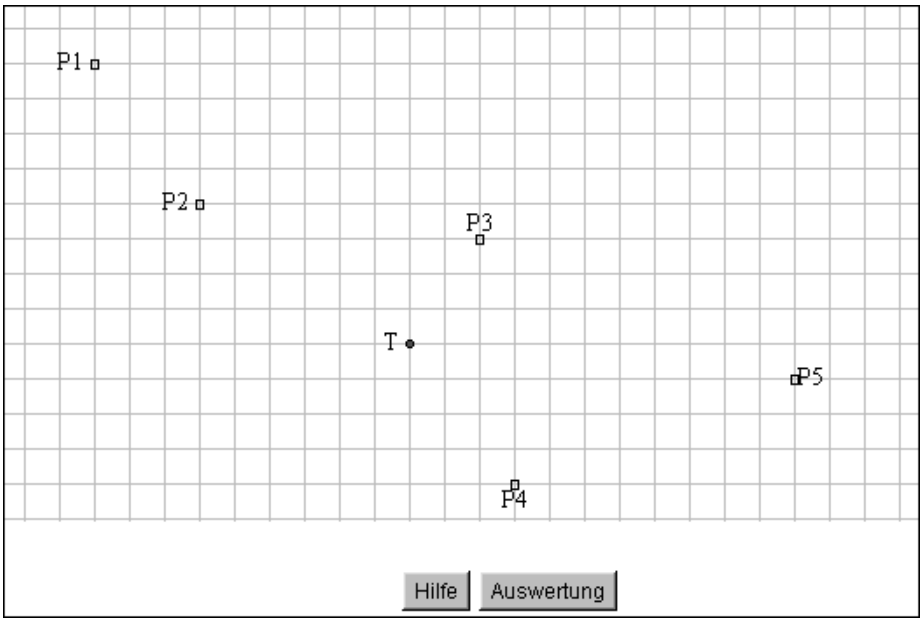

Abbildung 4.143: In der Stadt Orthopolis wollen sich die fünf Bewohner der Häuser $P_1, \ldots, P_5$ an dem Ort mit der kürzesten Abstandssumme treffen. Wo muß der Treffpunkt T liegen?

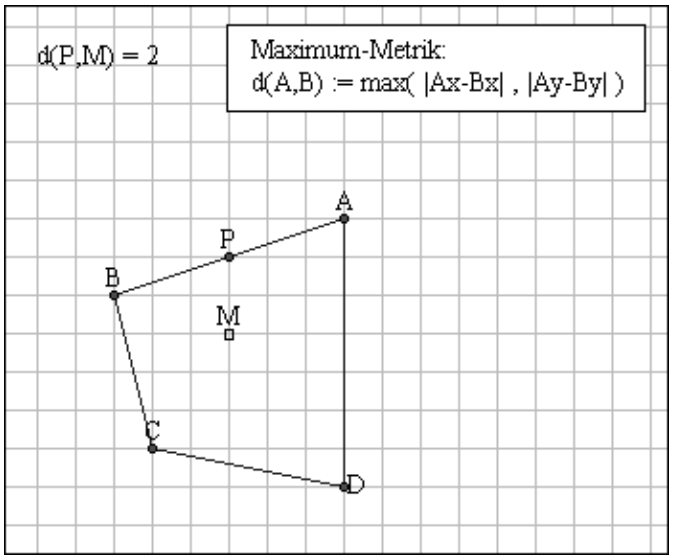

Abbildung 4.144: Kann man das Viereck $ABCD$ so variieren, daß es einen Kreis um M in der Maximum-Metrik bildet?

Eisenbahn-Metrik

Eine etwas weniger bekannte Metrik der Ebene ist die Eisenbahn-Metrik[45]. Will man in Frankreich mit der Eisenbahn reisen, muß man häufig über Paris fahren. Vor diesem Hintergrund läßt sich die folgende Abstandsdefinition einführen:

$$d_P(A,B) = \begin{cases} |AP| + |PB|, & \text{falls } P \notin AB \\ |AB|, & \text{falls } P \in AB \end{cases}$$

Die Figur in den Abbildungen 4.145 und 4.146 zeigt die beiden Fälle. Die Größe des Eisenbahn-Abstands wird dabei durch ein Funktional angezeigt.[46] Die Aufgabe könnte darin bestehen, herauszufinden, welche Form ein Kreis in der Eisenbahn-Metrik annimmt. Ferner läßt sich untersuchen, welche Bedeutung die Dreiecksungleichung in dieser Metrik hat oder wie geometrische Objekte (z. B. Mittelsenkrechte) aussehen.

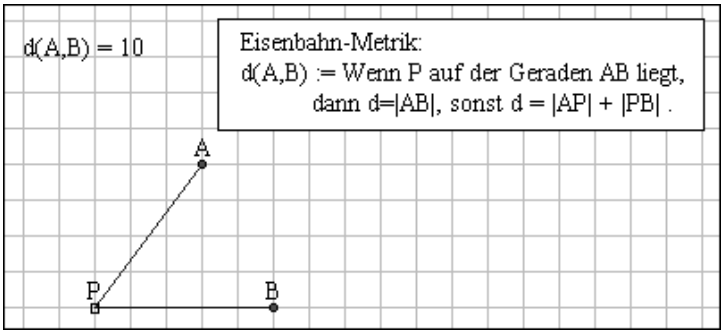

Abbildung 4.145: Erster Fall der Eisenbahn-Metrik: P liegt nicht auf AB.

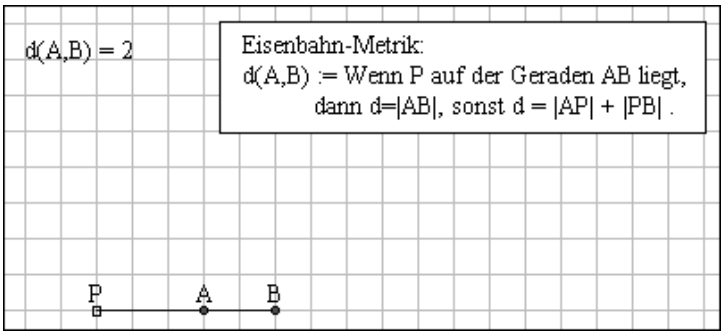

Abbildung 4.146: Zweiter Fall der Eisenbahn-Metrik: P liegt auf AB.

[45] Claus 1982, S. 52f

[46] Die Definition des Funktionals mit *GeoScript* ist ein Beispiel, bei dem eine Fallunterscheidung getroffen wird.

4.4.8 Verschiedenes

In diesem Abschnitt gebe ich einige Beispiele, die nicht direkt zur Geometrie zählen, aber sich dennoch gut durch Lernbausteine darstellen lassen.

Mengensprache

Die Abbildung 4.147 zeigt einen Lernbaustein zum Thema Mengensprache. Die Mengen A, B, C sind darin durch drei sich schneidende Kreise symbolisiert. Die ziehbaren Punkte $P_1, \ldots, P_8$ sind außerhalb der Mengen angeordnet. Sie sollen vom Schüler so verschoben werden, daß die Aussagen

1. $(A - B) \cup C = \{P_1, P_3, P_5, P_7\}$,
2. $(A \cup B) - (B \cup C) = \emptyset$,
3. $(A - B) + (A - C) = \{P_1, P_2, P_3, P_4\}$,
4. $A \cup (B - C) = \{P_1, P_2, P_3, P_4, P_6, P_8\}$ und
5. $(B \cup A) - (A \cap C) = \{P_2, P_4, P_6, P_8\}$

simultan erfüllt sind. Mit *GeoScript* lassen sich Prüfschlüssel definieren, mit denen man testen kann, in welchen der drei Mengen ein Punktobjekt enthalten bzw. nicht enthalten ist. Bei den zugehörigen Antwortwerten werden in der Aufgabe neben der richtigen Lösung auch acht falsche Antworttypen unterschieden. Dazu sind entsprechende Antwortkommentare vorbereitet, die den Schüler darauf hinweisen, welche der Punkte er falsch plaziert hat. Die Abbildung 4.148 zeigt eine Lösung der Aufgabe.

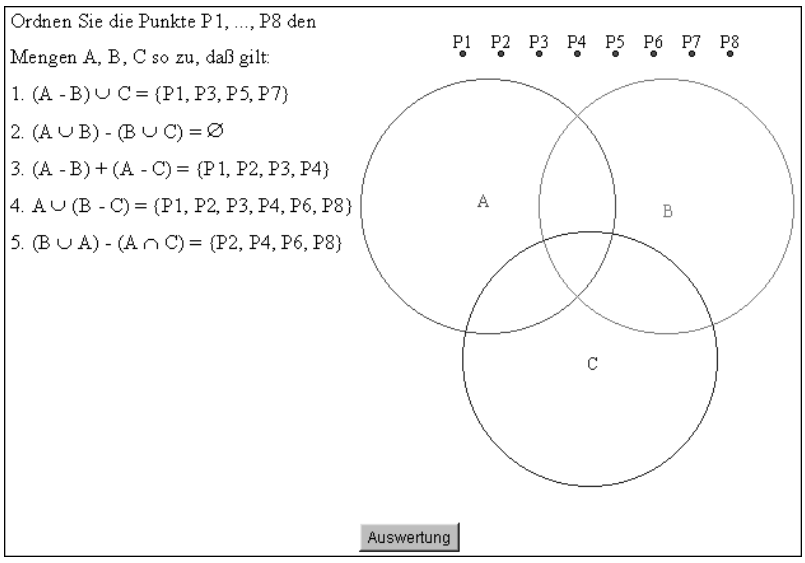

Abbildung 4.147: Aufgabe zum Thema Mengensprache.

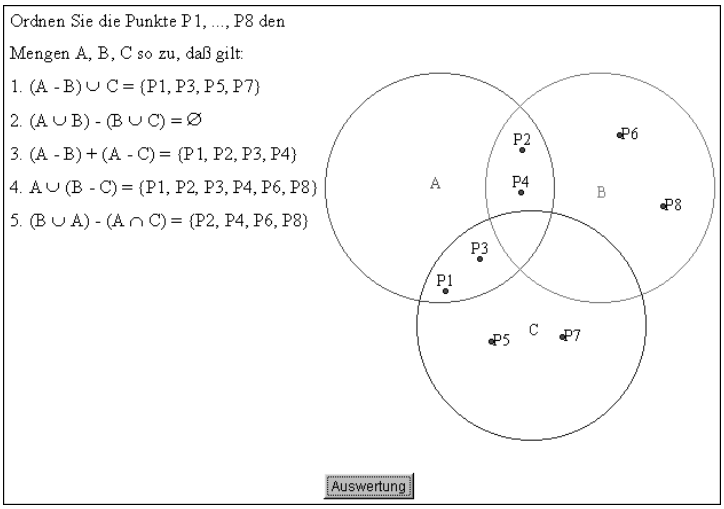

Abbildung 4.148: Lösung der Aufgabe aus Abbildung 4.147.

Labyrinthe

Labyrinthe sind ein beliebter Gegenstand der Unterhaltungsmathematik. Die Aufgabe besteht darin, den Weg von einem Ausgangspunkt in das Zentrum des Labyrinths oder vom Zentrum nach außen zu finden. In der Regel löst man diese Art von Aufgaben auf dem Papier, indem man den Weg mit einem Bleistift nachzeichnet. In einem Lernbaustein bietet es sich dagegen an, ein Punktobjekt in der Zeichenfläche zu verschieben und gleichzeitig die Ortsspur aufzuzeichnen. Dabei wird das Labyrinth durch eine Schwarz-Weiß-Grafik angezeigt. Damit das ziehbare Punktobjekt jedoch nicht auf einer Labyrinth-Wand gezogen werden kann, wird es an ein Punktmengenobjekt der Klasse `PointSetElement` gebunden. Die Punktmenge umfaßt dabei alle die Punkte, die nicht zur Labyrinth-Wand gehören. Man erhält diese, indem man die Grafik-Datei invertiert.

Die Abbildungen 4.149 und 4.150 zeigen zwei Figuren, in denen Labyrinthe in der oben beschriebenen Form realisiert sind. Die Vorlagen für die Grafiken stammen aus einem Buch von Dudeney[47], in dem er unter der Überschrift "Mazes and how to thread them" eine Reihe berühmter Labyrinthe vorstellt.

Tangram

Als Tangram bezeichnet man ein aus sieben Teilen bestehendes Puzzle, das ursprünglich aus China stammt. Tangram wird gespielt, indem man die Teile nebeneinander zu Figuren legt. Eine beliebte Aufgabenstellung ist aber auch, zu vorgegebenen Umrißfiguren herauszufinden, wie diese gelegt werden können.

Die Abbildungen 4.151 und 4.152 zeigen, wie die Tangram-Teile in einem Lernbaustein umgesetzt werden können. Weil bei einem Tangram aus Holz das Parallelogramm gewendet werden kann, muß diese Möglichkeit auch in einem Lernbaustein realisiert werden. Dazu ist ein Schalter vorgesehen, mit dem man zwischen den beiden Lagen umschalten kann (Abbildung 4.153 und 4.154).

[47] Dudeney 1917, S. 127-136

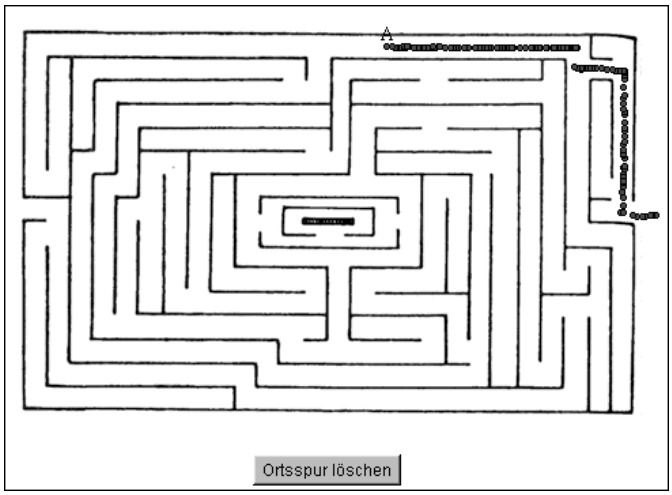

Abbildung 4.149: Welcher Weg führt in das Zentrum?

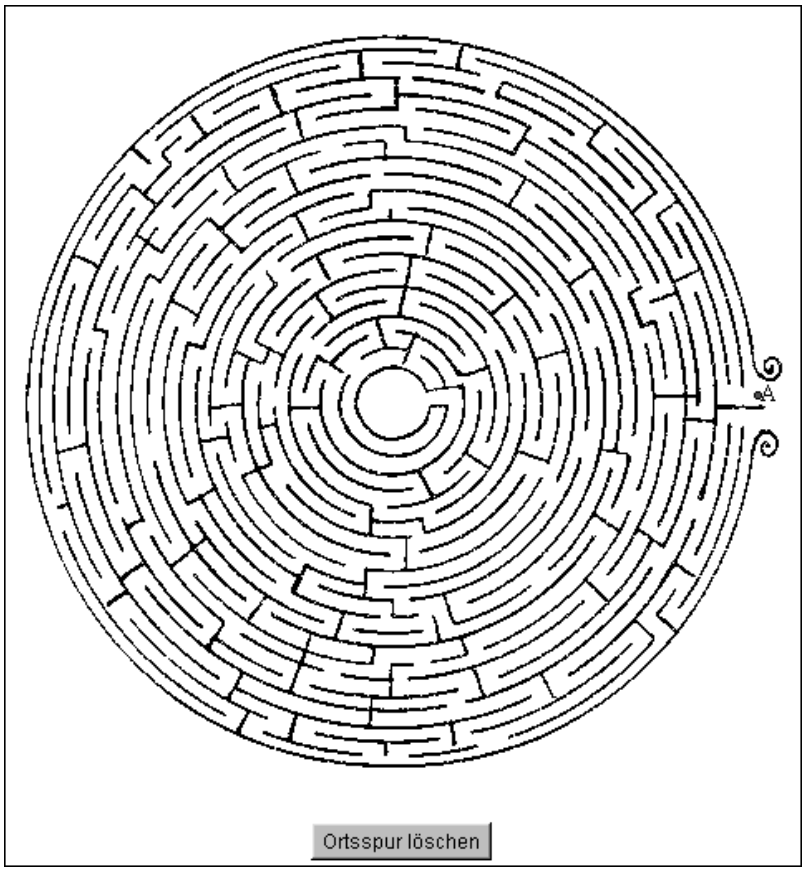

Abbildung 4.150: Zweites Beispiel für eine Labyrinth-Figur.

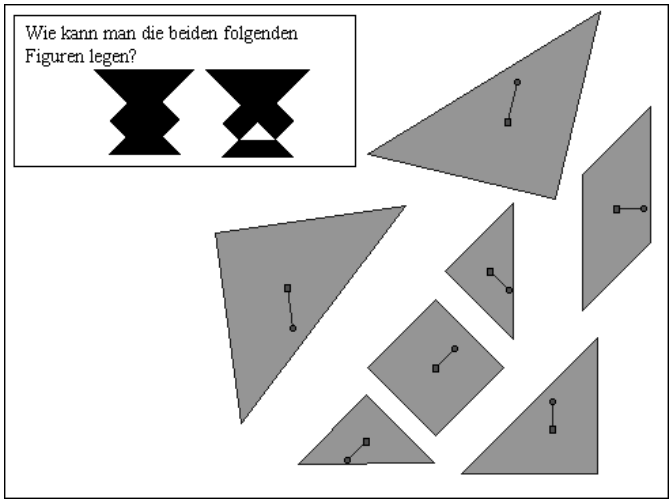

Abbildung 4.151: Wie kann man die beiden Vasen mit jeweils allen sieben Teilen legen?

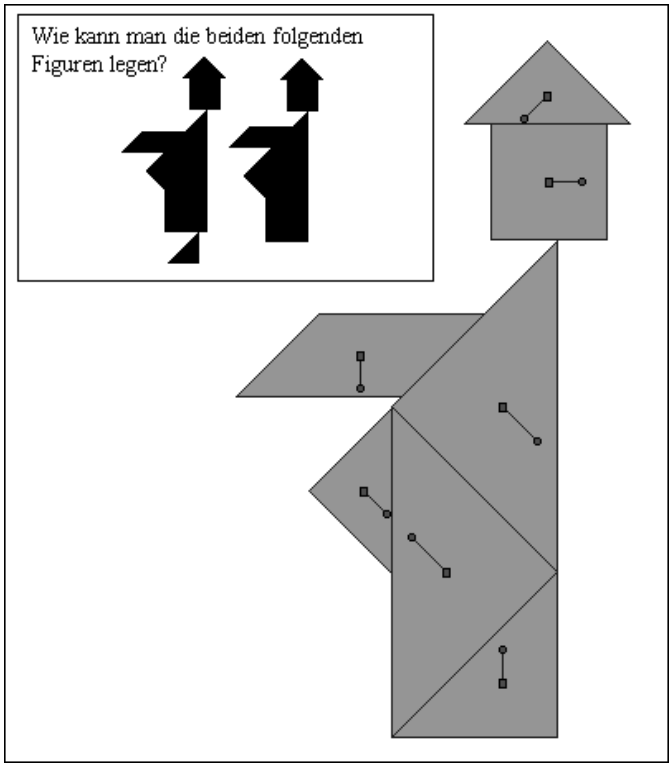

Abbildung 4.152: Wie lassen sich die beiden Männchen mit jeweils allen sieben Teilen legen?

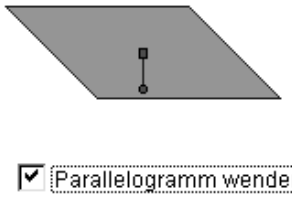

Abbildung 4.153: Erste Parallelogrammlage.

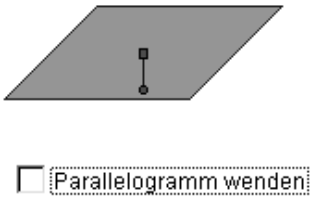

Abbildung 4.154: Zweite Parallelogrammlage.

Schachprobleme

Als Problem der acht Damen bezeichnet man die Aufgabe, acht Damen so auf einem Schachbrett zu positionieren, daß sie sich nicht gegenseitig bedrohen. Der in den Abbildungen 4.155-4.158 dargestellte Lernbaustein setzt diese Aufgabe durch eine interaktive Figur um.

Er zeigt einmal, wie grafische Objekte (Bilder) an ziehbare Punkte gebunden werden können, so daß der Lernbaustein überhaupt nicht mehr als geometrische Figur erscheint. Außerdem liefert ein Beispiel dafür, daß Figuren auch mit relativ komplexen Analysealgorithmen angereichert werden können.

Das Schachbrett und die acht Damen werden in der Zeichenfläche durch Bilddateien dargestellt. Während das Schachbrett dauerhaft in der Mitte der Zeichenfläche angezeigt wird, sind die restlichen Bilder an die Position von ziehbaren Punktobjekten gebunden. Wird eines dieser Punktobjekte verschoben, bewegt sich das Bild entsprechend mit. Für den Schüler entsteht dadurch der Eindruck, als würde er die Damen direkt verschieben.

Um die Brettstellung zu analysieren, wird mit *GeoScript* eine externe Java-Klasse aufgerufen, an die acht Punktobjekte als Parameter übergeben werden (Seite 401). Mit Hilfe von sogenannten "Backtracking-Algorithmen" wird die Brettstellung analysiert und eine gültige Lösung berechnet.[48] Der Schüler hat die Möglichkeit, sich das Ergebnis der Figurenanalyse anzeigen zu lassen. Neben der Bewertung der aktuellen Damenkonstellation kann er sich außerdem ein geeignetes Feld für die nächste Dame vorschlagen lassen (Abbildungen 4.156-4.158).

[48] Wirth 1975

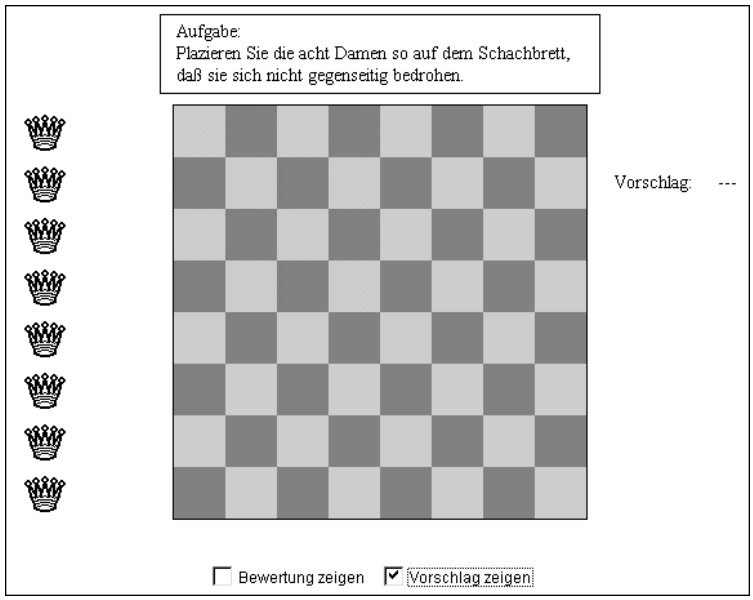

Abbildung 4.155: Die acht Damen sollen so auf dem Brett plaziert werden, daß sie sich nicht gegenseitig bedrohen.

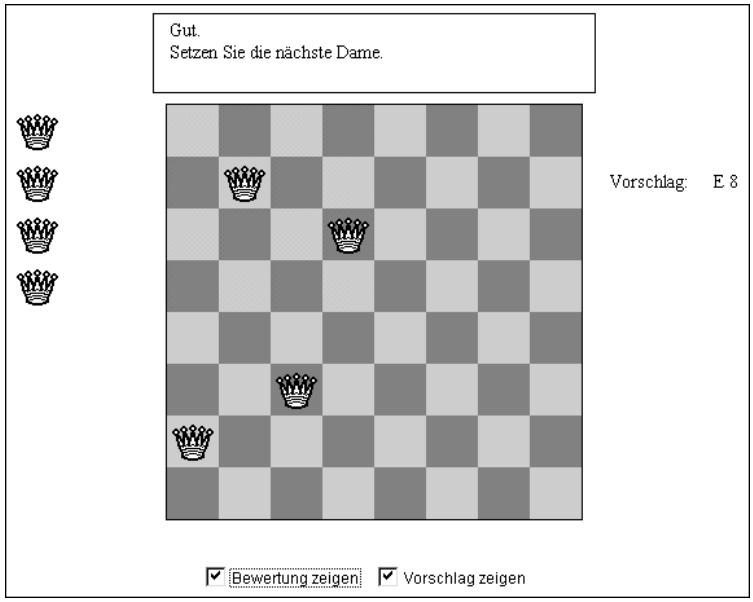

Abbildung 4.156: Die Figurenanalyse bewertet die Brettstellung und schlägt ein Feld für die nächste Dame vor.

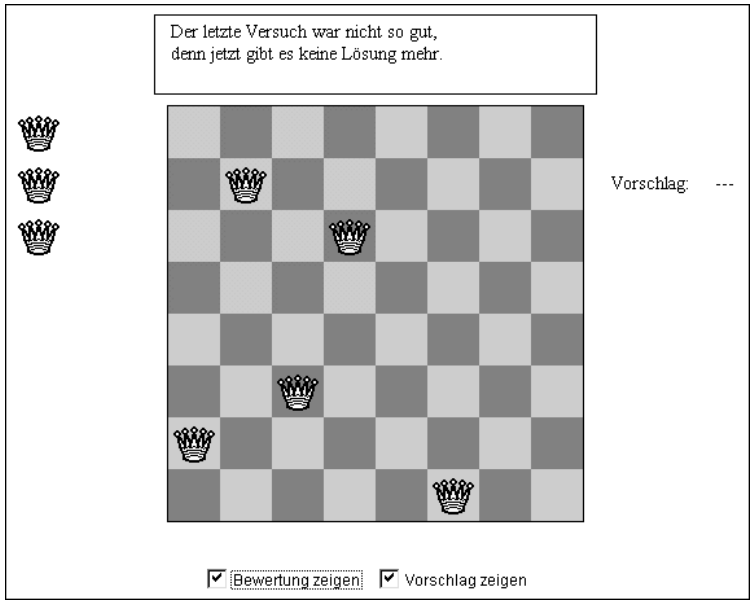

Abbildung 4.157: Die Figurenanalyse erkennt, ob die Aufgabe noch gelöst werden kann.

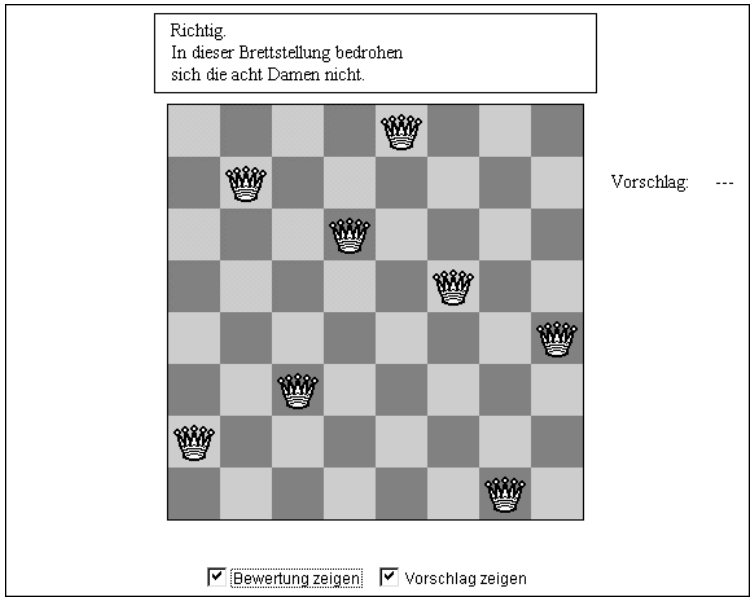

Abbildung 4.158: Eine mögliche Lösung des Problems der acht Damen.

Kapitel 5

Fallstudie: Ein Online-Skript zur Elementargeometrie

Auf der Basis der in den ersten vier Kapiteln dieser Arbeit dargestellten Überlegungen wurde ein Online-Skript zur Elementargeometrie erstellt und zum Ausgangspunkt einer formativen Evaluation gemacht. Im folgenden will ich einen kurzen Überblick über die Zielsetzung (Abschnitt 5.1) und die didaktische Konzeption (Abschnitt 5.2) geben. Darin wird das Augenmerk auf die Themen des Online-Skripts und die methodische Realisation gelegt. In den Abschnitten 5.3 und 5.4 wird beschrieben, wie die Evaluation durchgeführt worden ist und welche Ergebnisse festzustellen sind.

5.1 Zielsetzung und Zielgruppe

Als praktische Anwendung des Lernbaustein-Konzepts ist ein Online-Skript zur Elementargeometrie entwickelt worden. Darin wurden ausgewählte Kapitel des von Prof. Dr. Wellstein (1999) ausgearbeiteten Vorlesungstexts "Elementargeometrie" in untereinander vernetzte HTML-Dokumente umgesetzt und mit interaktiven Lernbausteinen angereichert.

Das Online-Skript ist ein Modul des Projekts "ZERO – Mathematik online" am Institut für Mathematik und ihre Didaktik der Universität Flensburg. In diesem Projekt wurden Lernmaterialien zum Lehramtsstudium Mathematik erstellt und im Internet veröffentlicht.[1] Das Projekt steht unter der Leitung von Prof. Dr. Schreiber und Prof. Dr. Wellstein. Es trägt die Arbeitsbezeichnung ZERO. Neben dem Thema Elementargeometrie sind dort weitere Materialien (etwa zur Arithmetik und Algebra, Kombinatorik, Problemlösen/Heuristik, u. a. m.) abrufbar. Ein interaktiver Aufgabentrainer steht zur Verfügung, um die in Vorlesungen und Übungen erworbenen Kenntnisse aufzufrischen, zu festigen oder zu überprüfen.

[1] Die Web-Seiten zum Projekt "Mathematik online" sind unter der Adresse http://www.uni-flensburg.de/mathe/zero/zero.html abrufbar.

Die Zielgruppe, für die das Online-Skript in erster Linie konzipiert ist, sind Mathematik-Studierende der Universität Flensburg. Für sie ist die Teilnahme an der Vorlesung Elementargeometrie obligatorisch. Die didaktische Intention des Online-Skripts bestand darin, einen interaktiven Lehrtext zu entwickeln, mit dem die Studierenden den Vorlesungsinhalt vor- oder nachbereiten können.

Weil die Distribution über das Internet einen Zugriff unabhängig vom Standort eines Rechners und vom verwendeten Betriebssystem ermöglicht, können auch Schüler und Lehrer an Schulen oder Studierende von anderen Universitäten das Online-Skript aufrufen und als Medium zum Nachschlagen, Lehren und Lernen einsetzen.

5.2 Didaktische Konzeption

Im folgenden möchte ich die didaktische Konzeption des Online-Skripts beschreiben. Dazu ist in Abschnitt 5.2.1 die Gliederung und der Aufbau der inhaltlichen Themen beschrieben. Anschließend wird in Abschnitt 5.2.2 dargelegt, wie der Lerninhalt methodisch in Web-Seiten umgesetzt worden ist und welche besonderen globalen Funktionen verfügbar sind.[2]

5.2.1 Die Themen des Online-Skripts

Das Online-Skript ist unterteilt in fünf Themen mit den Inhalten: Schwerpunkte, der Satz von Varignon, der Satz von Ceva, Winkelhalbierenden-Vierecke und Gelenkvierecke. Jedes dieser Themen setzt sich dabei aus den folgenden vier Komponenten zusammen:

1. **Lehrtext** Der Lehrtext zu einem Thema entspricht inhaltlich einem Kapitel des Vorlesungsskripts, das als Vorlage für die Umsetzung diente. Jeder Lehrtext ist in 3-10 Abschnitte und Unterabschnitte gegliedert.

2. **Aufgabensammlung** Im Anschluß an einen Lehrtext befindet sich jeweils eine Sammlung mit 5-10 Übungsaufgaben pro Thema. Zu mehr als der Hälfte der Aufgaben ist ein Lernbaustein vorhanden, der die Aufgabe durch eine passende Figur darbietet und durch den der Lösungsprozeß unterstützt werden soll. Alle Aufgaben ohne Figur sind schriftlich zu lösen.

3. **Literaturhinweise** Eine Liste mit themenspezifischen Literaturhinweisen ist im Anschluß an jede Aufgabensammlung verfügbar. Zusätzlich sind darin auch Verweise auf Web-Seiten mit passenden Inhalten aufgenommen.

4. **Themenspezifische Evaluation** Zu jedem Thema gibt es ein spezielles Evaluationsformular, durch das der Schüler themenspezifische Aspekte bewerten kann. In der in Abschnitt 5.3 beschriebenen Untersuchung wurden unter anderem diese Evaluationsformulare eingesetzt.

[2] Die folgenden Ausführungen beziehen sich auf den Stand von Januar 2000. Im Rahmen des übergeordneten Projekts "Mathematik online" wurden inzwischen Änderungen an der Navigation und der gestalterischen Darstellung vorgenommen, um ein einheitliches Layout aller Module zu gewährleisten.

5.2.2 Methodische Umsetzung

Nachdem die Themen des Online-Skripts und ihr Aufbau skizziert worden sind, sollen jetzt besondere Aspekte bei der methodischen Umsetzung hervorgehoben werden.

Den Textinhalt der Vorlage habe ich weitgehend unverändert übernommen, um einen systematischen und logischen Zugang zu den Inhalten zu gewährleisten. Ich habe jedoch versucht, die lineare Textgliederung etwas aufzulockern und alternative Zugriffsmöglichkeiten auf den Lehrtext zu eröffnen. Das Online-Skript kann dadurch in verschiedenen didaktischen Kontexten eingesetzt werden: als linearer und systematischer Kurs, aber auch als Nachschlagewerk oder zum Visualisieren und Demonstrieren von Sätzen durch gezieltes Aufrufen einzelner Lernbausteine. Insgesamt wurde Wert darauf gelegt, die besonderen Möglichkeiten des Mediums Internet zu nutzen.

Im folgenden möchte ich in aufzählender Form die wichtigsten Unterschiede im Vergleich mit einem gedruckten Vorlesungsskript darlegen und neue globale Funktionen beschreiben:

Textaufbau und Gliederung

Bei der Umsetzung des Lehrtexts in HTML-Seiten – und damit in eine Hypertextstruktur – habe ich den Inhalt aller Themen in mehrere Abschnitte untergliedert, damit der Lehrtext auch auf dem Bildschirm übersichtlich bleibt. Wichtig war mir dabei, daß der Lehrtext nicht zu sehr zergliedert wird. Sinneinheiten habe ich jeweils auf 1-3 Web-Seiten zusammengefaßt. In den Lehrtext sind an geeigneten Stellen sog. Hyperlinks (Verweise) eingefügt, die auf andere Textstellen in dem Online-Skript verweisen. Aus dem linearen Textfluß des gedruckten Vorlesungsskripts entstand auf diese Weise eine in sich vernetzte Hypertextstruktur.

Lernbausteine

Ein wesentlicher Aspekt bei der methodischen Umsetzung war das Ersetzen eines großen Teils der statischen Abbildungen durch interaktive Lernbausteine. Diese nehmen drei wichtige Funktionen ein:

1. Jeder geometrische Satz wird durch eine bewegliche Figur visualisiert. Im Unterschied zu einer statischen Abbildung in einem gedruckten Vorlesungsskript kann der Schüler sich dadurch interaktiv mit dem geometrischen Satz auseinandersetzen. Er kann ihn an beliebig vielen Figurenzuständen empirisch überprüfen. Außerdem werden durch einige Lernbausteine auch Beweise in mehreren Schritten demonstriert. Beispiele dafür sind etwa die Figuren zum Satz von Ceva (Figur 2.1 und 2.2), zum Satz von Varignon (Figur 1.1), zum Wittenbauer-Parallelogramm (Figur 4.4 und 4.5), zum Innenwinkelhalbierenden-Viereck (Figur 1.1) und zum Ecken- und Kantenschwerpunkt im Dreieck (Figur 3.1 und 3.2).

2. In den Lehrtext sind Lernbausteine zur Selbstkontrolle mit Antwortanalyse eingebettet. Dadurch hat der Schüler die Möglichkeit, sein Wissen über bestimmte Begriffe und Sätze selbsttätig zu überprüfen. Eventuell wird er durch die Bewertung auf eigene Wissensdefizite aufmerksam gemacht

und arbeitet daraufhin die entsprechenden Abschnitte noch einmal gezielt durch. Beispiele sind in den folgenden Themen zu finden: Schwerpunkte, Satz von Ceva und Winkelhalbierenden-Vierecke.

3. Zu über der Hälfte der ca. 40 Übungsaufgaben sind Lernbausteine vorbereitet. Diese veranschaulichen die Aufgabenstellung durch eine Figur. Inhaltlich geht es bei den Aufgaben vor allem darum, Sätze und Beweise zu finden, wobei der Schüler die Möglichkeit hat, die zugehörigen Figuren zu variieren und dabei heuristische Strategien zur Satz- und Beweisfindung anzuwenden (Abschnitt 4.2). Beispiele finden sich in den Aufgabensammlungen im Anschluß an die Lehrtexte.

Von den rund 50 Lernbausteinen sollen an dieser Stelle einige genannt werden, bei denen besondere Funktionen von *Geometria* zum Einsatz kommen, die mit anderen DG-Systemen nicht möglich sind.

- **Satz von Ceva** In den Figuren 2.1 und 2.2 werden Bilddateien eingebunden, um die im Satz verwendeten Formeln typographisch korrekt darzustellen. Die Aufgabe zur Selbstkontrolle (2.1) verwendet eine externe Java-Klasse, durch die das Teilverhältnis dreier Punkte berechnet wird.

  In der Figur 2.3 zum Beweis des Satzes von Ceva ist der Definitionsbereich von dem Punktobjekt P so eingeschränkt, daß P nur im Winkelraum BAC ziehbar ist.

  In dem Lehrtext sind in den Abschnitten 1.3 und 2.1 zwei Aufgaben zum Teilverhältnis und zum Ceva-Satz mit Antwortanalyse vorhanden. Bei der Bewertung werden auch nur teilweise richtige (bzw. falsche) Antworten erkannt und kommentiert.

- **Schwerpunkte** In den Figuren 3.1, 3.2 und 4.1 werden abhängig vom Figurenzustand Teilfiguren ein- und ausgeblendet. Die einzelnen Phasen der Schwerpunktbestimmung werden durch Textinformationen kommentiert und die zunehmende Masse des Schwerpunkts wird durch unterschiedliche Punktgrößen angedeutet.

  In der Figur zur Aufgabe 5 ist der Definitionsbereich von dem ziehbaren Punktobjekt A' so eingeschränkt, daß A' entlang der Strecke AB gezogen werden, jedoch nicht mit A zusammenfallen kann.

  In den Lehrtext sind in den Abschnitten 2 und 3 jeweils eine Variationsaufgabe zum Eckenschwerpunkt eingebettet. Bei der Antwortanalyse werden auch Antworten erkannt und kommentiert, die teilweise richtig sind.

- **Satz von Varignon** In der Figur 3.1 können durch Betätigen eines Schalters abwechselnd zwei spezielle Teilfiguren farblich hervorgehoben werden.

  In der Figur zur Aufgabe 3 wird eine Viereckskanalyse eingesetzt (Abschnitt 3.3.8), die den Viereckstyp des Varignon-Parallelogramms bestimmt.

- **Winkelhalbierenden-Vierecke** In den Figuren 1.1 und 2.2 werden griechische Buchstaben zur Winkelbezeichnung verwendet.

  In der Figur 1.1 wird der Inkreis des Vierecks genau dann eingeblendet, wenn es ein Sehnenviereck ist. Zu diesem Figurenzustand wird eine entsprechende Textinformation angezeigt.

In der Figur 1.2 ist der Zustandsraum so begrenzt, daß das Viereck $ABCD$ stets ein Tangentenviereck bleibt.

In den Abschnitten 1.1 und 2 gibt es zwei Variationsaufgaben zum Experimentieren, in denen spezielle Vierecke zu erstellen sind. Die Antwortanalyse erkennt die wichtigsten Viereckstypen und kommentiert die Schülerlösungen entsprechend.

- **Gelenkvierecke** In den Figuren 1.1 und 4.1-4.3 können jeweils beide Arme des Gelenkvierecks bewegt werden.

  In der Figur 2.1 zum Satz von Grashof ist der Zustandsraum so eingeschränkt, daß für die Längen der Stäbe a, b, c, d stets gilt: $a < b \leq c \leq d$. Außerdem zeigt die Figur eine Textinformation an, wenn die Grashofsche Bedingung erfüllt ist. Dadurch wird der Schüler darauf hingewiesen, daß der Arm a des Gelenkvierecks nicht voll drehbar ist. Im Unterschied zu entsprechenden Figuren in anderen DG-Systemen, ist in diesem Fall der Arm auch tatsächlich nicht voll drehbar.

Mehrfenstertechnik

Der eigentliche Lehrtext der Vorlage orientiert sich am Stil der klassischen Stoff-Exposition. Diese Struktur wurde beibehalten. Um den Lehrtext aber etwas weniger schematisch präsentieren zu können, wurde in den meisten Fällen der Beweis zu einem Satz in einem separaten Fenster untergebracht, das durch einen Hypertextverweis aufgerufen werden kann. Diese Form der Mehrfenstertechnik ermöglicht es, durch passende Fensteranordnung Satz, Figur und Beweis parallel zu betrachten (Abbildung 5.1).

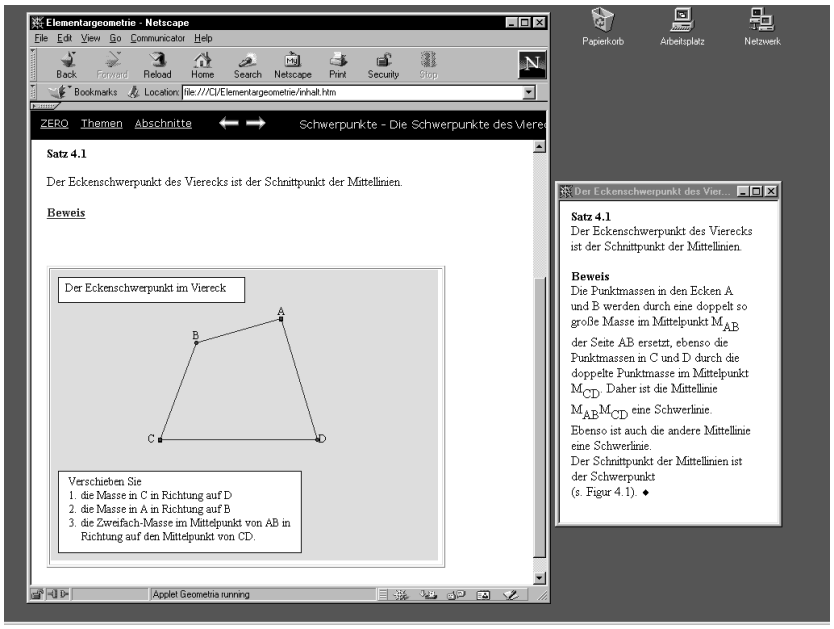

Abbildung 5.1: Mehrfenstertechnik.

Das Entscheidende ist jedoch die reduzierte Sicht, die durch das Ausblenden der Beweise erreicht wird. Der Inhalt einer Seite kann dadurch sehr übersichtlich dargestellt werden. Beim ersten Lesen kann man den Lehrtext schnell überblicken. Gleichzeitig hat man aber auch stets die Möglichkeit, sich einen ausführlichen Beweis zum jeweiligen Satz anzeigen zu lassen.

Neben den Beweisen sind auch die Lernbausteine zur Selbstkontrolle in separaten Fenstern dargestellt. Außerdem werden alle Verweise auf externe Web-Seiten in neu geöffneten Browserfenstern angezeigt.

Navigation

In dem Online-Skript gibt es drei alternative Funktionen zur Navigation: die Navigationsleiste, der Index und die Galerie. Im folgenden sollen diese Navigationshilfen beschrieben werden.

1. Die Navigationsleiste verläuft horizontal am oberen Bildschirmrand und ist das Hauptwerkzeug zum Navigieren in dem Online-Skript (Abbildung 5.2). Sie enthält fünf Verweisfelder, die die folgende Bedeutung haben (von links nach rechts):

   - **ZERO** Durch Anwählen von ZERO gelangt man auf eine Web-Seite, die auf weitere Materialien im Projekt "Mathematik online" verweist.
   - **Themen** Unter der Bezeichnung Themen befindet sich ein Verweis auf die Hauptseite mit der Themenübersicht (Abbildung 5.2). Durch Anwählen einer Themenüberschrift bekommt man die Untergliederung des Themas angezeigt (Abbildung 5.3). Von dort aus kann man direkt in die einzelnen Abschnitte des Lehrtexts, zu der Aufgabensammlung, zu den Literaturhinweisen oder zu der themenspezifischen Evaluation verzweigen.
   - **Abschnitte** Das Feld Abschnitte verweist auf die Gliederung des aktuellen Themas. Der Verweis ist nur ausführbar, wenn eins der fünf Themen zuvor angewählt wurde.
   - **Vorwärts- und Rückwärtspfeil** Mit Hilfe dieser Pfeile kann in dem Lehrtext abschnittsweise vor- und zurückgeblättert werden. Der Schüler wird dadurch im Sinne einer "Guided Tour" auf einem linearen Weg durch das Online-Skript geführt. Die "Guided Tour" folgt dem Textfluß des gedruckten Vorlesungsskripts.
   - **Anzeigefeld** Das Anzeigefeld enthält einen kurzen Bezeichner, der den aktuellen Seiteninhalt beschreibt. Es soll dem Schüler dazu dienen, sich leichter zu orientieren. In Abbildung 5.2 zeigt das Anzeigefeld die Themenübersicht an.

2. Der Index umfaßt eine alphabetische Liste der wichtigsten Begriffe, die in den fünf Lehrtexten verwendet werden. Dabei verweist jeder Begriff auf die entsprechende Stelle in dem Lehrtext. Der Schüler kann dem Hypertextverweis folgen und sich die entsprechende Textstelle direkt anzeigen lassen. Durch einen solchen Index läßt sich das Online-Skript als ein interaktives Nachschlagewerk verwenden (Abbildung 5.4).

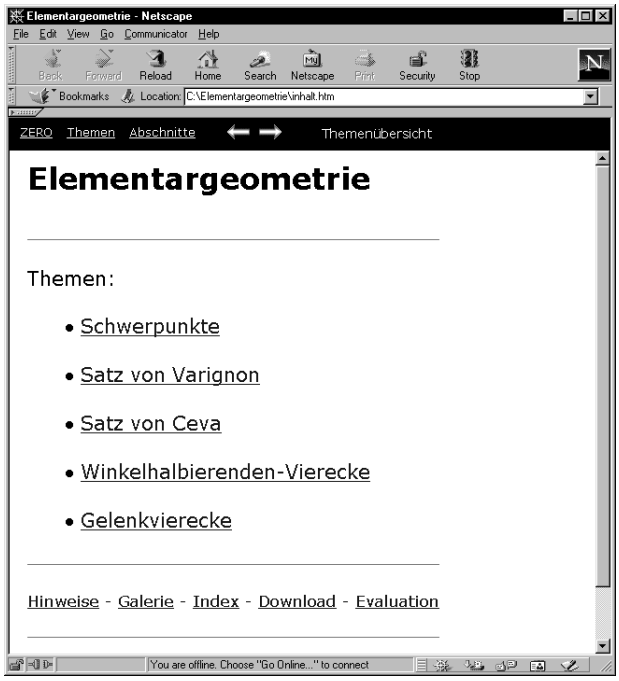

Abbildung 5.2: Themenübersicht.

Abbildung 5.3: Abschnittsübersicht vom Thema Satz von Ceva.

Abbildung 5.4: Index.

3. In der Galerie können die im Lehrtext und in den Aufgabensammlungen verwendeten Lernbausteine direkt aufgerufen werden. Dazu ist eine Auswahlliste vorhanden, aus der der Schüler einen Lernbaustein seiner Wahl gezielt aufrufen kann. Der Lernbaustein wird dann separat ohne umfließenden Lehrtext angezeigt. Mit dieser Funktion kann man sich einen ersten Überblick über die behandelten Sätze verschaffen. Außerdem läßt sich dadurch längeres Suchen im Lehrtext nach bestimmten Lernbausteinen umgehen (Abbildung 5.5).

Download

Unter der Überschrift Download wird die Möglichkeit geboten, das Online-Skript in gepackter und komprimierter Form vom Server der Universität Flensburg herunterzuladen, um es lokal auf einem Rechner zu installieren. Auf diese Weise können lange Ladezeiten vermieden werden, und die Geschwindigkeit beim Arbeiten erhöht sich.

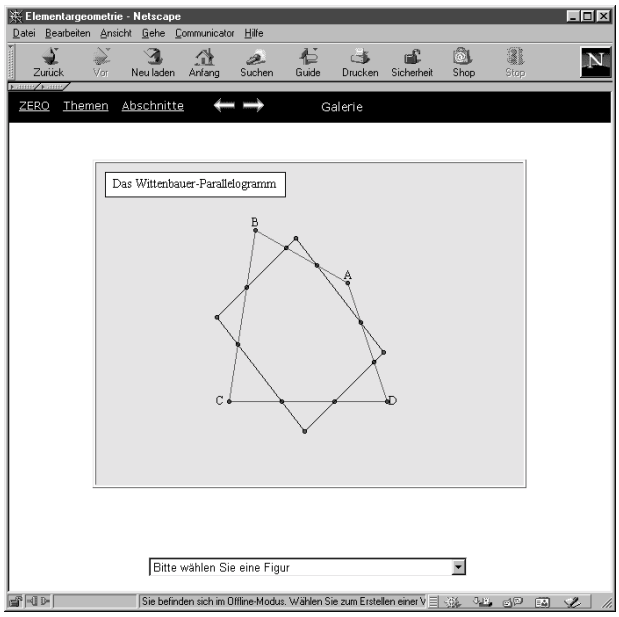

Abbildung 5.5: Galerie.

5.3 Durchführung der Evaluation

Das im vorangegangenen Abschnitt beschriebene Online-Skript wurde im Sommersemester 1999 durch Studierende der Pädagogischen Hochschulen in Weingarten und Schwäbisch Gmünd evaluiert. Das allgemeine Ziel der Evaluation war es, den praktischen Einsatz zu erproben. Die Fragestellungen, die dabei im Mittelpunkt standen, sind in Abschnitt 5.3.1 genannt. Anschließend werden die methodische Vorgehensweise und die verwendeten Testinstrumente (Abschnitt 5.3.2) sowie die Stichprobe der Evaluation (Abschnitt 5.3.3) erläutert. Die Ergebnisse der Untersuchung sind in Abschnitt 5.4 zusammenfassend dargestellt.

5.3.1 Ziele und Fragestellungen

Die Evaluation hatte zum Ziel, das im Rahmen dieser Arbeit entwickelte Online-Skript in einem Feldtest[3] zu erproben. Dabei standen die folgenden Fragestellungen im Zentrum der Untersuchung:

- Wie beurteilen die Probanden den Aufbau und die Darstellung?
- Wie bewerten die Probanden den Umgang mit den Figuren?
- Wie schätzen die Probanden die Unterstützung des Lernprozesses durch das Online-Skript ein?
- Wie stufen die Probanden die Wartezeit beim Laden der Web-Seiten und Initialisieren der Figuren ein?

[3] Schreiber 1998, S. 87

- Gibt es bei den genannten Antworten Unterschiede hinsichtlich der bearbeiteten fünf Themen?
- Gibt es bei den genannten Antworten Unterschiede hinsichtlich soziographischer Merkmale (Geschlecht, Alter, Beruf)?
- Welche Programmfehler treten auf und welche Verbesserungen schlagen die Probanden vor?

5.3.2 Methodisches Vorgehen

Im folgenden zeige ich, welche Formen von Evaluationsformularen eingesetzt wurden, welche Skalenarten darin verwendet sind und welche Items den einzelnen Untersuchungszielen zugeordnet sind.

Evaluationsformulare Zur Durchführung der Evaluation wurden insgesamt sechs verschiedene Evaluationsformulare eingesetzt. Zu jedem der fünf Themen in dem Online-Skript gab es eine Web-Seite mit einem themenspezifischen Evaluationsformular (Abschnitt 5.2.1). Diese waren identisch aufgebaut, bezogen sich aber auf das jeweils vom Probanden bearbeitete Thema. Daneben gab es ein Evaluationsformular, das sich auf allgemeine Aspekte bezog. Alle sechs Evaluationsformulare sind in dem Online-Skript aufrufbar und können bei bestehender Internet-Verbindung per E-Mail verschickt werden. Parallel wurden – aus technischen Gründen (Abschnitt 5.3.3) – im Verlauf der Untersuchung aber auch Evaluationsformulare in Form von gedruckten Fragebögen eingesetzt, wobei Aufbau und Inhalt nahezu identisch sind.[4] Es wurden lediglich die allgemeinen und die themenspezifischen Fragen zusammengefaßt und eine Frage nach dem bearbeiteten Thema hinzugefügt.

Skalenarten In den Evaluationsformularen werden insgesamt drei Formen der Datenerfassung eingesetzt:

1. Um die Items des Fragebogens zu messen, wurde eine sechsstufige Rangskala verwendet. Der Proband sollte durch Auswahl eines Werts der Rangskala den Grad seiner Zustimmung zu einer Aussage ausdrücken. Dabei bedeutet: 0 = "überhaupt nicht", 1 = "mit starken Einschränkungen", 2 = "mit Einschränkungen", 3 = "im wesentlichen", 4 = "fast vollständig" und 5 = "vollständig". Eine sechsstufige Skala wurde aus zwei Gründen gewählt. Zum einen sollte eine geradzahlige Skala eingesetzt werden, da sie den Probanden zwingt, sich tendenziell für einen der beiden Skalenpole zu entscheiden. Zum anderen würde eine vierstufige Skala zu wenig und eine achtstufige Skala zu stark differenzieren.
2. Die Kenndaten einer Person (Geschlecht, Alter und Beruf) wurden jeweils mittels einer Nominalskala gemessen. Dasselbe gilt für das vom Probanden vorwiegend bearbeitete Thema.
3. Für die Angabe von technischen Fehlern und Verbesserungsvorschlägen wurden keine Skalen verwendet. Stattdessen stand in dem Evaluationsformular ein Eingabefeld zur Verfügung. Auf den gedruckten Fragebögen waren einige Zeilen für eine schriftliche Beschreibung vorgesehen.

[4] Die Evaluationsformulare sind in Anhang E (Seite 407) abgedruckt.

Zuordnung der Items zu den Untersuchungszielen Im folgenden werden den oben genannten Untersuchungszielen die in den Evaluationsformularen verwendeten Items zugeordnet. Mit jedem Item sollte eine Variable gemessen werden. Die genaue inhaltliche Bedeutung der Variablen gebe ich bei der Darstellung der Ergebnisse (Abschnitt 5.4). An dieser Stelle werden vorerst nur die Variablenbezeichner und Fragebogen-Items genannt.

Aufbau und Darstellung Um zu erfahren, wie die Probanden den Aufbau und die Darstellung einschätzen, wurde nach den drei Variablen Handhabung, Gliederung und Sprachprägnanz gefragt. Die Probanden sollten dazu den Grad ihrer Zustimmung zu den folgenden Items auf einer sechsstufigen Rangskala angeben: *Die Handhabung des Online-Skripts fiel mir leicht. Die Gliederung des Online-Skripts erscheint mir sinnvoll. Die sprachliche Darstellung erscheint mir prägnant.*

Umgang mit den verwendeten Lernbausteinen Um von den Probanden Antworten auf die Frage nach dem Umgang mit den Lernbausteinen (den Figuren) zu erhalten, wurde nach den Variablen Übersichtlichkeit der Figuren, Erwartungskonformität des Figurenverhaltens und nach der Figureninteraktion gefragt. Die Probanden sollten jeweils den Grad ihrer Zustimmung zu den folgenden Aussagen angeben: *Die Figuren waren übersichtlich. Die Figuren funktionierten, wie ich es erwartete. Die Interaktion mit den Figuren fiel mir leicht.*

Unterstützung des Lernprozesses Um zu erfahren, wie die Probanden die Unterstützung des Lernprozesses durch das Online-Skript einschätzen, wurde nach den Variablen Bearbeitungsfreude, Verständniserleichterung und Schwierigkeitsgrad der Aufgaben gefragt. Die Probanden sollten jeweils den Grad ihrer Zustimmung zu den folgenden Items angeben: *Die interaktiven Figuren haben mir das Verständnis erleichtert. Ich habe gerne mit dem Online-Skript gearbeitet. Ich konnte die Aufgaben leicht lösen.*

Wartezeit beim Laden und Initialisieren Die Probanden sollten die Wartezeit beim Laden der Web-Seiten und Initialisieren der Figuren beurteilen, indem sie jeweils den Grad ihrer Zustimmung zu der folgenden Aussage angeben: *Die Wartezeit beim Laden und Initialisieren war akzeptabel.*

Differenzierung zwischen den bearbeiteten Themen Um Unterschiede zwischen den bearbeiteten fünf Themen hinsichtlich aller gemessenen Variablen feststellen zu können, wurde jeweils das bearbeitete Thema mit erfaßt. Bei den themenspezifischen Evaluationsformularen war das Thema bereits durch das Formular festgelegt. Bei den gedruckten Fragebögen war dazu eine entsprechende Angabe zu tätigen (*Ich habe vorwiegend mit dem folgenden Thema gearbeitet: ...*).

Differenzierung nach soziographischen Merkmalen Um die Bedeutung soziographischer Merkmale beim Arbeiten mit dem Online-Skript feststellen zu können, wurden Alter, Geschlecht und Beruf der Probanden erfaßt.

Technische Fehler und Verbesserungsvorschläge Nach dem gezielten Abfragen von speziellen Items wurden die Probenden abschließend aufgefordert, anzugeben, ob und wo Unzulänglichkeiten aufgetreten sind: *Was hat Ihnen insgesamt nicht so gut gefallen? Haben Sie Verbesserungsvorschläge? Haben Sie Programmfehler festgestellt?* Um eine Antwort zu formulieren, ist ein entsprechender Freiraum in den Evaluationsformularen vorgesehen.

5.3.3 Die Stichprobe

Die Evaluation hat im Sommersemester 1999 an der PH Weingarten und an der PH Schwäbisch Gmünd stattgefunden. An der PH Weingarten wurde sie von Prof. Dr. Schumann mit Lehramtsstudierenden im Fach Mathematik durchgeführt. Mitte Juni 1999 erhielten in zwei Seminarveranstaltungen die Studierenden eine kurze Einführung und bearbeiteten jeweils ein Thema ihrer Wahl. Anschließend sollten sie ein themenspezifisches und ein allgemeines Evaluationsformular ausfüllen und per E-Mail absenden. Insgesamt kamen in den beiden Veranstaltungen 14 Evaluationsformulare zusammen. An einem dritten Termin sollte eine weitere Evaluation mit einer Gruppe von über 30 Studierenden stattfinden. Dieser Versuch mußte jedoch abgebrochen werden, da ein simultaner Zugriff auf das Online-Skript mit 21 Rechnern nicht möglich war. Die Übertragungsgeschwindigkeit über das Internet war zu langsam. Folglich konnten auch keine Evaluationsformulare per E-Mail verschickt werden. Als Konsequenz daraus wurde beschlossen, die Studierenden als Hausaufgabe ein Thema bearbeiten zu lassen. Da sie den Zeitpunkt selbst wählen konnten, war zu erwarten, daß es zu keinem Datenstau kommen würde. In den folgenden Sitzungen wurden an über 50 Studierende Fragebögen in gedruckter Form ausgegeben, so daß das Online-Skript auch in der Offline-Version bearbeitet und evaluiert werden konnte. Bis zum Ende des Semesters gab es einen Rücklauf von 12 Fragebögen, so daß von der PH Weingarten mit insgesamt 26 Datensätzen gerechnet werden konnte.

Im Anschluß an die Untersuchung in Weingarten wurde von Dr. Hole von der PH Schwäbisch Gmünd eine zweite Evaluation durchgeführt. Dazu war das Online-Skript lokal auf den Rechnern des dortigen Rechenzentrums installiert. Die Evaluation fand an zwei Terminen im Juni 1999 statt. Beim ersten Einsatz bearbeiteten 22 Lehramtsstudierende im Fach Mathematik eine Stunde lang das Thema Winkelhalbierenden-Vierecke und füllten abschließend schriftlich einen Fragebogen aus. In der zweiten Sitzung bearbeiteten 9 Lehramtsstudierende wiederum das Thema Winkelhalbierenden-Vierecke. Allerdings standen hier nur 45 Minuten zur Verfügung. Beide Veranstaltungen wurden durch Dr. Hole geleitet.

Zusammen mit 7 Evaluationsformularen, die von Internet-Nutzern anonym abgesandt wurden, lagen insgesamt 64 Datensätze zur Auswertung vor. Davon waren allerdings nur 40 vollständig und lückenlos ausgefüllt worden. Bei den restlichen 24 Evaluationsformularen wurden jeweils einzelne Items – vermutlich aus Flüchtigkeit – nicht beantwortet.[5]

[5] Aufgrund dieser relativ geringen Zahl lückenlos ausgefüllter Fragebögen wurden auch die unvollständigen Datensätze, soweit wie möglich, mit in die Auswertung einbezogen. Aus diesem Grund wird im folgenden die Anzahl n stets geringfügig variieren.

Geschlechterverhältnis

Bezüglich des Verhältnisses der Geschlechter zueinander konnten 59 Fragebögen ausgewertet werden. Der Anteil der weiblichen Probanden liegt mit 71 % deutlich höher als der Anteil der männlichen Teilnehmer (demnach 29 %).

Dieses Ergebnis ist jedoch nicht unerwartet. Es spiegelt das allgemeine Geschlechterverhältnis von Lehramtsstudierenden im Fach Mathematik an diesen Hochschulen wieder.

Altersstruktur

Bezüglich der Altersstruktur konnten 59 Fragebögen ausgewertet werden. Um die Auswertung zu vereinfachen, wurden von vornherein sechs Altersklassen gebildet. Der Anteil der 0 bis 20 Jahre alten Probanden der Stichprobe betrug $20,3$ %, der 21 bis 30jährigen $69,5$ %, der 31 bis 40jährigen $6,8$ %, der 41 bis 50jährigen $1,7$ % und der 51 bis 60jährigen ebenfalls $1,7$ %. Probanden, die älter als 60 Jahre waren, gab es nicht.

Beruf

Im Hinblick auf den Kennwert Beruf konnten 59 Evaluationsformulare ausgewertet werden. In Form von Nominaldaten standen auf den Fragebögen zur Auswahl: Schüler/in, Student/in (Lehramt), Student/in (andere Fachrichtungen), Lehrberuf und "nicht aufgeführt". Insgesamt waren allerdings $94,9$ % der Probanden Lehramtsstudierende und nur $1,7$ % Studierende anderer Fachrichtungen. $3,4$ % der Befragten waren in einem Lehrberuf tätig. Schülerinnen und Schüler sowie Teilnehmer aus anderen Berufsgruppen gab es nicht.

Bearbeitetes Thema

Bezüglich des vom Probanden bearbeiteten Themas konnten insgesamt 48 Fragebögen ausgewertet werden. Danach haben $12,5$ % der Probanden das Thema "Schwerpunkte", $2,1$ % das Thema "Satz von Varignon", $6,3$ % das Thema "Satz von Ceva" und $79,2$ % das Thema "Winkelhalbierenden-Vierecke" bearbeitet. Das Thema "Gelenkvierecke" wurde nicht gewählt.

5.4 Ergebnisse der Evaluation

Nachfolgend werden die Ergebnisse der Evaluation vorgestellt und vor dem Hintergrund der didaktischen Konzeption des Online-Skripts diskutiert.

Bei der Darstellung der Ergebnisse gehe ich in der Reihenfolge vor, die bei der Definition der Untersuchungsziele vorgegeben wurde. Zu allen Variablen werde ich für eine erste Übersicht eine kurze Beschreibung geben. Danach differenziere ich das Ergebnis nach Geschlecht und prüfe mit Hilfe des t-Tests[6] für

[6] Als statistisches Werkzeug wurde das von Kleiter (1988-1996) entwickelte Programmpaket *KMSS-6* verwendet. Um den t-Test für unabhängige Stichproben durchzuführen, wurde daraus das Programm *STAT-II* eingesetzt. Dieses prüft beim Berechnen des t-Tests stets, ob Varianz-Homogenität vorliegt. Ist das nicht der Fall, so wird beim t-Test anders vorgegangen. Das Programm setzt in diesem Fall automatisch eine approximative *df*-Korrektur nach Nie u. a. (1975, S. 270) ein.

unabhängige Stichproben, ob signifikante Abweichungen zwischen den Mittelwerten bestehen. Bei allen Mittelwertvergleichen wurde eine zweiseitige Fragestellung untersucht und ein Signifkanzniveau von $\alpha = 5\%$ festgelegt. Anschließend wird noch zwischen den beiden Themen Winkelhalbierenden-Vierecke und Schwerpunkte bezüglich der jeweiligen Variablen differenziert. Da die verbleibenden drei Themen von weniger als 10 % der Probanden gewählt worden sind, werden diese nicht für eine themenspezifische Differenzierung herangezogen. Auf eine Unterscheidung nach Alter und Beruf habe ich ebenfalls verzichtet, da im Hinblick auf diese beiden Kenndaten nahezu homogene Gruppen vorgefunden wurden.

5.4.1 Aufbau und Darstellung

Die Ergebnisse der subjektiven Einschätzung von Aufbau und Darstellung des Online-Skripts werden im folgenden detailliert dargestellt.

Handhabung

Durch die Variable Handhabung sollte gemessen werden, ob die Probanden das Online-Skript richtig gebrauchen und einfach bedienen können. Konkret ist damit das Verwenden der Navigationsleiste gemeint, um sich im Hypertext zurechtzufinden. Die Handhabung umfaßt aber auch das Bedienen der Mehrfenstertechnik und das Unterscheiden zwischen statischen Abbildungen und den eingebetteten, interaktiven Figuren. Die Abbildung 5.6 zeigt, inwieweit die Probanden der Aussage *Die Handhabung des Online-Skripts fiel mir leicht* zugestimmt haben. Bereits auf den ersten Blick läßt sich eine rechtssteile Verteilung erkennen

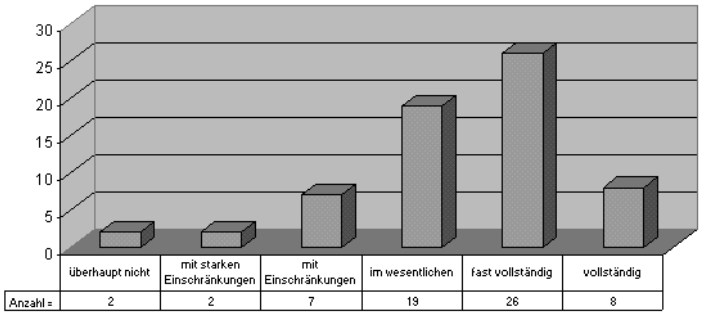

Abbildung 5.6: Deskriptive Auswertung der Variable Handhabung.

($n = 64$, $\bar{x} = 3,39$, $s = 1,14$). Werden die Antworten nach Geschlecht differenziert, so zeigen sich deutliche Unterschiede (Abbildung 5.7). Die geschlechtsspezifischen Unterschiede zwischen den Befragten im Hinblick auf die Handhabung sind signifikant (Tabelle 5.1). Bei der Differenzierung der Ergebnisse zur Handhabung zwischen den Themen Schwerpunkte und Winkelhalbierenden-Vierecke wird das Thema Schwerpunkte von den Probanden sehr unterschiedlich beurteilt (Abbildung 5.8). Von der einen Hälfte der Befragten wird die Handhabung als überhaupt nicht leicht oder als nur mit starken Einschränkungen leicht

empfunden. Die andere Hälfte hingegen beurteilt die Handhabung vollständig oder fast vollständig leicht. Dieses polarisierende Ergebnis muß allerdings vor dem Hintergrund gesehen werden, daß nur sehr wenige Probanden das Thema Schwerpunkte bearbeitet haben ($n = 6$).[7] Ein signifikanter Unterschied zwischen den beiden Themen läßt sich nicht feststellen. Das Ergebnis des t-Tests für unabhängige Stichproben zeigt die Tabelle 5.2. Bemerkenswert ist darin die hohe Standardabweichung beim Thema Schwerpunkte von mehr als zwei Stufen auf der Rangskala.

Interpretation Über die Hälfte aller Befragten fiel die Handhabung vollständig oder fast vollständig leicht. Somit kann davon ausgegangen werden, daß das Online-Skript als Ganzes angemessen einfach zu handhaben ist. Auch die Mehrfenstertechnik hat den Probenden anscheinend keine großen Probleme bereitet.[8] Interessant ist, daß zwischen den Geschlechtern ein signifikanter Unterschied in der Beurteilung der Handhabung liegt. Den männlichen Teilnehmern fällt die Handhabung leichter.

Nach einer Studie von Fittkau & Maaß[9] sind $77,5$ % aller Internetnutzer männlichen Geschlechts. Vermutlich besitzen auch bei der vorliegenden Stichprobe die männlichen Probanden im Umgang mit dem Internet und der Hypertextstruktur von Web-Seiten mehr Erfahrung. In diesem Zusammenhang wäre es sicherlich sinnvoll gewesen, auch die Vorkenntnisse der Probanden im Umgang mit dem Internet zu messen.

Zwischen den Themen ließ sich keine signifikante Mittelwert-Differenz feststellen. Dieses Ergebnis wurde auch erwartet, da die Handhabung eine globale Eigenschaft des Online-Skripts ist.

[7] Dieses gilt auch für alle folgenden Ergebnisse zur themenspezifischen Differenzierung.

[8] Ich war zu Beginn der Untersuchung unsicher, ob die Mehrfenstertechnik den Probanden Schwierigkeiten bereiten würde. Es kann nämlich passieren, daß Beweis- oder Figurenfenster in den Hintergrund geraten und nicht mehr sichtbar sind.

[9] Fittkau & Maaß 1999

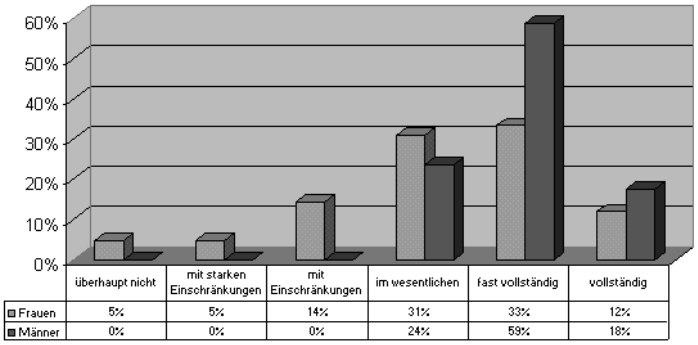

Abbildung 5.7: Geschlechtsspezifische Beurteilung der Handhabung.

Tabelle 5.1: Geschlechtsspezifische Beurteilung der Handhabung.

	n	$\bar{x}$	s	s^2
Frauen	42	$3,19$	$1,25$	$1,57$
Männer	17	$3,94$	$0,66$	$0,43$

$t = -2,99$ $df = 53$ $p = 0,004$

Die Mittelwert-Differenz ($\alpha = 5$ %, zweiseitig) ist signifikant.

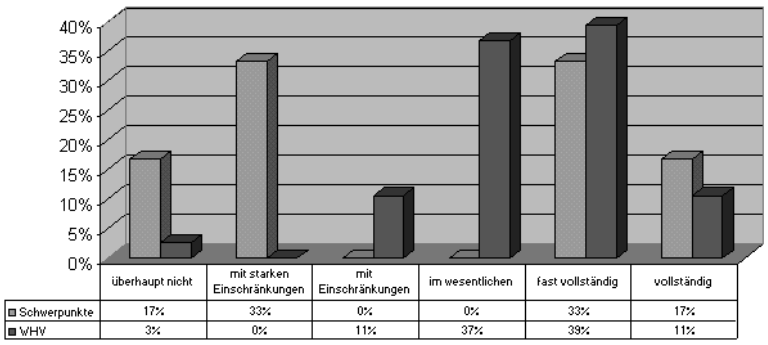

Abbildung 5.8: Themenspezifische Beurteilung der Handhabung.

Tabelle 5.2: Themenspezifische Beurteilung der Handhabung.

	n	$\bar{x}$	s	s^2
Schwerpunkte	6	$2,50$	$2,07$	$4,30$
Winkelh.-V.	38	$3,42$	$1,00$	$1,00$

$t = -1,07$ $df = 5$ $p = 0,34$

Die Mittelwert-Differenz ($\alpha = 5$ %, zweiseitig) ist nicht signifikant.

Gliederung

Mit Hilfe der Variablen Gliederung sollte gemessen werden, ob die Probanden den Aufbau des Online-Skripts als zweckmäßig und die Unterteilung der Themen in Abschnitte und Unterabschnitte als angemessen einschätzen. Der Aufbau des Online-Skripts bezieht sich dabei auf die sternförmige Hypertextstruktur, bei der die Web-Seite "Themenübersicht" als Ausgangspunkt bei der Navigation dient (Abbildung 5.2 auf Seite 183). Der Schüler kehrt immer wieder zu dieser Seite zurück, um etwa ein neues Thema aufzurufen. Die Gliederung der einzelnen Lehrtexte in Abschnitte und Unterabschnitte ist ähnlich wie in einem Lehrbuch angeordnet. Hierzu sollte geprüft werden, ob die Befragten diese Darbietungsform auch für das Medium Internet als angemessen beurteilen.

Der Aussage *Die Gliederung des Online-Skripts erscheint mir sinnvoll* haben die Probanden – wie in Abbildung 5.9 dargestellt – zugestimmt. Die deskriptive

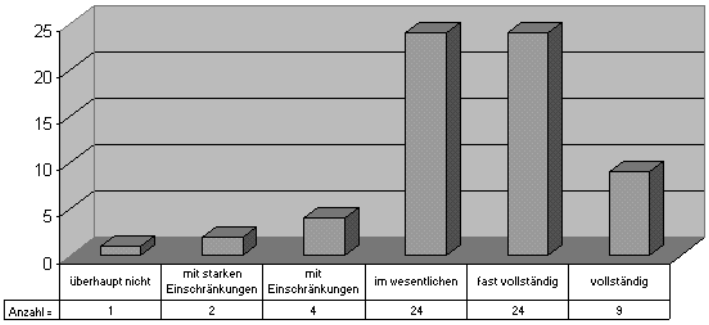

Abbildung 5.9: Deskriptive Auswertung der Variable Gliederung.

Auszählung ($n = 64$, $\bar{x} = 3,48$, $s = 1,02$) zeigt, daß die Mehrheit der Befragten der obigen Aussage positiv zustimmen kann. Dieses Ergebnis soll aber noch weiter nach dem Geschlecht differenziert werden. Die Abbildung 5.10 zeigt die prozentuale Verteilung der Antworten unterteilt nach Frauen und Männern. Im Vergleich beurteilen Männer die Gliederung durchschnittlich um $0,5$ Skalenpunkte positiver als Frauen. Insgesamt läßt sich jedoch kein signifikanter Mittelwert-Unterschied auf dem Niveau von $\alpha = 5$ % nachweisen (Tabelle 5.3). Untersucht man die Antworten differenziert nach den bearbeiteten Themen, so zeigt sich für die Gliederung des Themas Winkelhalbierenden-Vierecke eine positive Tendenz in der Beurteilung, während die Gliederung des Themas Schwerpunkte sehr unterschiedlich bewertet wird (Abbildung 5.11). Obwohl das Thema Winkelhalbierenden-Vierecke durchschnittlich um fast einen Skalenrang zustimmender eingestuft wird (Mittelwert-Differenz $= 0,95$), läßt sich mit Hilfe des t-Tests kein signifikanter Unterschied nachweisen (Tabelle 5.4).

Interpretation Weniger als 10 % aller Befragten beurteilen die Gliederung als eingeschränkt sinnvoll oder noch ablehnender. Über die Hälfte aller Probanden bewertet sie als vollständig sinnvoll oder fast vollständig sinnvoll. Vor diesem Ergebnis kann der Aufbau des Online-Skripts und die Gliederung der Themen als übersichtlich und leicht nachvollziehbar bezeichnet werden. Eine lehrbuchar-

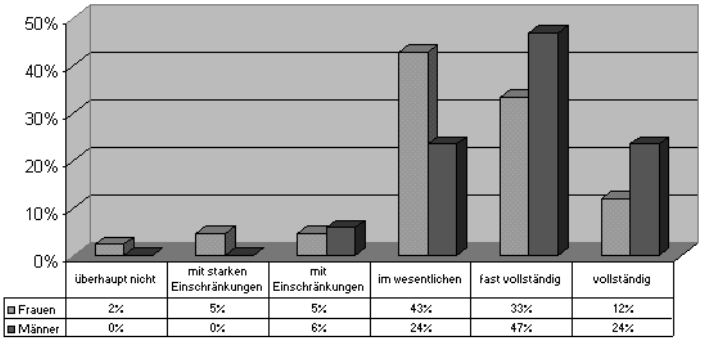

Abbildung 5.10: Geschlechtsspezifische Bewertung der Gliederung.

Tabelle 5.3: Geschlechtsspezifische Bewertung der Gliederung.

	n	$\bar{x}$	s	s^2
Frauen	42	3,36	1,08	1,16
Männer	17	3,88	0,86	0,74

$t = 1,79 \qquad df = 57 \qquad p = 0,08$

Die Mittelwert-Differenz ($\alpha = 5$ %, zweiseitig) ist nicht signifikant.

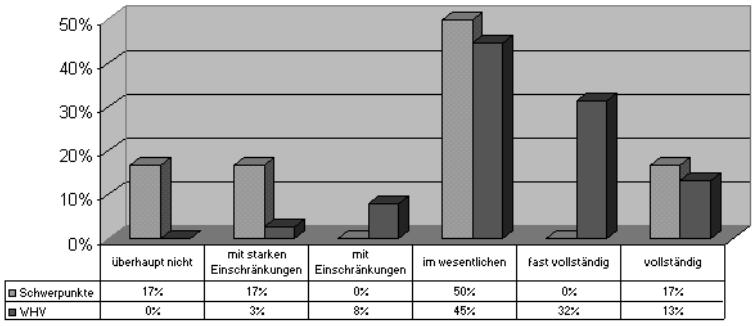

Abbildung 5.11: Themenspezifische Beurteilung der Gliederung.

Tabelle 5.4: Themenspezifische Beurteilung der Gliederung.

	n	$\bar{x}$	s	s^2
Schwerpunkte	6	2,50	1,76	3,10
Winkelh.-V.	38	3,45	0,92	0,85

$t = -1,29 \qquad df = 5 \qquad p = 0,25$

Die Mittelwert-Differenz ($\alpha = 5$ %, zweiseitig) ist nicht signifikant.

tige Gliederung ist den Probanden vermutlich bekannt und hilft ihnen, sich in der Hypertextstruktur der Web-Seiten schnell zurechtzufinden und die Orientierung nicht zu verlieren. Auf diese Weise wird auch ein zielloses Aufrufen von Verweisen vermieden.

Daß die Gliederung des Themas Schwerpunkte zwar nicht signifikant, aber tendenziell negativer beurteilt wird, mag darin begründet liegen, daß der Lehrtext mehr als doppelt so umfangreich und die einzelnen Abschnitte länger und weiter unterteilt sind als beim Thema Winkelhalbierenden-Vierecke.

Sprachprägnanz

Mit der Variablen Sprachprägnanz sollte untersucht werden, ob die sprachliche Wiedergabe der Definitionen und Sätze und die Beschreibung der Beweise in dem Online-Skript von den Befragten als klar und treffend beurteilt wird. Dazu wurden die Probanden befragt, inwieweit sie der Aussage *Die sprachliche Darstellung erscheint mir prägnant* zustimmen können. Die Abbildung 5.12 stellt das deskriptive Ergebnis dar. Die einfache Auszählung zeigt, daß

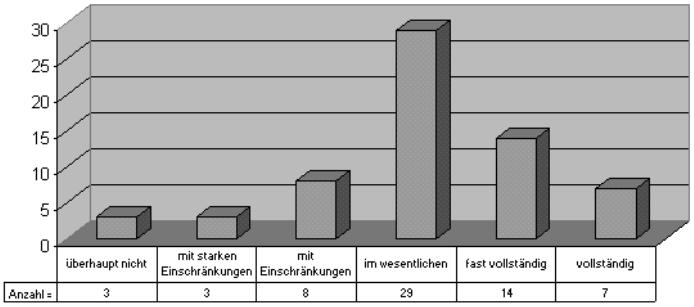

Abbildung 5.12: Deskriptive Auswertung der Variable Sprachprägnanz.

die Sprachprägnanz überwiegend positiv eingeschätzt wird ($n = 64$, $\bar{x} = 3,08$, $s = 1,19$). Wird dieses Ergebnis nach Geschlecht differenziert, so ergibt sich eine ausnahmslos zustimmende Bewertung durch die männlichen Teilnehmer. Dagegen empfinden ein Drittel der Teilnehmerinnen die sprachliche Darstellung nur mit Einschränkungen, mit starken Einschränkungen oder überhaupt nicht prägnant (Abbildung 5.13). Mit Hilfe des t-Tests läßt sich ein signifikanter Unterschied bezüglich der Mittelwert-Differenz feststellen (Tabelle 5.5). Betrachtet man die Beurteilungen differenziert nach den Themen Schwerpunkte und Winkelhalbierenden-Vierecke, so sind überraschenderweise bei dem Thema Schwerpunkte alle sechs Skalenwerte gleich häufig gewählt worden. Dadurch läßt sich keine Tendenz ablesen. Die Sprachprägnanz beim anderen Thema wird hingegen mehrheitlich positiv beurteilt (Abbildung 5.14). Die Prüfung auf Signifikanz der Mittelwert-Unterschiede fällt negativ aus. Eine Differenz von $0,5$ Skalenpunkten ist hier nicht ausreichend (Tabelle 5.6).

Interpretation Im Hinblick auf die deskriptive Auswertung läßt sich feststellen, daß die sprachliche Darstellung von drei Viertel aller Befragten als prägnant

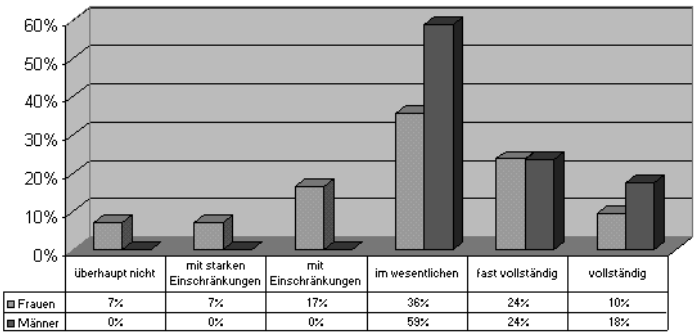

Abbildung 5.13: Geschlechtsspezifische Bewertung der Sprachprägnanz.

Tabelle 5.5: Geschlechtsspezifische Bewertung der Sprachprägnanz.

	n	$\bar{x}$	s	s^2
Frauen	42	$2,90$	$1,32$	$1,75$
Männer	17	$3,59$	$0,80$	$0,63$

$t = -2,44$ $\quad df = 48$ $\quad p = 0,02$

Die Mittelwert-Differenz ($\alpha = 5$ %, zweiseitig) ist signifikant.

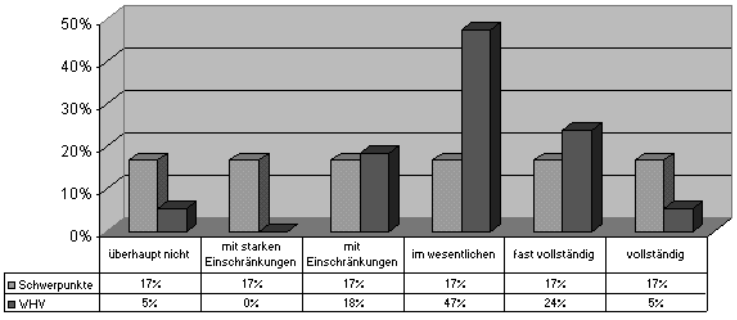

Abbildung 5.14: Themenspezifische Differenzierung der Sprachprägnanz.

Tabelle 5.6: Themenspezifische Beurteilung der Sprachprägnanz.

	n	$\bar{x}$	s	s^2
Schwerpunkte	6	$2,50$	$1,87$	$3,50$
Winkelh.-V.	38	$3,00$	$1,07$	$1,14$

$t = -0,64$ $\quad df = 6$ $\quad p = 0,55$

Die Mittelwert-Differenz ($\alpha = 5$ %, zweiseitig) ist nicht signifikant.

empfunden wird (45 % stimmen im wesentlichen zu, 20 % fast vollständig und 10 % vollständig). Daraus läßt sich schließen, daß der verwendete Sprachstil für die Zielgruppe Lehramtsstudierende im Fach Mathematik adäquat ist.

Ein interessantes Ergebnis ist die Signifikanz in der geschlechtsspezifischen Beurteilung. Es wäre sicherlich lohnenswert, in einer vertiefenden Studie zu untersuchen, worin hierfür die Ursache liegt. Ich konnte dafür allerdings keine Begründung finden.

5.4.2 Umgang mit den Lernbausteinen

Im folgenden soll untersucht werden, wie die Probanden den Umgang mit den Lernbausteinen in dem Online-Skript beurteilen. Dazu betrachte ich nacheinander die Variablen Übersichtlichkeit, Erwartungskonformität und Figureninteraktion und erläutere jeweils die gewonnenen Daten.

Übersichtlichkeit

Mit der Variablen Übersichtlichkeit wurde gemessen, ob die Probanden den Aufbau der Figur in einem Lernbaustein leicht überblicken können. Bei der praktischen Entwicklung des Online-Skripts bestand das Problem, eine geeignete Größe für die Zeichenfläche der Lernbausteine zu bestimmen. Die Zeichenfläche durfte einerseits nicht zu groß sein, da sie in den Textfluß des Lehrtexts eingebettet werden sollte. Auf der anderen Seite benötigen komplexe Figuren viel Fläche, um für den Schüler übersichtlich zu bleiben. In der methodischen Umsetzung wurde daher ein Kompromiß gewählt: Die Zeichenfläche besitzt eine Größe von 360×480 Bildschirmpunkten und ist damit relativ klein. Um Platz zu sparen, können deshalb in vielen Lernbausteinen Teile der Figur und Textfenster ein- und ausgeblendet werden. Die Abbildung 5.15 zeigt, wie die Befragten der Aussage *Die Figuren waren übersichtlich* zugestimmt haben. Schon auf den

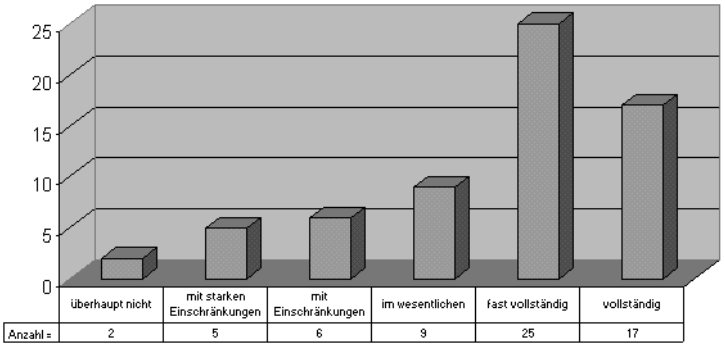

Abbildung 5.15: Deskriptive Auswertung der Variable Übersichtlichkeit.

ersten Blick zeigt sich hier eine rechtssteile Verteilung und damit eine eher positive Beurteilung der Übersichtlichkeit. Als deskriptive Kennwerte ergeben sich: $n = 64$, $\bar{x} = 3,58$ und $s = 1,36$. Das Ergebnis soll nach Geschlecht differenziert werden. Die Abbildung 5.16 stellt die entsprechende prozentuale Verteilung dar. Offensichtlich beurteilen die Teilnehmerinnen die Übersichtlichkeit der Figuren

etwas kritischer als die männlichen Befragten. In der Tendenz wird jedoch der obigen Aussage am häufigsten vollständig oder fast vollständig zugestimmt. Ein signifikanter Mittelwert-Unterschied läßt sich nicht nachweisen (Tabelle 5.7).

Bei der themenspezifischen Differenzierung zeigt sich ein interessantes Phänomen (Abbildung 5.17). Während beim Thema Winkelhalbierenden-Vierecke die Figuren zu über 60 % als vollständig und fast vollständig übersichtlich eingestuft werden, polarisieren die Einschätzungen beim Thema Schwerpunkte zu den Skalenendpunkten hin. Ein Drittel der Befragten beurteilt die Figuren zum Thema Schwerpunkte als vollständig übersichtlich. Dagegen empfinden 50 % diese als überhaupt nicht oder nur mit starken Einschränkungen übersichtlich. Insgesamt ist die Mittelwert-Differenz von $1,13$ Skalenpunkten zwischen den beiden Themen jedoch nicht signifikant (Tabelle 5.8).

Interpretation Aufgrund der Tatsache, daß zwei Drittel aller Befragten die Figuren als vollständig oder fast vollständig übersichtlich beurteilen, kann der gewählte Kompromiß zwischen Zeichenflächengröße und Figurenaufbau als angemessen bezeichnet werden.

Die polarisierende Bewertung der Übersichtlichkeit der Schwerpunkt-Figuren, zeigt aber auch, daß hier Schwierigkeiten aufgetreten sind. Die Ursache mag darin liegen, daß die didaktische Funktion der Lernbausteine beim Thema Schwerpunkte in erster Linie in dem Vorführen von Bewegungsphasen (Abschnitt 4.1.2) besteht. Die Lernbausteine demonstrieren in mehreren Phasen, wie die Schwerpunkte bei Dreiecken und Vierecken bestimmt werden. Dadurch wird der Aufbau der Figuren komplexer und weniger übersichtlich. Als Konsequenz sind diese Lernbausteine von mir noch einmal gezielt überarbeitet worden. Dabei wurde versucht, Textfenster, geometrische Figur und Schieberegler (Schalter) auf der Zeichenfläche stets gleichbleibend anzuordnen und dadurch die Übersichtlichkeit zu verbessern.

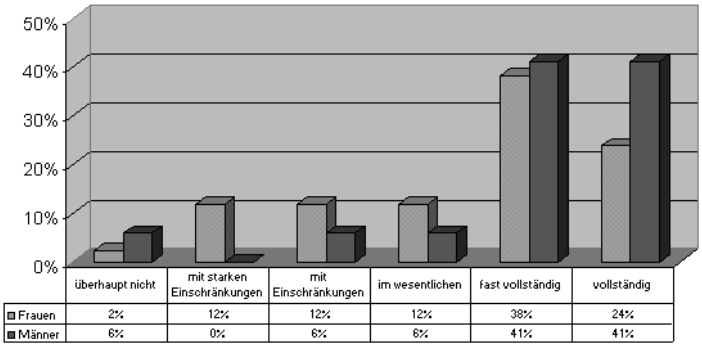

Abbildung 5.16: Geschlechtsspezifische Bewertung der Übersichtlichkeit.

Tabelle 5.7: Geschlechtsspezifische Bewertung der Übersichtlichkeit.

	n	$\bar{x}$	s	s^2
Frauen	42	$3,43$	$1,42$	$2,01$
Männer	17	$4,00$	$1,32$	$1,75$

$t = 1,43 \qquad df = 57 \qquad p = 0,15$

Die Mittelwert-Differenz ($\alpha = 5$ %, zweiseitig) ist nicht signifikant.

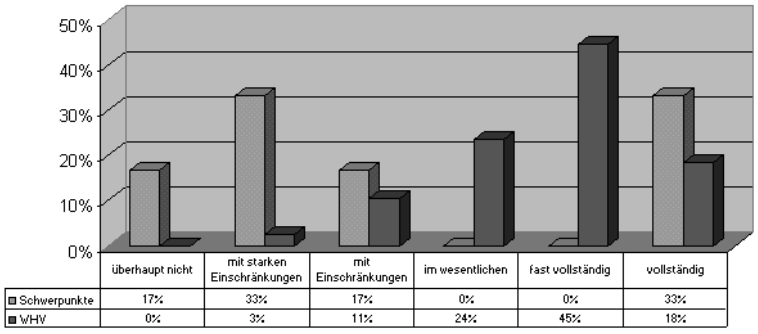

Abbildung 5.17: Themenspezifische Beurteilung der Übersichtlichkeit.

Tabelle 5.8: Themenspezifische Beurteilung der Übersichtlichkeit.

	n	$\bar{x}$	s	s^2
Schwerpunkte	6	$2,50$	$2,17$	$4,70$
Winkelh.-V.	38	$3,63$	$1,02$	$1,05$

$t = -1,26 \qquad df = 5 \qquad p = 0,26$

Die Mittelwert-Differenz ($\alpha = 5$ %, zweiseitig) ist nicht signifikant.

Erwartungskonformität

Die mit Erwartungskonformität bezeichnete Variable sollte messen, ob die Probanden intuitiv verstehen und nachvollziehen können, wie die Figuren sich bei Variation verhalten. Dabei ist vor allem interessant, wie die Befragten Lernbausteine bewerten, in denen Bewegungsphasen vorgeführt werden, wie es beim Thema Schwerpunkte der Fall ist. Um Auskunft darüber zu erhalten, wurden die Teilnehmerinnen und Teilnehmer befragt, inwieweit sie der Aussage *Die Figuren funktionierten, wie ich es erwartete* zustimmen konnten. Die Abbildung 5.18 stellt das Ergebnis der deskriptiven Auswertung dar. Die Auszählung der

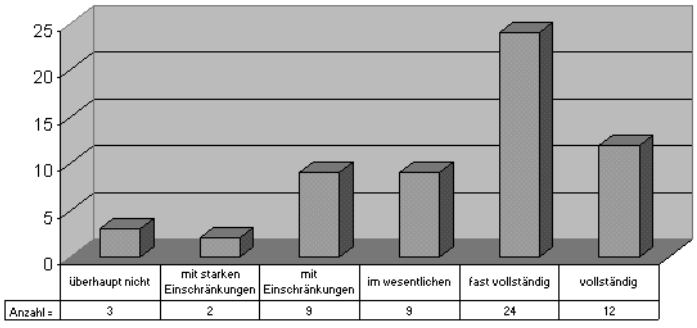

Abbildung 5.18: Deskriptive Auswertung der Variable Erwartungskonformität.

$n = 59$ Fragebögen liefert eine tendenziell positive Beurteilung der Erwartungskonformität ($\bar{x} = 3,44$ und $s = 1,34$). Untersucht man das Ergebnis getrennt nach weiblichen und männlichen Befragten, so ergibt sich die in Abbildung 5.19 dargestellte Verteilung. Die männlichen Teilnehmer stimmen der obigen Aussage zu 88 % im wesentlichen oder stärker zu. Die Erwartungskonformität wird von ihnen also als positiv bewertet. Lediglich 12 % erwarteten ein anderes Figurenverhalten. Von den Teilnehmerinnen sind 71 % in ihren Erwartungen tendenziell bestätigt worden. Insgesamt läßt sich aber kein signifikanter Unterschied zwischen den Geschlechtern feststellen (Tabelle 5.9).

Die themenspezifische Differenzierung zeigt, daß das Figurenverhalten beim Thema Winkelhalbierenden-Vierecke zum allergrößten Teil als erwartungskonform eingestuft wird (84 % stimmen im wesentlichen und stärker zu). Dagegen ist es beim Thema Schwerpunkte weniger positiv beurteilt worden. Immerhin die Hälfte der zum Thema Schwerpunkte Befragten konnte nur mit Einschränkungen, mit starken Einschränkungen oder überhaupt nicht zustimmen (Abbildung 5.20). Das Ergebnis des t-Tests erweist den Unterschied zwischen den Mittelwerten jedoch nicht als signifikant (Tabelle 5.10).

Interpretation Die Antwort auf die Frage nach der Erwartungskonformität des Figurenverhaltens hängt zu einem Teil davon ab, welche Vorkenntnisse die Probanden im Umgang mit dynamischer Geometrie besitzen. Ist ihnen der Unterschied zwischen ziehbaren und nicht-ziehbaren Punktobjekten bekannt? Haben sie schon einmal eine Figur im Zugmodus selbsttätig bewegt oder sogar konstruiert?

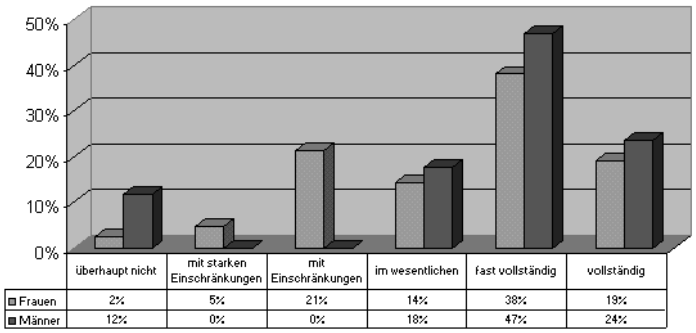

Abbildung 5.19: Geschlechtsspezifische Bewertung der Erwartungskonformität.

Tabelle 5.9: Geschlechtsspezifische Bewertung der Erwartungskonformität.

	n	$\bar{x}$	s	s^2
Frauen	42	3,38	1,29	1,66
Männer	17	3,59	1,50	2,26

$t = 0,53 \qquad df = 57 \qquad p = 0,60$

Die Mittelwert-Differenz ($\alpha = 5$ %, zweiseitig) ist nicht signifikant.

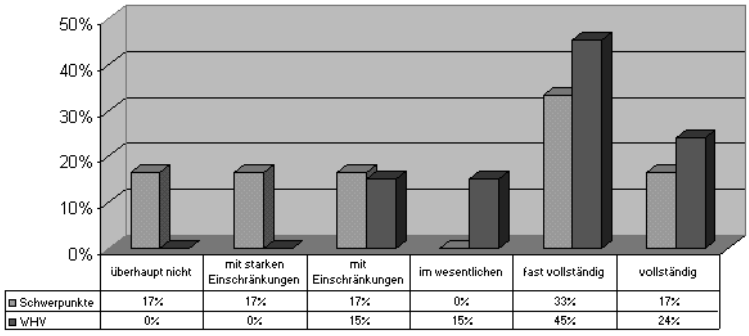

Abbildung 5.20: Themenspezifische Beurteilung der Erwartungskonformität.

Tabelle 5.10: Themenspezifische Beurteilung der Erwartungskonformität.

	n	$\bar{x}$	s	s^2
Schwerpunkte	6	2,67	1,97	3,87
Winkelh.-V.	33	3,79	0,99	0,98

$t = -1,37 \qquad df = 5 \qquad p = 0,23$

Die Mittelwert-Differenz ($\alpha = 5$ %, zweiseitig) ist nicht signifikant.

Nach Auskunft der Versuchsleiter besaßen die Probanden nur geringe Vorkenntnisse beim Arbeiten mit DG-Systemen. Vor diesem Hintergrund ist das Ergebnis, daß über 60 % der Befragten vollständig oder fast vollständig in ihren Erwartungen an das Figurenverhalten bestätigt worden sind, ausgesprochen positiv zu sehen. Es kann davon ausgegangen werden, daß die Lernbausteine weitgehend intuitiv verständlich sind.

Wie erwartet fiel die Bewertung des Figurenverhaltens beim Thema Schwerpunkte zwar nicht signifikant, aber tendenziell negativer aus, als beim Thema Winkelhalbierenden-Vierecke. Die Ursache dafür mag darin liegen, daß beim Vorführen von Bewegungsphasen die Variationsmöglichkeit einer Figur auf gewisse Bahnen und eine festgelegte Reihenfolge eingeschränkt ist.

Figureninteraktion

Die Variable Figureninteraktion mißt, wie die Probanden die Interaktion mit den Figuren in den Lernbausteinen beurteilen. Mit Interaktion ist in erster Linie das Variieren im Zugmodus gemeint. Neben dieser direkten Form kann eine Figur aber auch indirekt variiert werden, indem der Schüler Schieberegler verändert oder Schalter betätigt. Bei den Lernbausteinen zur Selbstkontrolle kommt außerdem die Interaktion mit der Menüleiste, etwa den Menüpunkt "Auswerten" anwählen, hinzu. Inwieweit die Befragten der Aussage *Die Interaktion mit den Figuren fiel mir leicht* zugestimmt haben, zeigt die Abbildung 5.21. Bereits auf

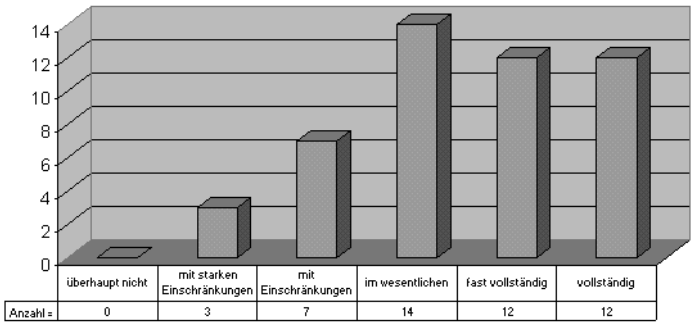

Abbildung 5.21: Deskriptive Auswertung der Variable Figureninteraktion.

den ersten Blick läßt sich eine sehr positive Einschätzung ablesen ($n = 48$, $\bar{x} = 3,48$, $s = 1,20$). Lediglich 21 % der Befragten beurteilen die Interaktion nur mit Einschränkungen oder mit starken Einschränkungen als leicht. Bemerkenswert ist die geschlechtsspezifische Differenzierung dieses Ergebnisses (Abbildung 5.22). Den männlichen Probanden fiel die Interaktion mit den Figuren deutlich leichter als den weiblichen Probanden. Die Mittelwert-Differenz von $1,18$ Skalenpunkten kann nach Durchführung des t-Tests mit $\alpha = 1$ % als sehr signifikant bezeichnet werden (Tabelle 5.11). Bei der Differenzierung der Antworten zwischen den Themen Schwerpunkte und Winkelhalbierenden-Vierecke wird das Thema Schwerpunkte von den Befragten sehr unterschiedlich bewertet. Die Hälfte von ihnen empfindet die Interaktion mit den Figuren nur mit Einschränkungen oder mit starken Einschränkungen als leicht, während die andere

Hälfte die Interaktion als vollständig und fast vollständig leicht einschätzt.

Beim Thema Winkelhalbierenden-Vierecke wird die Interaktion zu über 80 % mit im wesentlichen leicht (und besser) beurteilt (Abbildung 5.23). Ein signifikanter Mittelwert-Unterschied läßt sich zwischen den beiden Themen allerdings nicht feststellen (Tabelle 5.12).

Interpretation Die Hälfte aller Probanden bezeichnen die Interaktion mit den Figuren als vollständig oder fast vollständig leicht, ein weiteres Viertel stuft sie als im wesentlichen leicht ein. Da die Lehramtsstudierenden in Weingarten und Schwäbisch Gmünd nach Angaben der Versuchsleiter nur wenig Erfahrung im Umgang mit DG-Systemen besaßen, kann davon ausgegangen werden, daß man mit den Figuren auch ohne spezielle Einweisung arbeiten kann. In dem Online-Skript sind unter der Überschrift "Hinweise" verschiedene Formen der Interaktion an Beispielfiguren erläutert, etwa wie eine Figur im Zugmodus bewegt werden kann oder wie man Schalter und Schieberegler bedient.

Bemerkenswert ist unter den aufgeführten Ergebnissen der sehr signifikante Mittelwert-Unterschied zwischen den männlichen und weiblichen Probanden. Vermutlich besitzen die männlichen Teilnehmer – ähnlich wie bei der Variable Handhabung – mehr Erfahrung im Umgang mit Computern allgemein und können sich daher schneller und leichter auf unbekannte Programme einstellen. Die Ursache für die tendenziell schlechtere Beurteilung der Figuren beim Thema Schwerpunkte mag darin liegen, daß die Figuren größtenteils Bewegungsphasen demonstrieren. Dabei muß der Schüler Punktobjekte auf bestimmen Bahnen und in einer festgelegten Reihenfolge führen. Diese Figuren sind dadurch weniger flexibel zu variieren als die beim Thema Winkelhalbierenden-Vierecke.

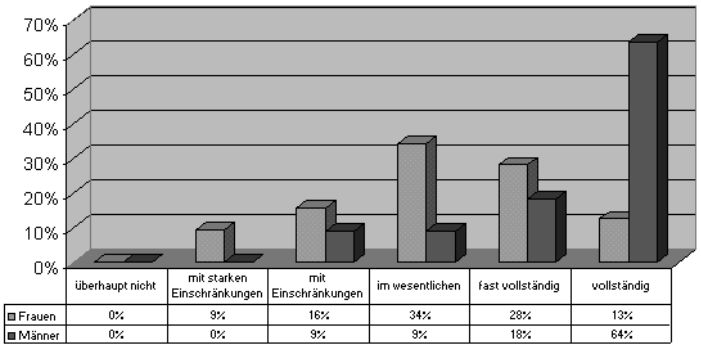

Abbildung 5.22: Geschlechtsspezifische Bewertung der Figureninteraktion.

Tabelle 5.11: Geschlechtsspezifische Bewertung der Figureninteraktion.

	n	$\bar{x}$	s	s^2
Frauen	32	3,19	1,15	1,32
Männer	11	4,36	1,03	1,05

$t = 3,00 \qquad df = 41 \qquad p = 0,005$

Die Mittelwert-Differenz ($\alpha = 1$ %, zweiseitig) ist sehr signifikant.

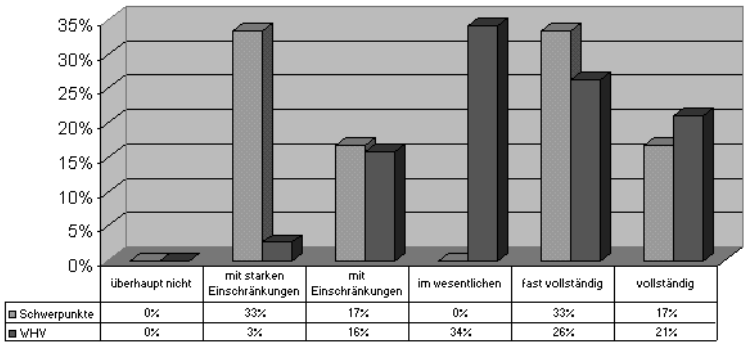

Abbildung 5.23: Themenspezifische Beurteilung der Figureninteraktion.

Tabelle 5.12: Themenspezifische Beurteilung der Figureninteraktion.

	n	$\bar{x}$	s	s^2
Schwerpunkte	6	2,83	1,72	2,97
Winkelh.-V.	38	3,47	1,08	1,17

$t = -0,88 \qquad df = 6 \qquad p = 0,59$

Die Mittelwert-Differenz ($\alpha = 5$ %, zweiseitig) ist nicht signifikant.

5.4.3 Unterstützung des Lernprozesses

Die Unterstützung des Lernprozesses wurde durch die drei Variablen Verständniserleichterung, Bearbeitungsfreude und Aufgabenschwierigkeitsgrad erfaßt. In den folgenden drei Abschnitten werde ich die Ergebnisse zu diesen Variablen detailliert darstellen.

Verständniserleichterung

Durch die Variable Verständniserleichterung sollte gemessen werden, ob und wie sehr die Lernbausteine den Probanden geholfen haben, die Inhalte des Lehrtexts zu verstehen und nachzuvollziehen. Dazu wurden die Teilnehmerinnen und Teilnehmer aufgefordert, zu der Aussage *Die interaktiven Figuren haben mir das Verständnis erleichtert* den Grad ihrer Zustimmung anzugeben. Die Abbildung 5.24 zeigt die Antwortverteilung zu diesem Item. Die deskriptive Auswertung

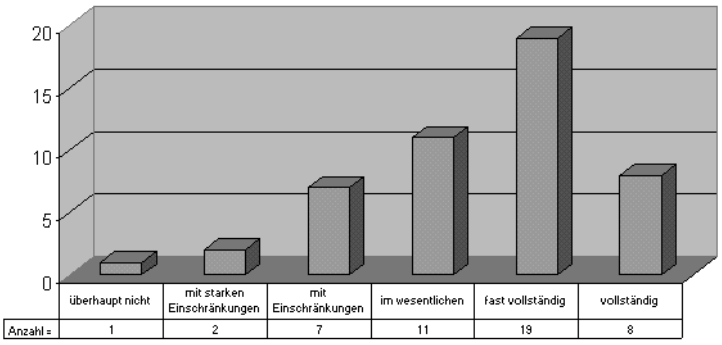

Abbildung 5.24: Deskriptive Auswertung der Variable Verständniserleichterung.

der $n = 48$ Fragebögen zur Variable Verständniserleichterung liefert eine positive Rückmeldung ($\bar{x} = 3,44$, $s = 1,18$). Über die Hälfte der Befragten können der oben genannten Aussage vollständig oder fast vollständig zustimmen. Differenziert man die Antworten nach Geschlecht, so stufen die männlichen Teilnehmer die Verständniserleichterung geringfügig stärker ein als die Teilnehmerinnen (Abbildung 5.25). Die Mittelwert-Differenz von $0,63$ Skalenpunkten ist jedoch nicht signifikant (Tabelle 5.13). Bei der themenspezifischen Auswertung zeigt sich für das Thema Schwerpunkte eine divergente Antwortverteilung. Genau 50 % der zu diesem Thema Befragten geben an, daß die Figuren ihnen das Verständnis überhaupt nicht oder nur mit starken Einschränkungen erleichtert hätten. Die anderen 50 % dagegen antworteten positiv zustimmend. Beim Thema Winkelhalbierenden-Vierecke dagegen werden die Figuren zu über 80 % als im wesentlichen, fast vollständig oder vollständig verständniserleichternd bezeichnet (Abbildung 5.26). Das Ergebnis der Signifikanzprüfung zwischen den beiden Stichproben ist jedoch negativ (Tabelle 5.14).

Interpretation Als Ergebnis kann herausgestellt werden, daß die Probanden die Verständniserleichterung durch die interaktiven Figuren positiv beurteilen: 23 % stimmen der obigen Aussage im wesentlichen zu, 40 % fast vollständig

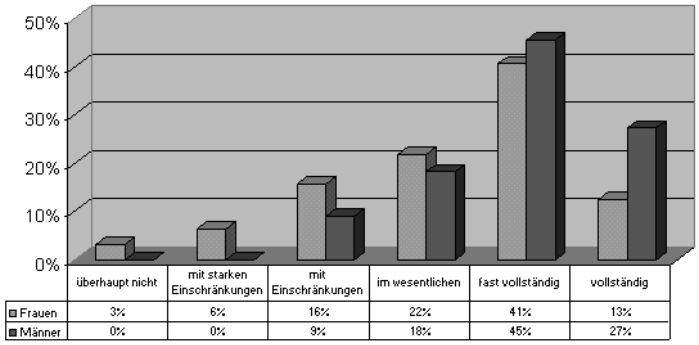

Abbildung 5.25: Geschlechtsspezifische Bewertung der Verständniserleichterung.

Tabelle 5.13: Geschlechtsspezifische Bewertung der Verständniserleichterung.

	n	$\bar{x}$	s	s^2
Frauen	32	$3,28$	$1,25$	$1,56$
Männer	11	$3,91$	$0,94$	$0,89$

$t = 1,52 \qquad df = 41 \qquad p = 0,13$

Die Mittelwert-Differenz ($\alpha = 5$ %, zweiseitig) ist nicht signifikant.

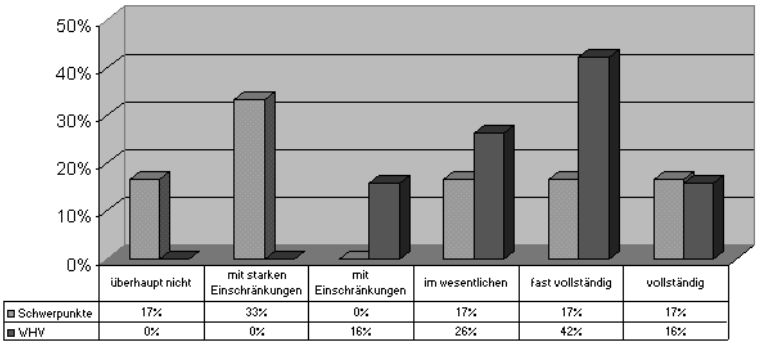

Abbildung 5.26: Themenspezifische Beurteilung der Verständniserleichterung.

Tabelle 5.14: Themenspezifische Beurteilung der Verständniserleichterung.

	n	$\bar{x}$	s	s^2
Schwerpunkte	6	$2,33$	$1,97$	$3,87$
Winkelh.-V.	38	$3,58$	$0,95$	$0,90$

$t = -1,52 \qquad df = 5 \qquad p = 0,19$

Die Mittelwert-Differenz ($\alpha = 5$ %, zweiseitig) ist nicht signifikant.

und 17 % vollständig. Diese Antworten legen nahe, daß die Figuren für den Lernprozeß des Schülers eine wichtige, unterstützende Rolle spielen.

Insbesondere die Figuren zum Thema Winkelhalbierenden-Vierecke erleichtern das Verständnis des Lerninhalts. Die didaktische Intention der Figuren zu diesem Thema liegt überwiegend in dem Visualisieren der Sätze. Während beim Thema Schwerpunkte das Demonstrieren von Bewegungsphasen im Vordergrund stand. Die Antworten zu diesem Thema zeigen allerdings, daß dieses auch einigen Probanden Probleme bereitet hat. Als Konsequenz wurden von mir alle Lernbausteine, in denen Bewegungsphasen demonstriert werden, noch einmal überarbeitet und die zugehörigen Anweisungstexte noch deutlicher formuliert.

Insgesamt stimmen die subjektiven Einschätzungen mit dem Ergebnis einer Lehrerbefragung von Schumann (1994, S. 35) überein. Zu der Frage *Wie schätzen Sie den Einfluß oder die Wirkung von* Cabri-Géomètre *auf ihr eigenes Geometrieverständnis ein?* wählten 73 % der Befragten auf einer fünfstufigen Rangskala die beiden positivsten Skalenwerte.

Bearbeitungsfreude

Mit der Variablen Bearbeitungsfreude sollte gemessen werden, ob die Probanden Vergnügen an der Arbeit mit dem Online-Skript finden. Dazu wurden diese befragt, inwieweit sie der Aussage *Ich habe gerne mit dem Online-Skript gearbeitet* zustimmen können. Die Abbildung 5.27 zeigt das Ergebnis der einfachen Auszählung. Durch den Mittelwert von $\bar{x} = 2,72$ ($n = 47$, $s = 1,10$) ist nur eine

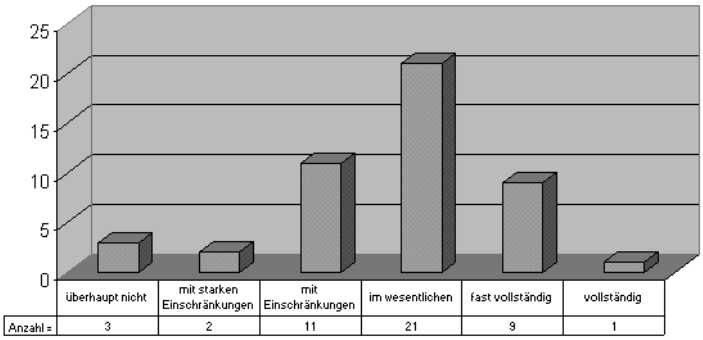

Abbildung 5.27: Deskriptive Auswertung der Variable Bearbeitungsfreude.

leicht positive Zustimmung auszumachen. Die Differenzierung nach Geschlecht weist keine wesentlichen Unterschiede auf (Abbildung 5.28). Die Antworten der weiblichen Probanden streuen zwar um 0,4 Skalenpunkte stärker als die der männlichen Teilnehmer. Die Mittelwert-Differenz von 0,20 Skalenpunkten ist jedoch ausgesprochen gering und nicht signifikant (Tabelle 5.15).

Differenziert man die Ergebnisse bezüglich der Bearbeitungsfreude nach den beiden Themen, so wird deutlich, daß die Hälfte der Personen, die das Thema Schwerpunkte bearbeitet haben, überhaupt nicht gerne oder nur mit starken Einschränkungen gerne mit dem Online-Skript gearbeitet haben. Dagegen empfinden beim Thema Winkelhalbierenden-Vierecke über 65 % im wesentlichen (oder stärker) Gefallen an der Arbeit (Abbildung 5.29). Obwohl der Mittelwert-

Unterschied von $1,25$ Skalenpunkten relativ hoch ist, läßt sich keine Signifikanz nachweisen (Tabelle 5.16).

Interpretation Von den Befragten gaben über 65 % an, im wesentlichen, vollständig oder fast vollständig gerne mit dem Online-Skript gearbeitet zu haben. Die Ursache dafür, daß die Bearbeitungsfreude bei den verbleibenden 35 % weniger ausgeprägt ist, hat vermutlich mehrere Gründe. Hier spielen auch äußere Faktoren mit hinein, wie etwa die Einstellung gegenüber dem Medium Computer, dem Interesse an der Elementargeometrie, dem Vorwissen auf diesem Gebiet oder der Geschwindigkeit der Übertragung.

Ein Proband schreibt als Anmerkung in einem Evaluationsformular, daß die Übertragungsleistung des Internets ihn sehr beim Ausprobieren und Erforschen der interaktiven Figuren gestört und ihm dem Spaß dadurch genommen hat.

Betrachtet man die Korrelationen der einzelnen Variablen untereinander, so korrelieren die Wartezeit und die Bearbeitungsfreude allerdings nur sehr gering ($r = 0,14$). In höherer Korrelation mit der Bearbeitungsfreude stehen hingegen die Variablen Gliederung ($r = 0,51$), Handhabung ($r = 0,49$) und Erwartungskonformität ($r = 0,45$) (Tabelle im Anhang F (Seite 411)).

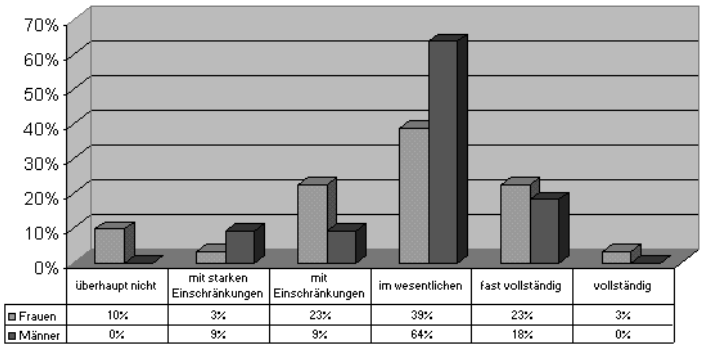

Abbildung 5.28: Geschlechtsspezifische Bewertung der Bearbeitungsfreude.

Tabelle 5.15: Geschlechtsspezifische Bewertung der Bearbeitungsfreude.

	n	$\bar{x}$	s	s^2
Frauen	31	$2,71$	$1,24$	$1,55$
Männer	11	$2,91$	$0,83$	$0,69$

$t = 0,49 \qquad df = 40 \qquad p = 0,63$

Die Mittelwert-Differenz ($\alpha = 5$ %, zweiseitig) ist nicht signifikant.

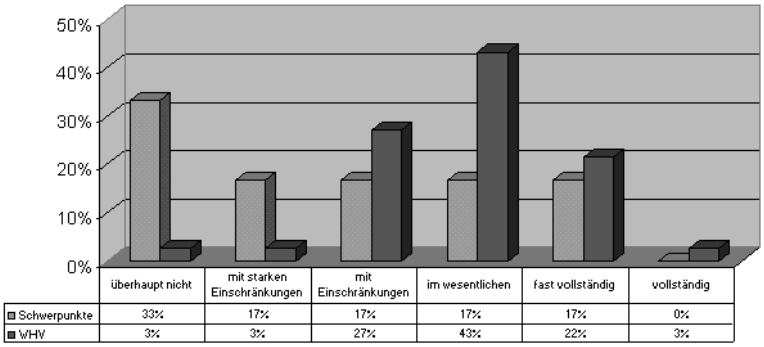

Abbildung 5.29: Themenspezifische Beurteilung der Bearbeitungsfreude.

Tabelle 5.16: Themenspezifische Beurteilung der Bearbeitungsfreude.

	n	$\bar{x}$	s	s^2
Schwerpunkte	6	$1,67$	$1,63$	$2,67$
Winkelh.-V.	37	$2,86$	$0,98$	$0,95$

$t = -1,75 \qquad df = 6 \qquad p = 0,13$

Die Mittelwert-Differenz ($\alpha = 5$ %, zweiseitig) ist nicht signifikant.

Aufgabenschwierigkeitsgrad

Mit der Variablen Aufgabenschwierigkeitsgrad sollte gemessen werden, wie leicht oder wie schwer die Befragten die Aufgaben in dem Online-Skript einstufen. Aufgaben kommen darin in zwei Formen vor. Als erstes gibt es Lernbausteine zur Selbstkontrolle, die in den Lehrtext eingebettet sind. Diese bearbeitet der Schüler, indem er die Figur in geeigneter Weise variiert. Danach kann er eine Antwortanalyse anfordern. Als zweites gibt es zu jedem Thema eine Sammlung von Satz- und Beweisfindungsaufgaben. In der Abbildung 5.30 ist dargestellt, inwieweit die Probanden der Aussage *Ich konnte die Aufgaben leicht lösen* zugestimmt haben. Auf den ersten Blick zeigt sich, daß die meisten Antworten

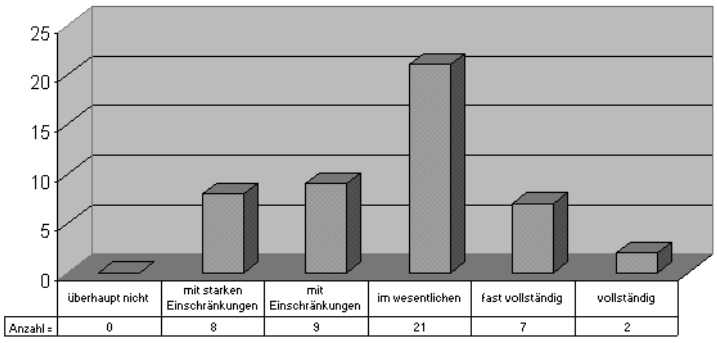

Abbildung 5.30: Deskriptive Auswertung der Variable Aufgabenschwierigkeitsgrad.

eine leicht positive Tendenz aufweisen ($n = 47$, $\bar{x} = 2,70$, $s = 1,06$). Als vollständig oder fast vollständig leicht werden die Aufgaben von weniger als 20 % der Probanden beurteilt. Fast ebenso viele (17 %) stufen die Aufgaben als schwierig ein (nur mit starken Einschränkungen leicht). Interessant ist die geschlechtsspezifische Differenzierung dieses Ergebnisses (Abbildung 5.31). Die Aufgaben werden von den männlichen Teilnehmern zu 90 % als im wesentlichen leicht oder fast vollständig leicht bewertet ($\bar{x} = 3,30$). Die Antworten weisen dabei nur eine geringe Standardabweichung von $s = 0,67$ Skalenpunkten auf. Dagegen liegt der Mittelwert der Antworten der Teilnehmerinnen bei $\bar{x} = 2,53$. Die Mittelwert-Differenz ist signifikant (Tabelle 5.17). Untersucht man die Antworten zum Aufgabenschwierigkeitsgrad differenziert zwischen den Themen, so ist auffällig, daß die Hälfte der Befragten, die das Thema Schwerpunkte bearbeitet haben, angeben, die Aufgaben seien nur mit starken Einschränkungen leicht. Im Unterschied dazu werden die Aufgaben zum Thema Schwerpunkte von einem Drittel als fast vollständig leicht eingestuft. Die Aufgaben zum zweiten Thema hingegen werden weniger polarisierend beurteilt: Knapp drei Viertel wählten die beiden mittleren Skalenwerte (Abbildung 5.32). Ein signifikanter Mittelwert-Unterschied läßt sich zwischen den beiden Themen allerdings nicht feststellen (Tabelle 5.18).

Interpretation Etwa zwei Drittel aller Befragten finden die Aufgaben im wesentlichen leicht oder mit geringen Einschränkungen leicht und weisen ihnen die

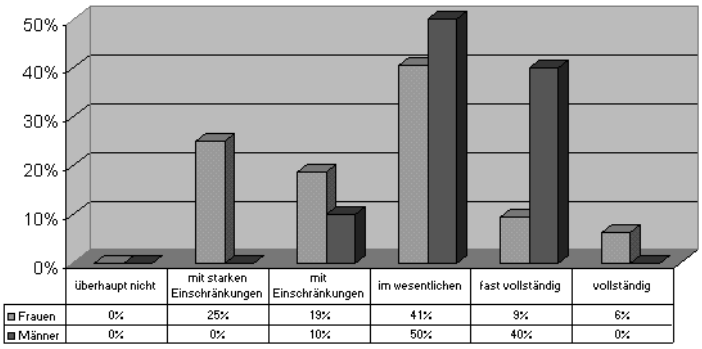

Abbildung 5.31: Geschlechtsspez. Bewertung des Aufgabenschwierigkeitsgrads.

Tabelle 5.17: Geschlechtsspez. Bewertung des Aufgabenschwierigkeitsgrads.

	n	$\bar{x}$	s	s^2
Frauen	32	$2,53$	$1,16$	$1,35$
Männer	10	$3,30$	$0,67$	$0,46$

$t = -2,59 \qquad df = 27 \qquad p = 0,01$

Die Mittelwert-Differenz ($\alpha = 5$ %, zweiseitig) ist signifikant.

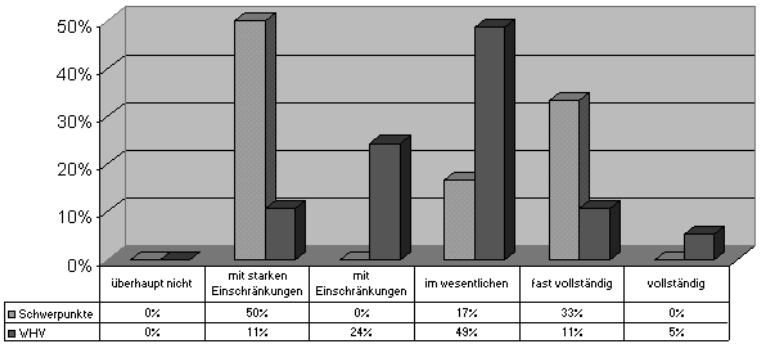

Abbildung 5.32: Themenspez. Beurteilung des Aufgabenschwierigkeitsgrads.

Tabelle 5.18: Themenspez. Beurteilung des Aufgabenschwierigkeitsgrads.

	n	$\bar{x}$	s	s^2
Schwerpunkte	6	$2,33$	$1,51$	$2,27$
Winkelh.-V.	37	$2,76$	$0,98$	$0,97$

$t = 0,91 \qquad df = 41 \qquad p = 0,63$

Die Mittelwert-Differenz ($\alpha = 5$ %, zweiseitig) ist nicht signifikant.

beiden mittleren Skalenwerte zu. Vor dem Hintergrund, daß eine gute Aufgabe weder zu leicht noch zu schwierig sein darf, scheint der Schwierigkeitsgrad der Aufgaben gerade angemessen zu sein. Konkrete Aussagen über einzelne Aufgaben können allerdings nicht gemacht werden, da durch die Evaluation nicht erfaßt wurde, welche einzelnen Aufgaben die Probanden richtig oder nicht richtig gelöst haben.

5.4.4 Wartezeit

Mit der Variablen Wartezeit sollte gemessen werden, ob die Dauer für das erstmalige Aufrufen des Java-Applets *Geometria*, die Zeitspanne für das Initialisieren der einzelnen Lernbausteine und die Zeitdauer für das Laden der Web-Seiten von den Befragten akzeptiert wird. Dabei hängt die Wartezeit in erster Linie von der Übertragungsgeschwindigkeit des jeweiligen Internetzugangs ab. Das Laden und Initialisieren des ca. 300 KB großen Applets dauert auch bei schneller Internetanbindung mindestens 60-90 Sekunden.

Die Abbildung 5.33 zeigt, inwieweit die Probanden der Aussage *Die Wartezeit beim Laden und Initialisieren war akzeptabel* zugestimmt haben. Die de-

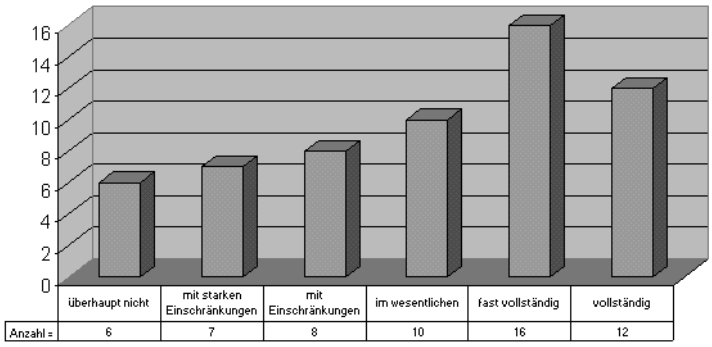

Abbildung 5.33: Deskriptive Auswertung der Variable Wartezeit.

skriptive Auswertung ($n = 59$, $\bar{x} = 3,0$, $s = 1,63$) ergibt insgesamt eine eher positive Akzeptanz der Wartezeit. Über 60 % der Befragten finden sie vollständig oder fast vollständig akzeptabel. Dem stehen 22 % der Probanden gegenüber, die die Wartezeit nur mit starken Einschränkungen oder überhaupt nicht billigen können.

Eine weitere Differenzierung ist wenig aussagekräftig. Das obige Ergebnis wird dadurch relativiert, daß etwa die Hälfte der Befragten (alle Probanden von der PH Schwäbisch Gmünd) das Online-Skript als eine lokal installierte Offline-Version verwendet haben, wodurch die Ladezeiten erheblich reduziert worden sind. Wie bereits erwähnt, mußte an der PH Weingarten ein Evaluationsversuch abgebrochen werden, weil der gleichzeitige Zugriff mit mehr als zwanzig Rechnern kein sinnvolles Arbeiten mehr möglich machte.[10] Die Übertragungsgeschwindigkeit über das Internet war zu gering. Dies ist im Prinzip das Hauptergebnis im Bezug auf die Variable Wartezeit.

[10] Diese Gruppe hat auch später keine Fragebögen mehr ausgefüllt und ist damit nicht mit in die Stichprobe aufgenommen worden.

5.4.5 Technische Fehler und Verbesserungsvorschläge

Ein Ziel der Evaluation des Online-Skripts war es, festzustellen, ob und welche Programmfehler beim praktischen Einsatz auftreten und welche Verbesserungen vorgeschlagen werden. Dazu war in den Evaluationsformularen, die die Probanden ausgefüllt haben, ein Freiraum vorgesehen, in dem eine Antwort frei formuliert werden konnte.

Aus Platzgründen sollen an dieser Stelle nicht sämtliche einzelnen, sondern nur die häufigsten Rückmeldungen zusammengefaßt aufgelistet werden. Als Kritikpunkte wurden genannt:

- Es werden zu viele Fachbegriffe und Kenntnisse vorausgesetzt. Mehrfach wurde auch ein Nachschlagelexikon gewünscht, um die verwendeten Begriffe und Abkürzungen noch einmal nachlesen zu können.
- Nicht bei allen Figuren stimmten die im Lehrtext gebrauchten Bezeichnungen mit den in den Figuren verwendeten überein. Diese Kritik bezog sich vor allem auf das Thema Winkelhalbierenden-Vierecke, in dem anfangs keine griechischen Buchstaben verwendet wurden, um die Winkel zu bezeichnen.
- Nicht zu allen Aufgaben sind Lösungen oder eine Antwortanalyse vorhanden. Einige Probanden empfanden die Aufgaben als zu schwer und zu umfangreich.
- Einige Befragte schreiben, daß es Fehler beim Laden der Lernbausteine gab oder daß die Ladezeiten zu lange dauerten.

Als Konsequenz dieser Befragung wurde das Online-Skript um ein alphabetisches Stichwortverzeichnis erweitert. Dieses ist zwar kein Nachschlagelexikon, aber der Schüler kann den Index verwenden, um die entsprechende Stelle im Lehrtext aufzurufen, an der ein Begriff definiert wird (Abschnitt 5.2.2).

Damit Bezeichnungen in Text und Figur übereinstimmend wiedergegeben werden können, wurde *Geometria* so erweitert, daß auch griechische Buchstaben als Objektbezeichner möglich sind.

Das Problem der langen Ladezeiten ist abhängig von der Geschwindigkeit der Internetanbindung und kann höchstens dadurch verbessert werden, daß die Größe des Applets *Geometria* reduziert wird. Jedoch sind auch dieser Alternative Grenzen gesetzt. Fehler beim Laden oder genauer beim Initialisieren treten äußerst selten auf und auch nur, wenn mehrere Lernbausteine auf einer Web-Seite dargestellt werden. Hier ist es ausreichend, die Seite noch einmal erneut zu laden. Problematisch kann es allerdings sein, wenn der Ladevorgang mehrfach vom Anwender unterbrochen wird, etwa durch Aufrufen von neuen Web-Seiten. Dieses kann zum Absturz des Web-Browsers führen.

Bis auf eine Ausnahme hat keiner der Befragten davon berichtet, daß ein Neustart eines Rechners erforderlich wurde. Sobald das Applet vollständig geladen und initialisiert ist, läuft es ausgesprochen stabil und robust. Softwaretechnische Fehler wurden durch die Probanden nicht festgestellt.

5.4.6 Zusammenfassung der Ergebnisse

Durch die Evaluation sollte herausgearbeitet werden, wie das Online-Skript beim praktischen Einsatz von den Probanden bewertet wird. Dazu wurden diese auf-

gefordert, den Aufbau und die Darstellung, den Umgang mit den Figuren und die Unterstützung des Lernprozesses zu beurteilen (Abschnitt 5.3.1). Neben dieser inhaltlichen Rückmeldung wurde außerdem nach der softwaretechnischen Funktionstüchtigkeit und der Beurteilung der Wartezeit beim Laden und Initialisieren gefragt. Hier war das erste Ergebnis der Evaluation, daß mit mehr als zwanzig Rechnern nicht gleichzeitig auf das Online-Skript zugegriffen werden kann. Soll eine größere Lerngruppe zeitgleich mit dem Online-Skript arbeiten, so ist zu empfehlen, es auf den lokalen Rechnern zu installieren oder es gezielt nacheinander aufzurufen. Das eigentliche Java-Applet *Geometria* läuft – sobald es vollständig geladen und initialisiert ist – stabil und robust. Die Probanden haben keine softwaretechnischen Fehler zurückgemeldet.

Der Aufbau und die Darstellung wurde wie folgt bewertet. Jeweils mehr als die Hälfte der Befragten stimmen vollständig oder fast vollständig zu, daß das Online-Skript leicht zu handhaben und sinnvoll gegliedert ist. Die Sprachprägnanz wird von einem Drittel so beurteilt.

Etwas besser wird von den Befragten der Umgang mit den Lernbausteinen bewertet. Jeweils über 60 % stimmen vollständig oder fast vollständig zu, daß die Figuren übersichtlich sind und erwartungsgemäß funktionieren. Ebenso viele meldeten zurück, daß Ihnen die Interaktion leicht fiel.

Im Hinblick auf die Unterstützung des Lernprozesses gaben über 50 % der Befragten an, die Figuren hätten ihnen das Verständnis vollständig oder fast vollständig erleichtert. Etwas getrübter dagegen ist die Freude beim Arbeiten eingeschätzt worden. Hier sind es nur etwas über 20 %, die die beiden positiven Endskalenwerte wählten.

Als Konsequenz auf die Verbesserungsvorschläge durch die Probanden wurde das Online-Skript nach der Evaluation um einen Index erweitert, durch den Begriffe schnell nachgeschlagen werden können. Ferner wurden die Objektbeschriftungen in den verwendeten Lernbausteinen überarbeitet und eine Galerie für einen direkten und alternativen Figurenzugriff hinzugefügt (Seite 182).

Bemerkenswertes Ergebnis bei der geschlechtsspezifischen Analyse ist, daß den weiblichen Probanden die Handhabung, die Interaktion mit den Figuren und das Lösen der Aufgaben signifikant weniger leicht fiel als den männlichen Probanden. Auch beurteilen sie die sprachliche Darstellung als weniger prägnant. Der Frage, ob sich solche Differenzen ausschließlich mit der allgemein größeren Erfahrung von Männern im Umgang mit Computern und Internet erklären lassen, wäre in einer vertiefenden Studie nachzugehen und kann nicht mit dem vorhandenen Datenmaterial untersucht werden.

Abschließend möchte ich feststellen: Das Online-Skript zur Elementargeometrie steht allen Internetnutzern kostenlos zur Verfügung und kann als didaktisches Medium zum Lehren und Lernen eingesetzt werden. Meine Hoffnung ist, daß vor allem die interaktiven Figuren bei den Lernenden Freude und Interesse an der Geometrie wecken und sie bestenfalls dazu anregen, eigene Lernbausteine zu entwickeln und auf Web-Seiten zu publizieren.

Anhang

Anhang A
Benutzeranleitung

Inhaltsverzeichnis

Vorbemerkungen

Das Java-Applet *Geometria* ermöglicht, geometrische Lernbausteine auf Web-Seiten darzustellen. Definiert werden diese in der Skriptsprache *GeoScript*. Mit dieser Benutzeranleitung möchte ich zeigen, wie man eigene Figuren und Aufgaben entwickeln kann.

Stellen Sie dazu bitte sicher, daß neben dieser Anleitung noch folgende Unterlagen verfügbar sind:

- die Konstruktionsreferenz zu *Geometria*,
- alle Beispiel-Dateien zu dieser Anleitung sowie
- die Java-Archiv-Datei *Geometria.jar*.

Aktuelle und überarbeitete Versionen aller Unterlagen finden Sie im Internet auf den Web-Seiten des Instituts für Mathematik und ihre Didaktik der Universität Flensburg unter der Adresse: `http://www.uni-flensburg.de/mathe/`.

Rotenburg (Wümme), im Juli 2000

Timo Ehmke

Systemanforderungen

Um geometrische Lernbausteine mit *Geometria* zu betrachten und zu erstellen, sind die folgenden Voraussetzungen erforderlich:

Hardware

- 1 MB Festplattenspeicher
- mindestens 16 MB Arbeitsspeicher
- Prozessor ab Pentium 166 Mhz

Software

- beliebiger Texteditor
- Java-Virtual-Machine ab Version 1.1.5
  (z. B. Netscape Communicator 4.0.6, Internet Explorer 4.0, HotJava 1.1.5)
- AppletViewer aus dem Java-Development-Kit
  (Dieses Programm ist für den Entwicklungsprozeß besonders geeignet, es geht aber auch ohne.)

1 Der Arbeitszyklus des Figurenautors

Bei der Entwicklung eines geometrischen Lernbausteins besteht der Arbeitszyklus des Figurenautors aus drei Schritten:

1. **Skriptdatei anlegen** Das Anlegen einer Skriptdatei ist der umfangreichste Teil des Entwicklungsprozesses. In einem Skript werden durch *GeoScript*-Ausdrücke sämtliche Bestandteile eines Lernbausteins definiert. Dazu zählen neben der eigentlichen geometrischen Figur auch Textfenster, Bilder und Hilfen. Ferner kann zu bestimmten Aufgaben eine Antwortanalyse eingebunden werden. Gespeichert wird ein Skript in einer Datei mit der Endung ".script".

2. **Layout-Vorlage erzeugen** Das Erscheinungsbild eines Lernbausteins wird in einer Layout-Vorlage festgelegt. Diese besteht aus einer Textdatei, in der den Systemvariablen von *Geometria* konkrete Werte zugewiesen werden. Verwendet man eine Layout-Vorlage für mehrere Lernbausteine, so ist eine einheitliche Darstellung gesichert. Jede Layout-Vorlage muß die Dateiendung ".style" besitzen.

3. **HTML-Dokument erzeugen** Im dritten Arbeitsschritt muß der Figurenautor ein HTML-Dokument erstellen, in das das Java-Applet *Geometria* eingebunden wird. Als Applet-Parameter werden die Dateinamen eines Skripts und einer Layout-Vorlage übergeben. Sobald ein solches HTML-Dokument von einem Web-Browser interpretiert wird, startet dieser den entsprechenden Lernbaustein.

Nach dem dritten Schritt beginnt der Arbeitszyklus oftmals bei Punkt 1, da meistens der erste Entwurf eines Skripts noch verbessert werden kann. Sind außerdem noch Fehler in einem Skript enthalten, so werden entsprechende Meldungen und Kommentare vom Web-Browser ausgegeben.

2 Diskussion von Beispielen

In diesem Kapitel werden ich an drei exemplarischen Beispielen darstellen, wie Skripte und Layout-Vorlagen aufgebaut sind und wie das Java-Applet *Geometria* in eine Web-Seite eingebunden werden kann.

2.1 Satz von Varignon

Nach dem Satz von Varignon bilden die Seitenmitten eines beliebigen Vierecks ein Parallelogramm. Die in Abbildung 34 dargestellte Figur visualisiert diesen Sachverhalt.

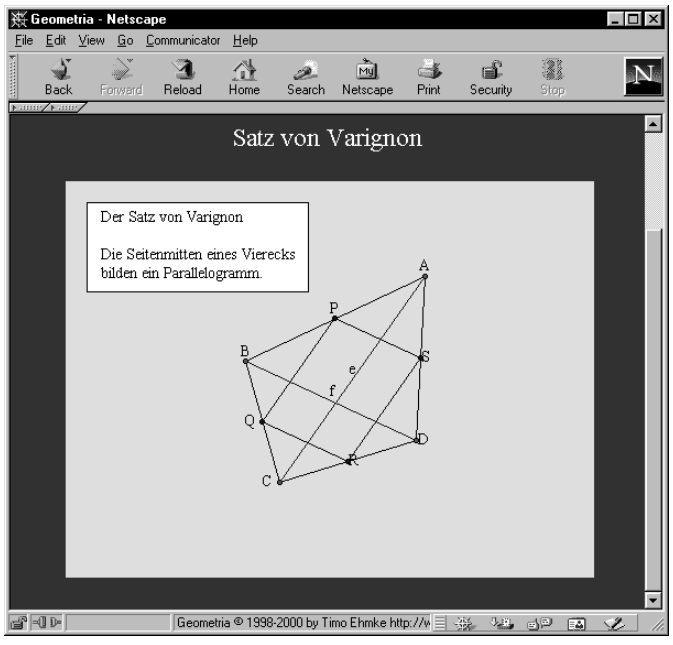

Abbildung 34: Figur zum Satz von Varignon.

2.1.1 Skript

Die Definition der Figur erfolgt in dem Skript `Satz_von_Varignon.script`, das den folgenden Inhalt besitzt:

```
   //
   // Datei:    Satz_von_Varignon.script
   //

 5 // Figurenbeschreibung
   // ====================

   e[1]  = A;  point;   dragable;  5.7,6.3;
   e[2]  = B;  point;   dragable;  -5.0,4.0;
10 e[3]  = C;  point;   dragable;  -3.1,-6.2;
   e[4]  = D;  point;   dragable;  4.1,-2.75;
   e[5]  = p1; polygon; polygon;   A,B,C,D;   "hideLabel"
   e[6]  = P;  point;   midpoint;  A,B;
   e[7]  = Q;  point;   midpoint;  C,B;
15 e[8]  = R;  point;   midpoint;  C,D;
   e[9]  = S;  point;   midpoint;  A,D;
   e[10] = p2; polygon; polygon;   P,Q,R,S;   "hideLabel"
   e[11] = e;  line;    connect;   A,C;
   e[12] = f;  line;    connect;   B,D;
20
   // Textfenster
   // ===========

   <Textbox>
25  Position = 20;20;200;80
      Der Satz von Varignon

      Die Seitenmitten eines Vierecks
      bilden ein Parallelogramm.
30 </Textbox>
```

Der Inhalt des Skripts besteht im wesentlichen aus Kommentierungen, der Figurenbeschreibung und der Definition eines Textfensters.

Kommentierungen Alle Zeilen eines Skripts, die mit der Zeichenfolge `//` beginnen, werden bei der Interpretation durch den Parser ignoriert. Auf diese Weise lassen sich beliebige Kommentare einfügen. In dem Beispielskript stehen Kommentare in den Zeilen 1-3, 5-6 und 21-22.

Figurenbeschreibung Die Figurenbeschreibung erfolgt nach einem Schema, das einer aus der Zeichenblatt-Geometrie bekannten Konstruktionsbeschreibung ähnelt. Pro Zeile wird genau ein geometrisches Objekt beschrieben. Dabei ist der Aufbau jeweils gleich. Alle zu erzeugenden Objekte werden listenartig durchnumeriert: `e[1]`, ..., `e[n]`. Nach dem Gleichheitszeichen folgt die eigentliche Definition des Objekts. Dazu müssen mindestens vier Konstruktionsdaten angegeben werden, die durch Semikolon separiert sind. Im einzelnen sind dies:

1. **Objektbezeichner** Mit dem Objektbezeichner wird das zu erzeugende Objekt auf der Zeichenfläche beschriftet. Dabei wird die Groß- und Kleinschreibung unterschieden. Der Objektbezeichner dient auch zum Referenzieren des Objekts bei der Definition von anderen Objekten. Daher dürfen keine Objektbezeichner doppelt gewählt werden.

2. **Klassenbezeichner** Durch den Klassenbezeichner wird die Klasse des zu erzeugenden Objekts bestimmt. Mögliche Klassen sind: `point` (Punkte),

`line` (Strecken, Strahlen, Geraden), `circle` (Kreise), `sector` (Kreisabschnitte), `polygon` (Polygone) und `measure` (Funktionale).

3. **Unterklassenbezeichner** Durch den Unterklassenbezeichner wird die Unterklasse des zu erzeugenden Objekts genau spezifiziert. Weil eine große Anzahl von Unterklassen verfügbar ist, möchte ich auf die Konstruktionsreferenz verweisen, in der diese systematisch dokumentiert sind.

4. **Objektdaten** Die Objektdaten bestehen in den meisten Fällen aus mehreren durch Kommata getrennten Parameterwerten, die zur Konstruktion und Initialisierung eines Objekts dienen. Die Konstruktionsreferenz gibt Auskunft darüber, welche Parameter als Objektdaten übergeben werden müssen.

5. **Layout-Angaben** Optional kann jedes Objekt mit Layout-Angaben versehen werden. Der Befehl `hideLabel` unterdrückt das Anzeigen des Objektbezeichners. Der Befehl `hidden` blendet ein Objekt vollständig aus. Für spezielle von der Layout-Vorlage abweichende Farb- und Formdarstellungen können außerdem konkrete Werte angegeben werden (siehe Konstruktionsreferenz).

In dem Beispielskript wird die Figur in den Zeilen 7-19 beschrieben. In der Zeile 7 wird dabei ein ziehbarer Punkt A definiert, dessen Anfangskoordinaten $x = 5,7$ und $y = 6,3$ sind. Die Zeile 12 enthält die Definition eines Polygons mit den vier Eckpunkten $ABCD$. Das Polygon zeichnet den Kantenzug zwischen den angegebenen Eckpunkten. Wegen des Befehls `hideLabel` wird der Objektbezeichner $p1$ nicht auf der Zeichenfläche angezeigt. In den Zeilen 13-16 werden vier Mittelpunkte definiert. Dazu sind jeweils bei den Objektdaten die Objektbezeichner von zwei Punkten angegeben, zu denen der Mittelpunkt konstruiert wird. Die Diagonalen des Vierecks $ABCD$ werden in den Zeilen 18 und 19 des Skripts definiert. Der Unterklassenbezeichner `connect` steht für eine Strecke. Als Objektdaten sind die beiden Streckenendpunkte anzugeben.

Textfenster Nach der Figurenbeschreibung folgt in dem Skript die Definition eines Textfensters. Sie beginnt mit dem Befehl `<Textbox>` in Zeile 24 und endet mit `</Textbox>`. Nach dem einleitenden Befehl wird die Position des Textfensters auf der Zeichenfläche festgelegt (Zeile 25). Die linke, obere Ecke besitzt die Fensterkoordinaten $(20, 20)$ und das Fenster ist 200 Bildschirmpunkte breit und 80 Bildschirmpunkte hoch. Der in den Zeilen 26-29 aufgeführte Text bildet den Inhalt des Textfensters.

2.1.2 Layout-Vorlage

Mit Hilfe einer Layout-Vorlage wird das gestalterische Aussehen eines Lernbausteins festgelegt. Dazu werden den Systemvariablen von *Geometria* konkrete Werte und allen geometrischen Objektklassen ein spezielles Farbschema zugewiesen. Der folgende Quelltext zeigt den wesentlichen Inhalt der Layout-Vorlage "Demo.style":

```
//
// Datei:    Demo.style
//
```

```
 5 APPLET_WIDTH      = 480
   APPLET_HEIGHT     = 360
   WORLD_X_MAX       = +16.0
   WORLD_X_MIN       = -16.0
   WORLD_Y_MAX       = +12.0
10 WORLD_Y_MIN       = -12.0
   GRIDSIZE          = 10
   GRIDCOLOR         = 235,205,180
   FONTSIZE          = 14
   FONT              = SERIF
15 BACKGROUNDCOLOR   = 255,225,200
   APPLETBGCOLOR     = 255,255,255
   CONTROLPANELCOLOR = 255,225,200
   USESEPARATEWINDOW = FALSE
   LANGUAGE          = GERMAN
20 SHOWLABEL         = TRUE
   SHOWGRID          = FALSE
   SHOWAXIS          = FALSE
   SNAPTOGRID        = FALSE
   ALLPOINTSDRAGABLE = FALSE
25 CHECKSYMBOLS      = FALSE
   DRAGMEASURE       = FALSE
   MEASURE_EXACTNESS = 3

   //
30 // Layout der geometrischen Objekte
   //

   <elementTable>
    point;   dragable;          black; red;   black; 0;    smallCircle
35  point;   free;              black; red;   black; 0;    smallCircle
    point;   lineSlider;        black; red;   black; 0;    smallCircle
    point;   lineSegmentSlider; black; red;   black; 0;    smallCircle
    point;   circleSlider;      black; red;   black; 0;    smallCircle
    point;   curveSlider;       black; red;   black; 0;    smallCircle
40  point;   areaSlider;        black; red;   black; 0;    smallCircle
    point;   polygonSlider;     black; red;   black; 0;    smallCircle
    point;   horizontal;        black; red;   black; 0;    smallCircle
    point;   vertical;          black; red;   black; 0;    smallCircle
    point;   functionDepend;    black; blue;  black; 0;    smallCircle
45  point;   fixed;             black; black; 0;     0;    smallSquare
    line;    connect;           black; 0;     blue;  0;
    line;    straightline;      black; 0;     black; 0;
    line;    pointSet;          0;     blue;  blue;  blue;
    circle;  radius;            0;     0;     blue;  0;
50  polygon; polygon;           0;     0;     black; 0;
   </elementTable>
```

Die Systemvariablen werden in den Zeilen 5-27 mit Werten belegt. Die Größe der Zeichenfläche wird in den Zeilen 5 und 6 angegeben. Die Zeilen 7-10 definieren das darin befindliche Weltkoordinatensystem. Wichtig ist außerdem die Variable `USESEPARATEWINDOW`, die bestimmt, ob die Zeichenfläche in einem separaten Fenster oder innerhalb der Web-Seite angezeigt wird. Die Bedeutung der restlichen Variablen schlagen Sie bitte in der Konstruktionsreferenz nach.

Die gestalterische Darstellung der geometrischen Objekte wird in den Zeilen 34-50 tabellarisch festgelegt. In den ersten beiden Spalten wird eine Objektklasse durch den Klassen- und Unterklassenbezeichner angegeben. Die Spalten 3-6 enthalten Farbbezeichner für die Beschriftungsfarbe, die Punktfarbe, die Linienfarbe und die Flächenfarbe. Die siebte Spalte ist bei Punktobjekten für eine Formkonstante vorgesehen.

2.1.3 HTML-Dokument

In dem dritten Arbeitsschritt ist ein HTML-Dokument zu erstellen, in das *Geometria* samt Skript und Layout-Vorlage eingebunden wird. Der folgende Ausschnitt zeigt den entsprechenden HTML-Befehl der Web-Seite aus der Abbildung 34:

```
   <APPLET
      code     = "Geometria"
      codebase = ""
      archive  = "Geometria.jar"
 5    height   = "360"
      width    = "480" >
      <PARAM name = "script"      value = "Satz_von_Varignon.script">
      <PARAM name = "style"       value = "Demo.style">
      <PARAM name = "startButton" value = "0">
10 </APPLET>
```

In den ersten sechs Zeilen ist das Applet definiert. Dazu muß ein Klassenbezeichner (`Geometria`), ggf. eine Pfadangabe, eine Archiv-Datei (`Geometria.jar`) sowie die Größe des Applets in Bildschirmpunkten angegeben werden. Durch die Parameter `script` und `style` werden die entsprechenden Dateinamen übergeben (Zeile 7-8). Der Wert des dritten Parameters `startButton` legt fest, ob der Lernbaustein sofort beim Aufrufen des HTML-Dokuments angezeigt werden soll oder – als platzsparende Alternative – erst nach Auslösen eines Buttons in einem separaten Fenster. Im zweiten Fall müßte dazu die Variable `USESEPARATEWINDOW` in der Layout-Vorlage auf `TRUE` gesetzt werden.

2.2 Parabel

Der folgende Lernbaustein zeigt eine Parabel $f(x) = ax^2 + bx + c$, deren Parameter durch Schieberegler variiert werden können. Entlang der Parabel läßt sich ein Punkt P bewegen. Zu P wird die entsprechende Tangente an die Parabel angezeigt (Abbildung 35).

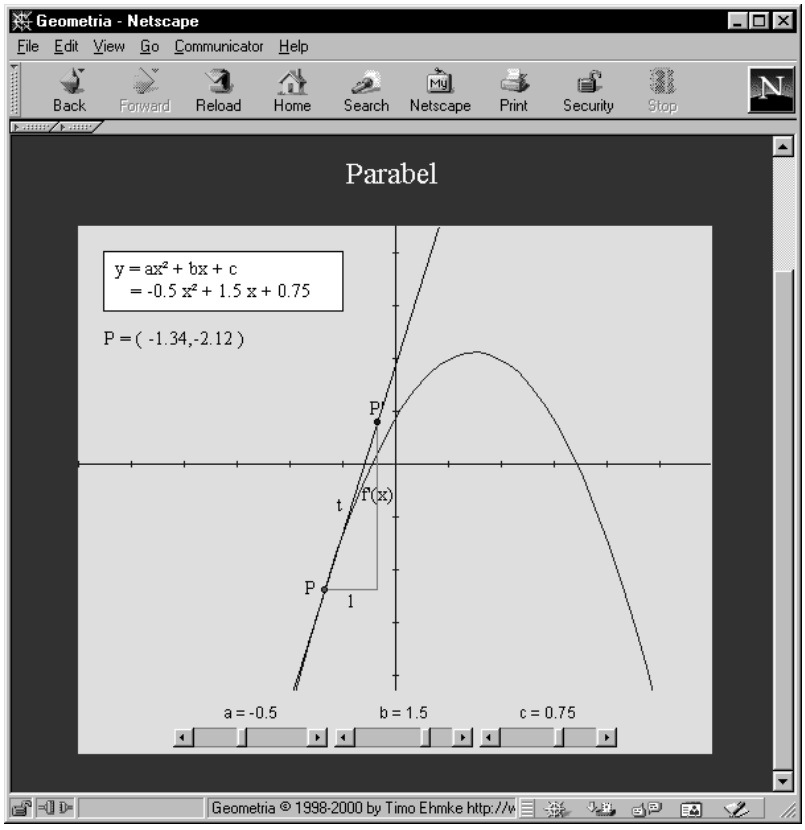

Abbildung 35: Parabel $f(x) = ax^2 + bx + c$.

2.2.1 Skript

Der in Abbildung 35 dargestellte Lernbaustein ist in der Datei `Parabel.script` definiert. Das Skript ist wie folgt aufgebaut:

```
  //
  // Datei:    Parabel.script
  //

5 // Systemvariablen
  // ===============

  WORLD_X_MAX   = +6.0
  WORLD_X_MIN   = -6.0
10 WORLD_Y_MAX   = +4.0
  WORLD_Y_MIN   = -4.0
```

```
   MEASURE_EXACTNESS = 2

   // Figurenbeschreibung
15 // ===================

   e[1]  = 0;  point;   fixed;          0.0,0.0;                          "hidden"
   e[2]  = K;  point;   coordSystem;    0,300,300,200,200;
   e[3]  = a;  measure; controller;     0.5,0.25,-3.0,3.0,100,"a = ","";
20 e[4]  = b;  measure; controller;     -1.0,0.25,-3.0,3.0,100,"b = ","";
   e[5]  = c;  measure; controller;     -2.0,0.25,-3.0,3.0,100,"c = ","";
   e[6]  = p;  line;    curve;          "t","calculate(a)*t^2+(calculate(b)*t)+(calculate(c))",-6.0,6.0,50;
   e[7]  = P;  point;   dragable;       -0.6,-1.3,p;
   e[8]  = f'; measure; calculate;      "2*calculate(a)*coordinateX(P)+(calculate(b))",0.5,-4.5,"f'(x) = ","";
25 e[9]  = P'; point;   functionDepend; "coordinateX(P)+1","coordinateY(P)+calculate(f')";
   e[10] = t;  line;    straightLine;   P,P';
   e[11] = P'';point;   functionDepend; "coordinateX(P)+1.0","coordinateY(P)"; "hidden"
   e[12] = f'(x);line;  connect;        P',P'';                              black;0;gray;0
   e[13] = 1;  line;    connect;        P'',P;                               black;0;gray;0
30 e[14] = m0; measure; coordinates;    P,-5.5,2.0,"P = ","";

   // Textfenster
   // ===========

35 <Textbox>
    Position = 20;20;180;-1
     y = ax^2 + bx + c
        = {"calculate(a)"} x^2 + {"calculate(b)"} x + {"calculate(c)"}
   </Textbox>
40
   <Textbox>
    Position = 260;20;-1;-1
     Extremwert
   </Textbox>
45
   // Ein- und Ausblenden von Objekten
   // ================================

   hidden[1] = "if (abs(calculate(f')) > 0.01) hide (Textbox_2)"
```

Vergleicht man das obige Skript mit dem Skript auf Seite 222, so fällt auf, daß es neben den beiden Abschnitten Figurenbeschreibung und Textfenster noch zwei neue Abschnitte gibt: Systemvariablen und Ein- und Ausblenden von Objekten.

Systemvariablen Im Abschnitt Systemvariablen können sämtliche Variablen aus der Layout-Vorlage mit neuen Werten überschrieben werden. Diese sind dann nur für das aktuelle Skript gültig. Dieses ist sinnvoll, wenn nur einige wenige Werte abgeändert werden sollen. In den Zeilen 8-11 wird ein neues Weltkoordinatensystem definiert. In der Zeile 12 wird die Anzeigegenauigkeit von numerischen Werten auf zwei Nachkommastellen gesetzt.

Figurendefinition Die Figurendefinition erfolgt in den Zeilen 17-30. Die Objekte `e[1]` und `e[2]` definieren ein Koordinatensystem, das auf der Zeichenfläche angezeigt wird. In den Zeilen 19-21 werden drei Schieberegler für die Parameter a, b, c erzeugt. *Geometria* ordnet dabei automatisch alle Interaktionselemente unterhalb der Zeichenfläche an. Die Parabel wird in Form einer parametrisierten Kurve in Zeile 22 erzeugt. Dabei wird mit dem `calculate`-Befehl auf die Werte der Schieberegler zugegriffen. Die Zeile 23 enthält die Definition eines ziehbaren Punkts P auf der Kurve. Die darauf folgende Zeile berechnet den Wert der Ableitung zu dem Punkt P. Die Tangente durch P an die Parabel wird mit Hilfe zweier Stützpunkte in den Zeilen 25-29 erzeugt. Das Objekt `e[14]` ist ein

Funktional, das die Koordinaten von P berechnet und auf der Zeichenfläche anzeigt.

Textfenster Die Definition der Textfenster erfolgt, wie bereits auf der Seite 223 beschrieben. Neu ist die Angabe des Werts `-1` für die Breite oder Höhe des Textfensters. Hierdurch wird veranlaßt, daß die Größe automatisch berechnet wird (Zeile 36 und 42). Um den Wert von Funktionalen in einem Textfenster anzuzeigen, muß ein entsprechender Term in geschweiften Klammern definiert sein (Zeile 37).

Ein- und Ausblenden von Objekten Mit Hilfe des `hidden`-Befehls kann *Geometria* veranlaßt werden, beim Eintreten bestimmter Figurenzustände Objekte ein- und auszublenden. Dazu wird in dem Beispielskript in der Zeile 49 die Bedingung `abs(calculate(f')) > 0.01` angegeben. Ist diese erfüllt, so wird das aufgeführte Objekt (das zweite Textfenster) ausgeblendet.

2.2.2 Layout-Vorlage

Als Layout-Vorlage wurde die bereits in Abschnitt 5.4.6 auf Seite 223 beschriebene ".style"-Datei verwendet.

2.2.3 HTML-Dokument

Das HTML-Dokument, durch das Skript und Layout-Vorlage eingebunden werden, ist identisch mit dem Beispiel auf Seite 225. Lediglich der Name des Skripts (Zeile 7) ist geändert und die Höhe des Applet-Fensters wurde vergrößert (Zeile 5). Letzteres ist erforderlich, weil die in dem Skript definierten Interaktionselemente (Schieberegler) unterhalb der Zeichenfläche angeordnet werden. Würde man das Applet-Fenster nicht vergrößern, könnte die Zeichenfläche nicht vollständig angezeigt werden.

```
   <APPLET
      code     = "Geometria"
      codebase = ""
      archive  = "Geometria.jar"
 5    height   = "400"
      width    = "480" >
      <PARAM name = "script"      value = "Parabel.script">
      <PARAM name = "style"       value = "Demo.style">
      <PARAM name = "startButton" value = "0">
10 </APPLET>
```

2.3 Drehstreckung

Der folgende Lernbaustein enthält zum abbildungsgeometrischen Thema "Drehstreckung eines Dreiecks" eine Aufgabe mit Antwortanalyse. Im Unterschied zu den beiden vorigen Beispielen wird der Lernbaustein jedoch nicht innerhalb der Web-Seite angezeigt. Dort befindet sich nur ein Button mit der Beschriftung "Start". Betätigt man den Button, öffnet sich ein neues Fenster, in dem der Lernbaustein dargestellt wird (Abbildungen 36 und 37).

Abbildung 36: Web-Seite mit Start-Button.

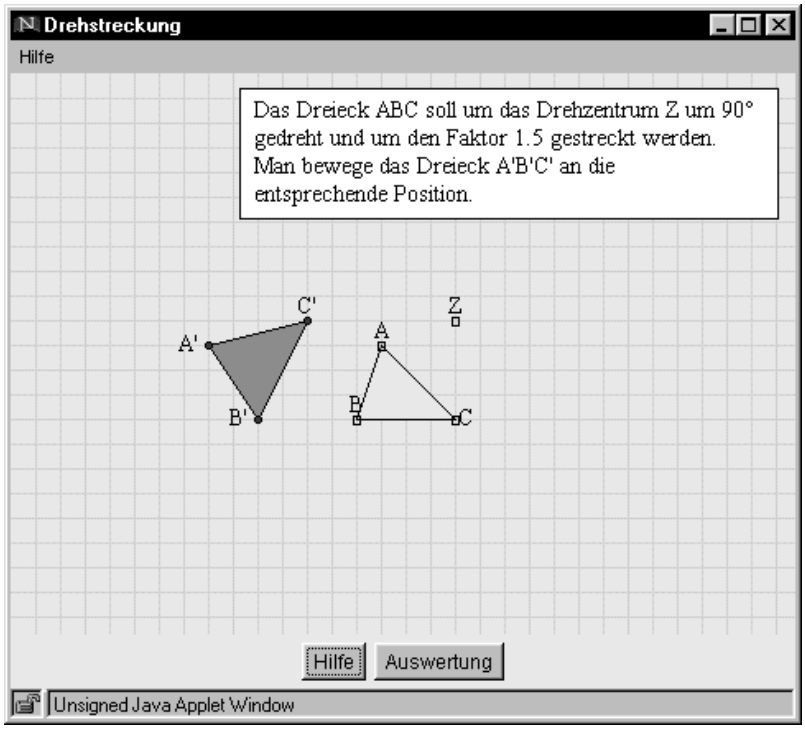

Abbildung 37: Lernbaustein im separaten Fenster.

2.3.1 Skript

Das folgende Skript zeigt, wie der Lernbaustein definiert ist. Aus Platzgründen habe ich einige Zeilenumbrüche eingefügt, die in dem ursprünglichen Skript nicht vorhanden sind. Die Zeilen 66-69, 74-77, 82-85, 90-94, 98-101, 106-109, 114-117 und 122-124 müssen zu jeweils einer Zeile zusammengefaßt werden, damit sie vom Parser korrekt interpretiert werden können.

```
   //
   // Datei: Drehstreckung.script
   //

 5 // Figurenbeschreibung
   // ===================

   e[1]  = A;  point;   fixed;     -1.0,1.0;
   e[2]  = B;  point;   fixed;     -2.0,-2.0;
10 e[3]  = C;  point;   fixed;     2.0,-2.0;
   e[4]  = Z;  point;   fixed;     2.0,2.0;
   e[5]  = P1; polygon; polygon;   A,B,C;                0;0;black;yellow
   e[6]  = A'; point;   dragable;  -8.0,1.0;
   e[7]  = B'; point;   dragable;  -6.0,-2.0;
15 e[8]  = C'; point;   dragable;  -4.0,2.0;
   e[9]  = P2; polygon; polygon;   A',B',C';             0;0;black;green
   e[10] = A1; point;   rotation;  A,Z,-1.570796327,1.0; "hidden"
   e[11] = B1; point;   rotation;  B,Z,-1.570796327,1.0; "hidden"
   e[12] = C1; point;   rotation;  C,Z,-1.570796327,1.0; "hidden"
20 e[13] = A2; point;   rotation;  A,Z,-1.570796327,1.5; "hidden"
   e[14] = B2; point;   rotation;  B,Z,-1.570796327,1.5; "hidden"
```

```
   e[15] = C2; point;   rotation;  C,Z,-1.570796327,1.5; "hidden"
   e[16] = b1; measure; button;    "Hilfe","help";
   e[17] = b2; measure; button;    "Auswertung","evaluate";

   // Textfenster
   // ===========

   <Textbox>
    Position = 140;10;-1;-1
     Das Dreieck ABC soll um das Drehzentrum Z um 90°
     gedreht und um den Faktor 1.5 gestreckt werden.
     Man bewege das Dreieck A'B'C' an die
     entsprechende Position.
   </Textbox>

   // Hilfen
   // ======

   <Help>
     Führe zuerst mit dem Dreieck A'B'C'
     die Drehung durch und dann die
     Streckung.
   </Help>

   // Antwortanaylse
   // ==============

   <Problem>
    MAX_ANSWER = 0
    condition[1] = "isIncident(A2,A')"
    condition[2] = "isIncident(B2,B')"
    condition[3] = "isIncident(C2,C')"
    condition[4] = "isIncident(A1,A')"
    condition[5] = "isIncident(B1,B')"
    condition[6] = "isIncident(C1,C')"
   </Problem>

   <Answer 1>
    key = "condition[1] AND condition[2] AND condition[3]"
    comment[1] = "Richtig. /n /nDie Drehstreckung wurde korrekt durchgeführt."
   </Answer 1>

   <Answer 2>
    key = "condition[4] AND condition[5] AND condition[6]"
    comment[1] = "Ihre Antwort ist nicht richtig. /n
                  Die Drehung wurde zwar korrekt durchgeführt, /n
                  es fehlt aber noch die Streckung./n
                  Versuchen Sie es noch einmal."
   </Answer 2>

   <Answer 3>
    key = "NOT(condition[1]) AND condition[2] AND condition[3]"
    comment[1] = "Ihre Antwort ist teilweise richtig. /n
                  Die Bildpunkte B' und C' stimmen, /n
                  aber A' ist noch nicht korrekt plaziert. /n
                  Versuchen Sie es noch einmal."
   </Answer 3>

   <Answer 4>
    key = "NOT(condition[2]) AND condition[1] AND condition[3]"
    comment[1] = "Ihre Antwort ist teilweise richtig. /n
                  Die Bildpunkte A' und C' stimmen, /n
                  aber B' ist noch nicht korrekt plaziert. /n
                  Versuchen Sie es noch einmal."
   </Answer 4>

   <Answer 5>
    key = "NOT(condition[3]) AND condition[2] AND condition[1]"
    comment[1] = "Ihre Antwort ist teilweise richtig. /n
                  Die Bildpunkte A' und B' stimmen, /n
                  aber C' ist noch nicht korrekt plaziert. /n
```

```
                Versuchen Sie es noch einmal."
   </Answer 5>
95
   <Answer 6>
    key = "NOT(condition[1]) AND NOT(condition[2]) AND condition[3]"
    comment[1] = "Ihre Antwort ist teilweise richtig. /n
                Die Bildpunkte A' und B' stimmen nicht, /n
100             nur C' ist korrekt plaziert. /n
                Versuchen Sie es noch einmal."
   </Answer 6>

   <Answer 7>
105  key = "NOT(condition[3]) AND NOT(condition[2]) AND condition[1]"
    comment[1] = "Ihre Antwort ist teilweise richtig. /n
                Die Bildpunkte B' und C' stimmen nicht, /n
                nur A' ist korrekt plaziert. /n
                Versuchen Sie es noch einmal."
110 </Answer 7>

   <Answer 8>
    key = "NOT(condition[1]) AND NOT(condition[3]) AND condition[2]"
    comment[1] = "Ihre Antwort ist teilweise richtig. /n
115             Die Bildpunkte A' und C' stimmen nicht, /n
                nur B' ist korrekt plaziert. /n
                Versuchen Sie es noch einmal."
   </Answer 8>

120 <Answer 9>
    key = "1"
    comment[1] = "Ihre Antwort ist nicht richtig. /n
                Die Bildpunkte A', B' und C' sind nicht korrekt plaziert. /n
                Versuchen Sie es noch einmal."
125 </Answer 9>
```

Das Skript ist unterteilt in die Abschnitte: Figurenbeschreibung, Textfenster, Hilfen und Antwortanalyse.

Figurenbeschreibung Die Definition der Figur erfolgt in den Zeilen 8-24. Bei den Objekten `e[1]-e[4]` handelt es sich um Punkte mit festen Koordinaten, die nicht verschoben werden können. Die Punkte `A'`, `B'` und `C'` sind dagegen in der Zeichenfläche frei ziehbar (Zeile 13-15). In den Zeilen 12 und 16 wird jeweils ein Polygon definiert, bei dem die Farbdefinition aus der Layout-Vorlage mit neuen Werten überschrieben wird. Durch die Angabe von `0;0;black;yellow` werden die Beschriftung und die Eckpunkte nicht angezeigt. Die Linienfarbe wird auf schwarz gesetzt und die Flächenfüllfarbe ist gelb. In den Zeilen 17-22 werden Bildpunkte einer Drehstreckung definiert. Durch den Befehl `"hidden"` sind diese jedoch auf der Zeichenfläche nicht sichtbar. Sie dienen bei der Antwortanalyse dazu, richtige und falsche Antworten zu erkennen. Bei den letzten beiden Objekten (Zeile 23-24) handelt es sich um zwei Buttons, die mit "Hilfe" und "Auswertung" beschriftet sind. Die Befehlswörter `evaluate` und `help` sorgen dafür, daß nach dem Anklicken die Antwortanalyse gestartet oder der Hilfetext angezeigt wird.

Textfenster Ein Textfenster wird durch die Zeilen 29-35 definiert. Der Wert `-1` für die Definition der Fensterbreite und -höhe bewirkt eine automatische Berechnung der Fenstergröße.

Hilfen Hilfetexte werden ähnlich definiert wie Textfenster. Eine Positionsangabe ist jedoch nicht erforderlich, weil der Hilfetext in einem separaten Fenster

angezeigt wird. Bei mehr als einer Hilfe werden diese in der definierten Reihenfolge nacheinander angezeigt. In dem obigen Skript ist die Hilfe in den Zeilen 40-44 festgelegt.

Antwortanalyse Durch eine Antwortanalyse soll die Antwort des Schülers (der aktuelle Figurenzustand) bewertet werden. Dazu wird als erstes festgelegt, wie oft die Antwortanalyse aufgerufen werden kann (Zeile 50). Der Wert `0` bedeutet vereinbarungsgemäß, daß es unbegrenzt viele Versuche gibt. In den Zeilen 51-56 werden nun Prüffunktionen (Aussagen über den Figurenzustand) formuliert, die wahr oder falsch sein können. Sie werden im folgenden bei der Definition der einzelnen Antwortwerte verwendet, um sogenannte Prüfschlüssel zu beschreiben. Ein Prüfschlüssel `key` besteht aus einer booleschen Verknüpfung der definierten Prüffunktionen. Jeder Prüfschlüssel gehört zu einem Antwortwert, der durch die Befehlswörter `<Answer i>` und `</Answer i>` umklammert ist. Zu jedem Antwortwert wird noch mindestens ein Antwortkommentar definiert, der dem Schüler angezeigt wird, sobald der zugehörige Prüfschlüssel den Wert 1 annimmt.

Wird die Antwortanalyse durch Betätigen des Buttons "Auswertung" ausgelöst, so wird der Reihe nach geprüft, welcher der definierten Prüfschlüssel zutrifft. Der erste Antwortwert muß deshalb die richtige Lösung definieren (Zeile 59-62). Der letzte Antwortwert (Zeile 120-125) enthält als Prüfschlüssel nur den Wert `1`, d. h. dieser Schlüssel paßt immer und verhindert, daß es zu einer Antwort keinen Antwortwert gibt.

Die Antwortkommentare müssen im Unterschied zu der obigen Darstellung jeweils in einer Zeile stehen. Durch das Zeichen `/n` erreicht man einen Zeilenumbruch bei der Ausgabe. Pro Antwortwert kann man beliebig viele Antwortkommentare definieren. Diese müssen dann mit `comment[1]`, `comment[2]`, `comment[3]`, ... unterschieden werden. Dies ist dann sinnvoll, wenn die Anzahl von Antwortversuchen begrenzt worden ist.

2.3.2 Layout-Vorlage

Als Layout-Vorlage wird bei diesem Lernbaustein die Datei "Demo2.style" verwendet. Sie unterscheidet sich zu der auf Seite 223 beschriebenen in den folgenden Punkten:

Die Fenstergröße `APPLET_WIDTH` und `APPLET_HEIGHT` ist auf 640 × 480 Bildschirmpunkte vergrößert worden. Die Variable `USESEPARATEWINDOW` wurde auf `TRUE` gesetzt, dadurch wird der Lernbaustein in einem eigenen Fenster angezeigt. Durch das Setzen der booleschen Variablen `SNAPTOGRID` und `SHOWGRID` wird ein Hintergrundraster in der Zeichenfläche angezeigt und der Rasterfangmodus eingeschaltet.

2.3.3 HTML-Dokument

Das HTML-Dokument weist im Unterschied zu den beiden vorherigen Beispielen einige Änderungen auf. Die Appletgröße wurde auf 40 × 60 Bildschirmpunkte verkleinert, da sie nur den Start-Button enthalten soll (Zeile 5-6). Um diesen zu erzeugen, wird der Wert des Parameters `startButton` auf `1` gesetzt (Zeile 9). Sobald der Button ausgelöst wird und die Variable `USESEPARATEWINDOW` = `TRUE`

gesetzt ist, öffnet sich ein neues Fenster mit dem Lernbaustein. Der Applet-Hintergrund kann durch den Parameter `backgroundColor` verändert werden. Hierzu ist ein Farbwert im RGB-Schema angegeben (Zeile 10).

```
   <APPLET
      code     = "Geometria"
      codebase = ""
      archive  = "Geometria.jar"
 5    height   = "40"
      width    = "60" >
      <PARAM name = "script"          value = "Drehstreckung.script">
      <PARAM name = "style"           value = "Demo2.style">
      <PARAM name = "startButton"     value = "1">
10    <PARAM name = "backgroundColor" value = "0,64,128">
   </APPLET>
```

3 Praktische Figurenerstellung

Im folgenden möchte ich einige Hinweise zur praktischen Figurenerstellung geben.

3.1 Arbeiten mit Browsern

Während die Skripte, Layout-Vorlagen und HTML-Dokumente mit jedem beliebigen Texteditor erstellt oder abgeändert werden können, ist zum Testen ein Java-Interpreter erforderlich. Ein solcher Interpreter ist beispielsweise der Java-AppletViewer, der Netscape Communicator oder der Internet Explorer.

3.1.1 AppletViewer

Der AppletViewer ist Bestandteil des Java-Development-Kits und eignet sich gut zum Testen eigener Skripte. Unter Windows wird das Programm im DOS-Fenster gestartet. Dazu ist die Befehlszeile `appletviewer MyDocument.htm` einzugeben. Alle auftretenden Fehlermeldungen werden ebenfalls in das DOS-Fenster ausgegeben. Der besondere Vorteil des AppletViewers liegt darin, daß er einem lokal gestarteten Applet erlaubt, von der Festplatte zu lesen und geänderte Dateien zu speichern. Startet man einen Lernbaustein in einem separaten Fenster, so sind über die Menüleiste einige Funktionen verfügbar, die das Entwickeln oder Optimieren eines Lernbausteins erleichtern können:

- Durch den Menübefehl "Datei – Öffnen" können durch Auswahl in einem Öffnen-Dialogfenster beliebige Skripte von der Festplatte gestartet werden.
- Durch den Menübefehl "Figur – Neu laden" kann ein bereits geladenes Skript erneut eingelesen und interpretiert werden. Dieses ist zweckmäßig, wenn man Änderungen im Skript vornehmen und direkt danach das Resultat sehen möchte.
- Durch den Menübefehl "Bearbeiten - Position speichern" werden in dem Skript alle Anfangskoordinaten ziehbarer Punkte durch die aktuellen Koordinaten ersetzt. Die Figurenerstellung wird dadurch erleichtert, weil man beim ersten Entwurf eines Skripts, die Anfangskoordinaten erst einmal beliebig wählt. Die exakte Lage läßt sich anschließend im Zugmodus besser festlegen.

- Bei der Skripterstellung kommt es häufiger vor, daß man in die `e[]`-Liste der geometrischen Objekte neue Zeilen einfügen oder vorhandene löschen möchte. Anschließend muß die fortlaufende Numerierung wieder hergestellt werden. Dies kann durch die Funktion "Bearbeiten - Prüfe e[]-Liste" automatisiert werden.

3.1.2 Netscape Communicator

Um mit dem Netscape Communicator einen Lernbaustein zu testen, muß man lediglich die entsprechende Web-Seite aufrufen. Während der Testphase sollte man darauf achten, daß der Cache abgeschaltet ist. Dazu öffnet man am besten unter "Edit – Preferences – Advanced – Cache" das Dialogfenster und setzt die Cache-Größe auf 0. Eventuell ist es sinnvoll, im Profile-Manager ein Profil etwa mit den Namen "Geometria" anzulegen und nur dort den Cache abzuschalten.

Alle Fehlermeldungen von *Geometria* werden beim Netscape Communicator in eine Java-Console ausgegeben. Diese wird durch Anwählen des Menü-Befehls "Communicator – Tools – Java-Console" geöffnet.

3.1.3 Internet Explorer

Um mit dem Internet Explorer einen Lernbaustein zu prüfen, braucht man ebenfalls nur das entsprechende HTML-Dokument zu starten. Während der Testphase sollte man darauf achten, daß der Cache abgeschaltet ist. Dazu öffnet man das Dialogfenster unter "Ansicht – Optionen – Erweitert – Einstellungen" und führt die Änderungen durch.

Alle Fehlermeldungen von *Geometria* werden beim Internet Explorer in eine Datei "javalog.txt" geschrieben, die sich im Verzeichnis "c:\windows\java\" befindet.

3.2 Allgemeine Hinweise

In diesem Abschnitt sollen einige Hinweise gegeben werden, die beim praktischen Erstellen von Lernbausteinen nützlich sind.

- Die Fehlersuche in einem Skript wird durch Einschalten des Menü-Befehls "Einstellungen – Fehlersuche" erleichtert. Dadurch werden alle (sonst unterdrückten) Fehlermeldungen ausgegeben. Außerdem wird protokolliert, welche Objekte erzeugt werden. Man sieht dadurch sofort, bei welchem Objekt der Parser steckenbleibt.

- Durch einen doppelten Mausklick auf die Zeichenfläche werden die aktuellen Mauszeigerkoordinaten des Weltkoordinatensystems in die Zwischenablage kopiert. Diese können dann in einem Skript eingefügt werden. Dadurch wird das Festlegen der Anfangsposition der Figur auf der Zeichenfläche etwas erleichtert.

- Einigen Browsern bereiten beim Zugriff über das Internet Leerzeichen oder Sonderzeichen in Dateinamen Probleme. Verzichten Sie am besten bei der Bezeichnung von Skript und Layout-Vorlage darauf.

Anhang B
Konstruktionsreferenz

Inhaltsverzeichnis

1 Geometrische Objekte

1.1 Ziehbare Punkte

1.1.1 Punkt in Zeichenfläche

Ein Punkt, der frei innerhalb der gesamten Zeichenfläche gezogen werden kann, wird durch die Unterklassen `Dragable` oder `Free` realisiert. Als Objektdaten müssen die beiden Anfangskoordinaten angegeben werden.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`point`	`free`	<x-Koordinate [d]>, <y-Koordinate [d]>
<Name [s]>	`point`	`dragable`	<x-Koordinate [d]>, <y-Koordinate [d]>

Beispiel:

```
e[1] = A;   point;   free;       3.0, 3.0;
e[2] = B;   point;   dragable;   0.0, 0.0;
```

1.1.2 Punkt in Punktmenge

Ein Punkt, der innerhalb einer Punktmenge gezogen werden kann, wird durch die Unterklasse `Dragable` realisiert. Als Objektdaten müssen die beiden Anfangskoordinaten und ein Punktmengenobjekt angegeben werden.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`point`	`dragable`	<x-Koordinate [d]>, <y-Koordinate [d]>, <Punktmenge [PS]>

Beispiel:

```
e[1] = Pset; line;   pointSet;   "Kurve01.gif", 10000, 10;
e[2] = A;    point;  dragable;   2.7,5.4, Pset;
```

1.1.3 Punkt in Kreis- oder Polygonfläche

Ein Punkt, der innerhalb der Fläche eines Kreises oder eines Polygons gezogen werden kann, wird durch die Unterklasse `Dragable` realisiert. Als Objektdaten müssen die beiden Anfangskoordinaten, ein Kreis- oder Polygonobjekt, das als Bezugsobjekt dient, sowie der Zusatz `"area"` angegeben werden.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`point`	`dragable`	<x-Koordinate [d]>, <y-Koordinate [d]>, <Kreis [C]>, `"area"`
<Name [s]>	`point`	`dragable`	<x-Koordinate [d]>, <y-Koordinate [d]>, <Polygon [PG]>, `"area"`

Beispiel:

```
e[1] = P1;   point;    dragable;      6.0,-1.0;
e[2] = P2;   point;    dragable;      8.0,2.0;
e[3] = k1;   circle;   radius;        P1,P2;
e[4] = P3;   point;    dragable;      2.0,-1.0,k1;
e[5] = P4;   point;    dragable;      6.0,-2.0,k1,"area";
e[6] = P5;   point;    dragable;      0.0,0.0;
e[7] = P6;   point;    dragable;      -8.0,2.0;
e[8] = P7;   point;    dragable;      -10.0,-7.0;
e[9] = P8;   point;    dragable;      0.0,-7.0;
e[10] = p;   polygon;  polygon;       P5,P6,P7,P8;
e[11] = P9;  point;    dragable;      -3.0,-2.0,p,"area";
```

1.1.4 Punkt auf Objekt

Mit "Punkt auf Objekt" wird ein ziehbarer Punkt bezeichnet, der entlang der Bahn von eindimensionalen Objekten gezogen werden kann. Zu diesen Objekten zählen: Strecken, Strahlen, Geraden, Polygone, Kreise, Kreisbögen, Kurven und Ortslinien. Um einen solchen Punkt erzeugen zu können, müssen als Objektdaten die beiden Anfangskoordinaten und das Bezugsobjekt angegeben werden.

Beispiel:

```
e[1] = P1;    point;    dragable;      -9.9,8.9;
e[2] = P2;    point;    dragable;       7.0,9.0;
e[3] = s1;    line;     straightLine;   P1,P2;
e[4] = P3;    point;    dragable;      -5.0,4.0,s1;
```

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`point`	`dragable`	<x-Koordinate [d]>, <y-Koordinate [d]>, <Strecke [L]>
<Name [s]>	`point`	`dragable`	<x-Koordinate [d]>, <y-Koordinate [d]>, <Strahl [R]>
<Name [s]>	`point`	`dragable`	<x-Koordinate [d]>, <y-Koordinate [d]>, <Gerade [ST]>
<Name [s]>	`point`	`dragable`	<x-Koordinate [d]>, <y-Koordinate [d]>, <Kreis [C]>
<Name [s]>	`point`	`dragable`	<x-Koordinate [d]>, <y-Koordinate [d]>, <Kreisbogen [SE]>
<Name [s]>	`point`	`dragable`	<x-Koordinate [d]>, <y-Koordinate [d]>, <Polygon [PG]>
<Name [s]>	`point`	`dragable`	<x-Koordinate [d]>, <y-Koordinate [d]>, <Kurve [CU]>
<Name [s]>	`point`	`dragable`	<x-Koordinate [d]>, <y-Koordinate [d]>, <Ortslinie [LO]>

1.1.5 Horizontal ziehbarer Punkt

Ein Punkt, der nur horizontal ziehbar ist, wird durch die Unterklasse `Horizontal` realisiert. Dazu muß ein Punkt P angegeben werden, von dem der ziehbare Punkt abhängig ist und der den Wert der y-Koordinate bestimmt. Außerdem ist ein Initialisierungswert für die x-Koordinate festzulegen.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`point`	`horizontal`	<Punkt [P]>, <x-Koordinate [d]>

Beispiel:

```
e[1] = A;    point;    dragable;    0.0,-5.0;
e[2] = H;    point;    horizontal;  A,3.0;
```

1.1.6 Vertikal ziehbarer Punkt

Ein Punkt, der nur vertikal ziehbar ist, wird durch die Unterklasse `Vertical` realisiert. Dazu muß ein Punkt P angegeben werden, von dem der ziehbare Punkt abhängig ist und der den Wert der x-Koordinate bestimmt. Außerdem ist ein Initialisierungswert für die y-Koordinate festzulegen.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`point`	`vertical`	<Punkt [P]>, <y-Koordinate [d]>

Beispiel:

```
e[1] = A;    point;    dragable;    0.0,-5.0;
e[2] = V;    point;    vertical;    A,3.0;
```

1.2 Nicht-ziehbare Punkte

1.2.1 Fixpunkt

Ein Fixpunkt besitzt eine feste Lage in der Zeichenfläche und kann nicht verschoben werden. Als Objektdaten sind zwei Koordinatenwerte anzugeben.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`point`	`fixed`	<x-Koordinate [d]>, <y-Koordinate [d]>

Beispiel:

```
e[1] = A;  point;  fixed;        1.0,1.0;
e[2] = B;  point;  fixed;        0.0,0.0;
```

1.2.2 Mittelpunkt

Den Mittelpunkt zwischen zwei Punkten erhält man durch die Unterklasse `Midpoint`. Als Objektdaten müssen zwei Punkte übergeben werden.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`point`	`midpoint`	<Punkt [P]>, <Punkt [P]>

Beispiel:

```
e[1] = A;   point;   dragable;  1.0,1.0;
e[2] = B;   point;   dragable;  0.0,0.0;
e[3] = M;   point;   midpoint;  A,B;
```

1.2.3 Schnittpunkt

Schnittpunkte zwischen Strecken, Strahlen, Geraden und Kreisen werden durch die Klasse `Intersection` erzeugt. Beim Schnitt eines Kreises mit einem zweiten Objekt, werden die Schnittpunkte durch Angabe der Werte 1 oder 2 unterschieden. Hier gibt es außerdem noch einen softwaretechnisch besonderen Fall: Wenn in einer speziellen Figurenkonstruktion bereits ein Schnittpunkt durch einen ziehbaren Punkt, der z. B. den Radius bestimmt, festgelegt ist, dann bewirkt die Angabe des Werts -1, daß der neu zu erzeugende Schnittpunkt nicht mit dem vorhandenen zusammenfällt.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`point`	`intersection`	<Strecke, Strahl oder Gerade [L]>, <Strecke, Strahl oder Gerade [L]>
<Name [s]>	`point`	`intersection`	<Strecke, Strahl oder Gerade [L]>, <Kreis [C]>, <1. oder 2. Lsg. [i]>
<Name [s]>	`point`	`intersection`	<Kreis [C]>, <Kreis [C]>, <1. oder 2. Lsg. [i]>

Beispiel:

```
e[1] = A;    point;    dragable;      1.0,1.0;
e[2] = B;    point;    dragable;      0.0,0.0;
e[3] = C;    point;    dragable;      8.0,8.0;
e[4] = D;    point;    dragable;      5.0,-3.0;
e[5] = E;    point;    dragable;      0.0,3.0;
```

```
e[6] = g;    line;     straightLine;  A,B;
e[7] = k;    circle;   circumcircle;  C,D,E;
e[8] = I1;    point;   intersection;  g,k,1;
e[9] = I2;    point;   intersection;  g,k,2;
```

1.2.4 Fußpunkt

Zu einem Punkt A und einer Geraden BC kann mit Hilfe der Unterklasse `Foot` ein Fußpunkt D erzeugt werden, so daß die Gerade DA orthogonal zu BC ist.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`point`	`foot`	<Punkt A [P]>, <Gerade BC [L]>
<Name [s]>	`point`	`foot`	<Punkt A [P]>, <Punkt B [P]>, <Punkt C [P]>

Beispiel:

```
e[1] = A;   point;   dragable;     1.0,3.0;
e[2] = B;   point;   dragable;     0.0,4.0;
e[3] = C;   point;   dragable;     2.0,3.0;
e[4] = g;   line;    straightLine; B,C;
e[5] = D;   point;   foot;         A,g;
```

1.2.5 Strecke abtragen

Mit Hilfe der beiden Unterklassen `Cutoff` und `Extend` lassen sich Strecken abtragen. Im ersten Fall werden die beiden Strecken AB und CD betrachtet. Dazu wird ein Punkt E erzeugt, indem die Strecke CD von A aus in Richtung auf B abgetragen wird, so daß $|AE| = |CD|$. Im zweiten Fall werden wiederum zwei Strecken AB und CD betrachtet. Dazu wird ein Punkt E erzeugt, indem die Strecke CD von B aus in die zu A entgegengesetzte Richtung abgetragen wird, so daß $|BE| = |CD|$.
Beispiel:

```
e[1] = A;   point;   dragable;     1.0,3.0;
e[2] = B;   point;   dragable;     0.0,4.0;
e[3] = C;   point;   dragable;     2.0,3.0;
e[4] = D;   point;   dragable;     1.0,1.0;
e[5] = E;   point;   cutoff;       A,B,C,D;
e[6] = F;   point;   extend;       A,B,C,D;
```

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`point`	`cutoff`	<Punkt A [P]>, <Punkt B [P]>, <Punkt C [P]>, <Punkt D [P]>
<Name [s]>	`point`	`cutoff`	<Strecke AB [L]>, <Strecke CD [L]>
<Name [s]>	`point`	`extend`	<Punkt A [P]>, <Punkt B [P]>, <Punkt C [P]>, <Punkt D [P]>
<Name [s]>	`point`	`extend`	<Strecke AB [L]>, <Strecke CD [L]>

1.2.6 Punkt im speziellen Streckenverhältnis

Mit Hilfe der beiden Unterklassen `Proportion` und `MeanProportional` lassen sich spezielle Streckenverhältnisse abtragen. Im ersten Fall werden die vier Strecken AB, CD, EF und GH betrachtet. Dazu wird ein Punkt I erzeugt, der auf der Strecke GH liegt und für den gilt:

$$\frac{AB}{CD} = \frac{EF}{GI}.$$

Im zweiten Fall werden die drei Strecken AB, CD und EF betrachtet. Es wird ein Punkt G erzeugt, der auf der Strecke EF liegt und für den gilt:

$$\frac{AB}{CD} = \frac{CD}{EG}.$$

Beispiel:

```
e[1] = A;   point;   dragable;          1.0,3.0;
e[2] = B;   point;   dragable;          0.0,4.0;
e[3] = C;   point;   dragable;          2.0,3.0;
e[4] = D;   point;   dragable;          1.0,1.0;
e[5] = E;   point;   dragable;          3.0,2.0;
e[6] = F;   point;   dragable;          0.0,0.0;
e[7] = G;   point;   meanProportional;  A,B,C,D,E,F;
```

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`point`	`proportion`	<Punkt A [P]>, <Punkt B [P]>, <Punkt C [P]>, <Punkt D [P]>, <Punkt E [P]>, <Punkt F [P]>, <Punkt G [P]>, <Punkt H [P]>
<Name [s]>	`point`	`proportion`	<Strecke AB [L]>, <Strecke CD [L]>, <Strecke EF [L]>, <Strecke GH [L]>
<Name [s]>	`point`	`meanProportional`	<Punkt A [P]>, <Punkt B [P]>, <Punkt C [P]>, <Punkt D [P]>, <Punkt E [P]>, <Punkt F [P]>
<Name [s]>	`point`	`meanProportional`	<Strecke AB [L]>, <Strecke CD [L]>, <Strecke EF [L]>

1.2.7 Bildpunkt einer Drehung oder Drehstreckung

Der Bildpunkt einer Drehung oder Drehstreckung wird durch die Unterklasse `Rotation` realisiert. Für die Drehung sind als Objektdaten ein Punkt P als Urbild und ein Punkt Z als Drehzentrum anzugeben. Der Drehwinkel φ kann durch einen konstanten Wert, durch ein veränderliches Funktional oder durch drei Punkte mit $\varphi = \angle ABC$ definiert sein. Außerdem ist ein Streckfaktor s festzulegen. Soll keine Streckung ausgeführt werden, so ist $s = 1.0$ zu setzen oder einfacher: man gibt keinen Wert für s an.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`point`	`rotation`	<Punkt P [P]>, <Punkt Z [P]>, <Drehwinkel φ [d]>
<Name [s]>	`point`	`rotation`	<Punkt P [P]>, <Punkt Z [P]>, <Drehwinkel φ [d]>, <Streckfaktor s [d]>
<Name [s]>	`point`	`rotation`	<Punkt P [P]>, <Punkt Z [P]>, <Drehwinkel φ [M]>, <Streckfaktor s [M]>
<Name [s]>	`point`	`rotation`	<Punkt P [P]>, <Punkt Z [P]>, <Punkt A [P]>, <Punkt B [P]>, <Punkt C [P]>, <Streckfaktor s [d]>

Beispiel:

```
e[1] = P;   point;   dragable;     1.0,1.0;
e[2] = Z;   point;   dragable;     0.0,0.0;
e[3] = A;   point;   dragable;     1.0,3.0;
e[4] = B;   point;   dragable;     0.0,4.0;
e[5] = C;   point;   dragable;     4.0,5.0;
e[6] = P';  point;   rotation;     P,Z,3.14;
e[7] = P''; point;   rotation;     P,Z,A,B,C;
```

1.2.8 Bildpunkt einer Geraden- oder Schubspiegelung

Der Bildpunkt einer Geraden- oder Schubspiegelung wird durch die Unterklasse `Mirror` realisiert. Für eine Geradenspiegelung ist ein Urbildpunkt P und eine Gerade g anzugeben, die als Spiegelachse dient. Für eine Schubspiegelung sind neben P und g zwei Werte für die x- und y-Translation festzulegen.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`point`	`mirror`	<Punkt P [P]>, <Gerade g [L]>
<Name [s]>	`point`	`mirror`	<Punkt P [P]>, <Gerade g [L]>, <x-Translation [d]>, <y-Translation [d]>

Beispiel:

```
e[1] = P;    point;    dragable;     1.0,1.0;
e[2] = A;    point;    dragable;     -9.0,8.0;
e[3] = B;    point;    dragable;     7.0,9.0;
e[4] = g;    line;     straightLine; A,B;
e[5] = P';   point;    mirror;       P,g;
e[6] = P'';  point;    mirror;       P,g,1.5,1.5;
```

1.2.9 Bildpunkt einer Streckung oder Punktspiegelung

Der Bildpunkt einer Streckung und einer Punktspiegelung wird durch die Unterklasse `Mirror` realisiert. Für die Streckung sind als Objektdaten ein Punkt P als Urbild, ein Punkt Z als Streckzentrum sowie ein Streckfaktor s anzugeben. Die Punktspiegelung benötigt nur die Angabe von P und Z.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`point`	`mirror`	<Punkt P [P]>, <Punkt Z [P]>
<Name [s]>	`point`	`mirror`	<Punkt P [P]>, <Punkt Z [P]>, <Streckfaktor s [d]>

Beispiel:

```
e[1] = P;   point;   dragable;     1.0,1.0;
e[2] = Z;   point;   dragable;     0.0,0.0;
e[3] = P';  point;   mirror;       P,Z;
e[4] = P''; point;   mirror;       P,Z,1.5;
```

1.2.10 Bildpunkt einer Verschiebung

Der Bildpunkt einer Verschiebung wird durch die Unterklasse `Translation` realisiert. Für die Verschiebung müssen als Objektdaten ein Urbildpunkt P und ein Verschiebevektor v angegeben werden. Der Vektor kann dabei durch zwei konstante Werte v_x und v_y oder durch zwei Punkte V_1 und V_2 definiert werden.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`point`	`translation`	<Punkt P [P]>, <v_x [d]>, <v_y [d]>
<Name [s]>	`point`	`translation`	<Punkt P [P]>, <Punkt V_1 [P]>, <Punkt V_2 [P]>

Beispiel:

```
e[1] = P;   point;   dragable;     1.0,1.0;
e[2] = A;   point;   dragable;     0.0,0.0;
e[3] = B;   point;   dragable;     4.0,0.0;
e[4] = P';  point;   translation;  P,4.0,3.0;
e[5] = P''; point;   translation;  P,A,B;
```

1.2.11 Bildpunkt einer Kreisspiegelung

Der Bildpunkt einer Kreisspiegelung wird durch die Unterklasse `Invert` realisiert. Als Objektdaten müssen ein Punkt als Urbild und ein Kreis übergeben werden.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`point`	`invert`	<Punkt [P]>, <Kreis [C]>

Beispiel:

```
e[1] = A;   point;   dragable;     1.0,1.0;
e[2] = B;   point;   dragable;     0.0,0.0;
e[3] = C;   point;   dragable;     2.0,8.0;
e[4] = k;   circle;  circumcircle; A,B,C;
e[5] = P;   point;   dragable;     4.0,0.0;
e[6] = P';  point;   invert;       P,k;
```

1.2.12 Funktionsabhängiger Punkt

Funktionsabhängige Punkte werden durch die Klasse `FunctionDepend` erzeugt. Dazu sind als Objektdaten zwei Koordinatenfunktionen anzugeben, die durch zwei Objekte der Klasse `MeasureCalculate` definiert werden. Wahlweise kann ein solcher Punkt in einem speziellen Koordinatensystem (Seite 270) angezeigt werden.
Beispiel:

```
e[1] = A;  point;  dragable;        1.0,1.0;
e[2] = B;  point;  dragable;        0.0,0.0;
```

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`point`	`functionDepend`	<Fkt. x-Koord. [M]>, <Fkt. y-Koord. [M]>
<Name [s]>	`point`	`functionDepend`	<Fkt. x-Koord. [M]>, <Fkt. y-Koord. [M]>, <Koordinatensystem [CO]>

```
e[3] = C;  point;  functionDepend;  "(coordinateX(A)+coordinateX(B))/2",
                                    "(coordinateY(A)+coordinateY(B))/2";
```

1.2.13 Winkelhalbierendenpunkt

Zu drei Punkten A, B, C kann man durch die Unterklasse `AngleBiSector` einen Punkt D erzeugen, der auf der Winkelhalbierenden des Winkels $\angle ABC$ liegt.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`point`	`angleBiSector`	<Punkt A [P]>, <Punkt B [P]>, <Punkt C [P]>

Beispiel:

```
e[1] = A;   point;   dragable;       1.0,1.0;
e[2] = B;   point;   dragable;       0.0,0.0;
e[3] = C;   point;   dragable;       4.0,0.0;
e[4] = D;   point;   angleBiSector;  A,B,C;
```

1.2.14 Winkelteilendenpunkt

Zu drei Punkten A, B, C kann man durch die Unterklasse `AngleDivider` einen Punkt D erzeugen, der auf der Geraden liegt, die den Winkel $\angle ABC$ im Verhältnis $1 : n$ teilt.
Beispiel:

```
e[1] = A;   point;   dragable;       1.0,1.0;
e[2] = B;   point;   dragable;       0.0,0.0;
e[3] = C;   point;   dragable;       4.0,0.0;
e[4] = D;   point;   angleDivider;   A,B,C,3;
```

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`point`	`angleDivider`	<Punkt A [P]>, <Punkt B [P]>, <Punkt C [P]>, <n [i]>
<Name [s]>	`point`	`angleDivider`	<Punkt A [P]>, <Punkt B [P]>, <Punkt C [P]>, <n [d]>

1.2.15 Vierter Parallelogrammpunkt

Zu drei Punkten A, B, C kann man mit Hilfe der Unterklasse `Parallelogram` einen Punkt D erzeugen, so daß $ABCD$ ein Parallelogramm ist.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`point`	`parallelogram`	<Punkt A [P]>, <Punkt B [P]>, <Punkt C [P]>

Beispiel:

```
e[1] = A;   point;   dragable;       1.0,1.0;
e[2] = B;   point;   dragable;       0.0,0.0;
e[3] = C;   point;   dragable;       4.0,0.0;
e[4] = D;   point;   parallelogram;  A,B,C;
```

1.2.16 Mittelpunkt eines Kreises

Der Mittelpunkt eines Kreises wird durch die beiden Unterklassen `Center` und `CircumCenter` realisiert. Als Objektdaten müssen entweder ein Kreis oder drei Punkte übergeben werden.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`point`	`center`	<Kreis [C]>
<Name [s]>	`point`	`circumCenter`	<Punkt [P]>, <Punkt [P]>, <Punkt [P]>,

Beispiel:

```
e[1] = A;   point;   dragable;     1.0,1.0;
e[2] = B;   point;   dragable;     0.0,0.0;
```

```
e[3] = C;   point;   dragable;     2.0,8.0;
e[4] = k;   circle;  circumcircle; A,B,C;
e[5] = M;   point;   center;       k;
```

1.2.17 Eckpunkt eines Polygons

Zugriff auf den n-ten Eckpunkt eines Polygons erhält man durch die Unterklasse `Vertex`. Als Objektdaten müssen ein Polygon und der Wert n übergeben werden.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`point`	`vertex`	<Polygon [PG]>, <n [i]>

Beispiel:

```
e[1] = A;   point;   dragable;     1.0,1.0;
e[2] = B;   point;   dragable;     0.0,0.0;
e[3] = p;   polygon; square;       A,B;
e[4] = C;   point;   vertex;       p,3;
e[5] = D;   point;   vertex;       p,4;
```

1.2.18 Endpunkt einer Strecke

Durch die Unterklassen `First` und `Last` erhält man Zugriff auf die beiden Endpunkte einer Strecke.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`point`	`first`	<Strecke [L]>
<Name [s]>	`point`	`last`	<Strecke [L]>

Beispiel:

```
e[1] = A;   point;   dragable;     1.0,3.0;
e[2] = B;   point;   dragable;     0.0,4.0;
e[3] = k1;  circle;  radius;       A,B;
e[4] = C;   point;   dragable;     2.0,3.0;
e[5] = D;   point;   dragable;     2.0,4.0;
e[6] = k2;  circle;  radius;       C,D;
e[7] = s;   line;    bichord;      k1,k2;
e[8] = P1;  point;   first;        s;
e[9] = P2;  point;   last;         s;
```

1.2.19 Punkt für ähnliche Dreiecke

Mit Hilfe der Unterklasse `Similar` läßt sich zu einem Dreieck ABC und zwei Punkten D und E ein dritter Punkt F erzeugen, so daß ABC und DEF zueinander ähnlich sind.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`point`	`similar`	<Punkt D [P]>, <Punkt E [P]>, <Punkt A [P]>, <Punkt B [P]>, <Punkt C [P]>

Beispiel:

```
e[1] = A;   point;   dragable;     1.0,3.0;
e[2] = B;   point;   dragable;     0.0,4.0;
e[3] = C;   point;   dragable;     2.0,3.0;
e[4] = D;   point;   dragable;     1.0,1.0;
e[5] = E;   point;   dragable;     3.0,2.0;
e[6] = F;   point;   similar;      D,E,A,B,C;
```

1.3 Strecke, Strahl, Gerade

1.3.1 Strecke

Eine Strecke wird definiert durch zwei Punkte A und B. Durch die optionale Angabe zweier Zeichenketten s_1 und s_2 kann die Länge der Strecke gemessen und auf der Zeichenfläche ausgegeben werden. Dabei werden s_1 und s_2 vor und hinter dem Wert der Streckenlänge angezeigt.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`line`	`connect`	<Punkt A [P]>, <Punkt B [P]>
<Name [s]>	`line`	`connect`	<Punkt A [P]>, <Punkt B [P]>, <Präfix s_1 [s]>, <Postfix s_2 [s]>

Beispiel:

```
e[1] = A;   point;   dragable;  1.0,1.0;
e[2] = B;   point;   dragable;  0.0,0.0;
e[3] = s;   line;    connect;   A,B,"s = "," L.E.";
```

1.3.2 Strahl

Ein Strahl wird definiert durch zwei Punkte S und T. Der Strahl beginnt im Punkt S und verläuft durch T.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`line`	`ray`	<Punkt S [P]>, <Punkt T [P]>

Beispiel:

```
e[1] = A;   point;   dragable;  1.0,1.0;
e[2] = B;   point;   dragable;  0.0,0.0;
e[3] = s;   line;    ray;       A,B;
```

1.3.3 Gerade

Eine Gerade wird definiert durch zwei Punkte A und B.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`line`	`straightLine`	<Punkt A [P]>, <Punkt B [P]>

Beispiel:

```
e[1] = A;   point;   dragable;      1.0,1.0;
e[2] = B;   point;   dragable;      0.0,0.0;
e[3] = g;   line;    straightLine;  A,B;
```

1.3.4 Orthogonale

Eine Orthogonale wird durch die Unterklasse `Perpendicular` realisiert. Als Objektdaten müssen dazu eine Gerade g und ein Punkt P übergeben werden, so daß eine Orthogonale h erzeugt wird, mit $h \perp g$ und $P \in h$.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`line`	`perpendicular`	<Punkt P [P]>, <Gerade g [L]>

Beispiel:

```
e[1] = A;   point;   dragable;      1.0,1.0;
e[2] = B;   point;   dragable;      0.0,0.0;
e[3] = C;   point;   dragable;      4.0,0.0;
e[4] = g;   line;    straightLine;  A,B;
e[5] = h;   line;    perpendicular; C,g;
```

1.3.5 Parallele

Eine Parallele wird durch die Unterklasse `Parallel` realisiert. Als Objektdaten müssen dazu eine Gerade g und ein Punkt P übergeben werden, so daß eine Parallele h erzeugt wird, mit $h \parallel g$ und $P \in h$.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`line`	`parallel`	<Punkt P [P]>, <Gerade g [L]>

Beispiel:

```
e[1] = A;   point;   dragable;      1.0,1.0;
e[2] = B;   point;   dragable;      0.0,0.0;
e[3] = C;   point;   dragable;      4.0,0.0;
e[4] = g;   line;    straightLine;  A,B;
e[5] = h;   line;    parallel;      C,g;
```

1.3.6 Winkelhalbierende

Eine Winkelhalbierende wird durch die Unterklasse `AngleBiSector` realisiert. Dazu muß ein Winkel $\angle ABC$ durch drei Punkte angegeben werden.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`line`	`angleBiSector`	<Punkt A [P]>, <Punkt B [P]>, <Punkt C [P]>

Beispiel:

```
e[1] = A;   point;   dragable;      1.0,1.0;
e[2] = B;   point;   dragable;      0.0,0.0;
e[3] = C;   point;   dragable;      4.0,0.0;
e[4] = w;   line;    angleBiSector; A,B,C;
```

1.3.7 Winkelteilende

Eine Winkelteilende ist eine Gerade, die einen Winkel $\angle ABC$ im Verhältnis $1 : n$ teilt. Sie wird durch die Unterklasse `AngleDivider` realisiert. Als Objektdaten sind für den Winkel drei Punkte anzugeben sowie ein Wert für n.
Beispiel:

```
e[1] = A;   point;   dragable;      1.0,1.0;
e[2] = B;   point;   dragable;      0.0,0.0;
e[3] = C;   point;   dragable;      4.0,0.0;
e[4] = w;   line;    angleDivider;  A,B,C,3.0;
```

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`line`	`angleDivider`	<Punkt A [P]>, <Punkt B [P]>, <Punkt C [P]>, <n [d]>

1.3.8 Dreieckshöhe

Die Höhe h_a in einem Dreieck ABC wird durch die Unterklasse `Foot` realisiert. Als Objektdaten sind drei Punkte A, B und C anzugeben.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`line`	`foot`	<Punkt A [P]>, <Punkt B [P]>, <Punkt C [P]>
<Name [s]>	`line`	`foot`	<Punkt A [P]>, <Strecke BC [L]>

Beispiel:

```
e[1] = A;   point;   dragable;  1.0,1.0;
e[2] = B;   point;   dragable;  0.0,0.0;
e[3] = C;   point;   dragable;  4.0,0.0;
e[4] = ha;  line;    foot;      A,B,C;
e[5] = c;   line;    connect;   A,B;
e[6] = hc;  line;    foot;      C,c;
```

1.3.9 Kreissehne

Eine Kreissehne kann durch die beiden Unterklassen `Chord` und `BiChord` realisiert werden. Im ersten Fall ist die Sehne definiert durch den Schnitt einer Geraden AB mit einem Kreis k. Im zweiten Fall ist die Sehne bestimmt durch den Schnitt zweier Kreise k_1 und k_2.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`line`	`chord`	<Punkt A [P]>, <Punkt B [P]>, <Kreis k [C]>
<Name [s]>	`line`	`biChord`	<Kreis k_1 [C]>, <Kreis k_2 [C]>

Beispiel:

```
e[1] = A;   point;   dragable;  1.0,1.0;
e[2] = B;   point;   dragable;  0.0,0.0;
e[3] = C;   point;   dragable;  4.0,0.0;
e[4] = D;   point;   dragable;  4.0,2.0;
e[5] = k1;  circle;  radius;    A,B;
e[6] = k2;  circle;  radius;    C,D;
e[7] = s;   line;    biChord;   k1,k2;
```

1.4 Kurven

1.4.1 Parametrisierte Kurven

Parametrisierte Kurven lassen sich durch die Unterklasse `Curve` realisieren. Sie können in kartesischen Koordinaten in der Form

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} f_x\,(t) \\ f_y\,(t) \end{pmatrix} \text{ und } t \in [t_0, t_1]$$

oder durch polare Koordinaten in der Form

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} r\cos t \\ r\sin t \end{pmatrix} \text{ mit } r = F(t) \text{ und } t \in [0, t_1]$$

angegeben werden.

Die Funktionen $f_x\,(t)$, $f_y\,(t)$ und $F(t)$ werden durch Objekte der Klasse `MeasureCalculate` realisiert, der Kurvenparameter ist darin mit `t` zu bezeichnen. Ferner ist eine Anzahl n von Stützpunkten anzugeben, durch die die Kurve approximiert wird.

Falls die Kurve nicht in dem Weltkoordinatensystem der Zeichenfläche dargestellt werden soll, kann als Alternative ein Koordinatensystem-Objekt (Seite 270) angegeben werden.
Beispiel:

```
e[1] = a;     measure;  controller;  10.0,10.0,5.0,1.0,"a = ","";
e[2] = b;     measure;  controller;  10.0,8.0,2.0,1.0,"b = ","";
e[3] = p1;    line;     curve;       "t","calculate(a)*t^2",
                                     -4.0,4.0,50;
e[4] = O;     point;    fixed;       4.0,4.0;
e[5] = coord; point;    coordSystem; O,200,200,200,200;
e[6] = p1;    line;     curve;       "t","calculate(a)*t^2+
                                     calculate(b)*t",coord,
                                     -4.0,4.0,50;
```

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`line`	`curve`	<$f_x\,(t)$ [MC]>, <$f_y\,(t)$ [MC]>, <t_0 [d]>, <t_1 [d]>, <n [i]>
<Name [s]>	`line`	`curve`	<$f_x\,(t)$ [MC]>, <$f_y\,(t)$ [MC]>, <Koord.-Syst. [CO]>, <t_0 [d]>, <t_1 [d]>, <n [i]>
<Name [s]>	`line`	`curve`	<$F\,(t)$ [MC]>, <t_1 [d]>, <n [i]>
<Name [s]>	`line`	`curve`	<$F\,(t)$ [MC]>, <Koord.-Syst. [CO]>, <t_1 [d]>, <n [i]>

1.4.2 Externe Kurven-Klassen

Mit Hilfe der Unterklasse `Curve` lassen sich Kurven einbinden, die durch separate, speziell entwickelte Java-Klassen berechnet werden. Die Objektdaten bestehen dabei aus n Zeichenketten $S_1, \ldots, S_n$ mit $1 \leq n \leq 10$. Vereinbarungsgemäß enthält S_1 den Bezeichner der Klasse, die eingebunden werden soll. Die restlichen Zeichenketten können weitere Objektbezeichner oder numerische Werte als Parameter für die Kurvenberechnung enthalten.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`line`	`curve`	<S_1 [s]>, <...>, <S_n [s]>

Beispiel:

```
e[1] = A;   point;   dragable;    -5.0,0.0;
e[2] = B;   point;   dragable;     5.0,0.0;
e[3] = s;   measure; controller;  10.0,11.0,1.0,3.0,"","";
e[4] = k;   line;    curve;       "Kochkurve","A","B","s";
```

1.4.3 Ortslinie

Mit Hilfe der Unterklasse `Locus` lassen sich Ortslinien abhängiger Punkte realisieren. Als Objektdaten müssen dazu zwei Punkte O und M sowie eine Führungslinie k angegeben werden. Wird M nun entlang von k bewegt, so approximiert die zu erzeugende Ortslinie die Bahn von O mit n Stützpunkten.

Als Führungslinie können die geometrischen Objekte Gerade, Kreis, Ortslinie oder Kurve dienen. Handelt es sich bei der Führungslinie um eine Ortslinie oder um eine Kurve, so braucht diese nicht explizit bei den Objektdaten angegeben zu werden. Sie wird als Bezugsobjekt von M automatisch erkannt.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`line`	`locus`	<Punkt O [P]>, <Punkt M [P]>, <Gerade k [L]>, <Stützpkte. n [i]>
<Name [s]>	`line`	`locus`	<Punkt O [P]>, <Punkt M [P]>, <Kreis k [C]>, <Stützpkte. n [i]>
<Name [s]>	`line`	`locus`	<Punkt O [P]>, <Punkt M [P]>

Beispiel:

```
e[1] = A;   point;   dragable;    -5.0,0.0;
e[2] = B;   point;   dragable;    -7.13,5.2;
e[3] = C;   point;   dragable;    10.13,7.93;
e[4] = s1;  line;    connect;     A,B;
e[5] = s2;  line;    connect;     C,B;
e[6] = P1;  point;   dragable;    -9.89,-2.58,s1;
e[7] = P2;  point;   proportion;  A,B,A,P1,B,C,B,C;
e[8] = s3;  line;    connect;     P1,P2;
e[9] = P3;  point;   proportion;  A,B,A,P1,P1,P2,P1,P2;
e[10] = l;  line;    locus;       P3,P1,s1,50;
```

1.4.4 Kegelschnitte

Kegelschnitte können mit Hilfe der Klasse `Conic` durch vier alternative Definitionsformen erzeugt werden:

1. durch die Angabe des Parameters p, der linearen Exzentrizität e sowie eines Drehwinkels α und einer Verschiebung um v_x und v_y,
2. durch die Angabe von a^2 und b^2 als Achsenabschnitte,
3. durch die Angabe der Parameter A, B, C, D, E, F der allgemeinen Kegelschnittgleichung und

4. durch die Angabe von fünf Punkten, durch die der Kegelschnitt verlaufen soll.

Anmerkung: Zum aktuellen Zeitpunkt (Stand: Juli 2000) ist die Klasse `Conic` noch nicht vollständig fehlerfrei. In speziellen Lagen wird der Kegelschnitt noch unzureichend approximiert, so daß der Kurvenverlauf als zu wenig geglättet erscheint.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`line`	`conic`	<Parameter p [M]>, <Exzentr. e [M]>, <Winkel α [M]>, <Transl. v_x [M]>, <Transl. v_y [M]>
<Name [s]>	`line`	`conic`	<Achsenabs. a^2 [M]>, <Achsenabs. b^2 [M]>
<Name [s]>	`line`	`conic`	<Parameter A [M]>, <Parameter B [M]>, <Parameter C [M]>, <Parameter D [M]>, <Parameter E [M]>, <Parameter F [M]>
<Name [s]>	`line`	`conic`	<Punkt [P]>, <Punkt [P]>, <Punkt [P]>, <Punkt [P]>, <Punkt [P]>

Beispiel:

```
e[1] = A;   point;   dragable;     1.0,7.0;
e[2] = B;   point;   dragable;    -5.0,1.0;
e[3] = C;   point;   dragable;     3.0,6.0;
e[4] = D;   point;   dragable;     5.0,0.0;
e[5] = E;   point;   dragable;    -4.0,4.0;
e[6] = c;   line;    conic;        A,B,C,D,E;
```

1.5 Kreise

1.5.1 Kreis

Ein Kreis wird durch die beiden Unterklassen `Radius` und `CircumCircle` realisiert. Im ersten Fall wird der Kreis definiert durch die Angabe eines Mittelpunkts M und den Radius r. Der Radius r kann dabei auf drei Arten festgelegt werden:

1. durch einen Punkt P mit $r = |MP|$,
2. durch zwei Punkte P_1P_2 mit $r = |P_1P_2|$ oder
3. durch eine Strecke AB mit $r = |AB|$.

Im zweiten Fall wird der Kreis definiert durch die Angabe von drei Punkten ABC, die auf der Kreislinie liegen.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`circle`	`radius`	<Punkt M [P]>, <Punkt P [P]>
<Name [s]>	`circle`	`radius`	<Punkt M [P]>, <Punkt P_1 [P]>, <Punkt P_2 [P]>
<Name [s]>	`circle`	`radius`	<Punkt M [P]>, <Strecke AB [L]>
<Name [s]>	`circle`	`circumCircle`	<Punkt A [P]>, <Punkt B [P]>, <Punkt C [P]>

Beispiel:

```
e[1] = A;   point;   dragable;      1.0,1.0;
e[2] = B;   point;   dragable;      0.0,0.0;
e[3] = C;   point;   dragable;      4.0,0.0;
e[4] = D;   point;   dragable;      4.0,2.0;
e[5] = k1;  circle;  radius;        A,B;
e[6] = k2;  circle;  circumCircle;  A,C,D;
```

1.5.2 Inversion eines Kreises an einem Kreis

Durch die Inversion eines Kreises an einem Kreis wird wiederum ein Kreis erzeugt. Dieses läßt sich durch die Unterklasse `Invert` realisieren. Als Objektdaten sind zwei Kreise k_1 und k_2 anzugeben.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`circle`	`invert`	<Kreis k_1 [C]>, <Kreis k_2 [C]>

Beispiel:

```
e[1] = A;   point;   dragable;      1.0,1.0;
e[2] = B;   point;   dragable;      0.0,0.0;
e[3] = C;   point;   dragable;      4.0,0.0;
e[4] = D;   point;   dragable;      4.0,2.0;
e[5] = k1;  circle;  radius;        A,B;
e[6] = k2;  circle;  radius;        C,D;
e[7] = k3;  circle;  invert;        k1,k2;
```

1.6 Kreissektoren

1.6.1 Kreisbogen

Mit Hilfe der Unterklassen `Sector` und `Arc` können Kreisbogen erzeugt werden. Im ersten Fall ist dazu ein Winkel $\angle ABC$ durch drei Punkte anzugeben, wobei der Kreisbogen mit dem Kreismittelpunkt B und dem Radius AB von A aus in Richtung auf C verläuft.

Im zweiten Fall wird ein Kreisbogen durch die Angabe von drei Punkten ABC auf der Kreislinie definiert. Der Bogen wird dabei durch die Punkte A und C begrenzt.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`sector`	`sector`	<Punkt B [P]>, <Punkt A [P]>, <Punkt C [P]>
<Name [s]>	`sector`	`arc`	<Punkt A [P]>, <Punkt B [P]>, <Punkt C [P]>

Beispiel:

```
e[1] = A;   point;   dragable;      1.0,1.0;
e[2] = B;   point;   dragable;      0.0,0.0;
e[3] = C;   point;   dragable;      4.0,0.0;
e[4] = s;   sector;  sector;        B,A,C;
e[5] = a;   sector;  arc;           A,B,C;
```

1.6.2 Winkelbogen

Mit Hilfe der Unterklasse `Angle` kann ein Winkelbogen erzeugt werden. Der Winkel $\angle ABC$ wird dabei durch drei Punkte definiert. Durch die optionale

Angabe zweier Zeichenketten s_1 und s_2 kann die Winkelgröße berechnet und ihr Wert auf der Zeichenfläche ausgegeben werden. Dabei werden s_1 und s_2 vor und hinter der Winkelgröße angezeigt.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`sector`	`angle`	<Punkt A [P]>, <Punkt B [P]>, <Punkt C [P]>
<Name [s]>	`sector`	`angle`	<Punkt A [P]>, <Punkt B [P]>, <Punkt C [P]>, <Präfix s_1 [s]>, <Postfix s_2 [s]>

Beispiel:

```
e[1] = A;   point;   dragable;      1.0,1.0;
e[2] = B;   point;   dragable;      0.0,0.0;
e[3] = C;   point;   dragable;      4.0,0.0;
e[4] = s;   sector;  angle;         A,B,C,"\beta = "," °";
```

1.7 Polygone

1.7.1 Polygon mit vorgegebenen Eckpunkten

Aus einer vorgegebenen Anzahl von Eckpunkten kann durch die Unterklasse `Polygon` ein Vieleck realisiert werden. Als Objektdaten sind dazu n Eckpunkte $P_1, \ldots, P_n$ anzugeben, mit $3 \leq n \leq 17$.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`polygon`	`polygon`	<Punkt P_1 [P]>, ..., <Punkt P_n [P]>

Beispiel:

```
e[1] = A;   point;   dragable;      1.0,1.0;
e[2] = B;   point;   dragable;      0.0,0.0;
e[3] = C;   point;   dragable;      4.0,0.0;
e[4] = D;   point;   dragable;      4.0,2.0;
e[5] = p;   polygon; polygon;       A,B,C,D;
```

1.7.2 Gleichseitiges Dreieck

Gleichseitige Dreiecke werden durch die Unterklasse `EquilateralTriangle` realisiert. Als Objektdaten sind dazu zwei Eckpunkte A und B anzugeben, die die Seitenlänge festlegen.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`polygon`	`equilateralTriangle`	<Punkt A [P]>, <Punkt B [P]>

Beispiel:

```
e[1] = A;   point;   dragable;              1.0,1.0;
e[2] = B;   point;   dragable;              0.0,0.0;
e[3] = p;   polygon; equilateralTriangle;   A,B;
```

1.7.3 Ähnliches Dreieck

Ähnliche Dreiecke lassen sich durch die Unterklasse `Similar` realisieren. Als Objektdaten sind dazu ein Dreieck ABC und zwei Eckpunkte D und E anzugeben.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`polygon`	`similar`	<Punkt D [P]>, <Punkt E [P]>, <Punkt A [P]>, <Punkt B [P]>, <Punkt C [P]>

Beispiel:

```
e[1] = A;   point;   dragable;   -6.0,5.0;
e[2] = B;   point;   dragable;   -3.0,5.0;
e[3] = C;   point;   dragable;   -5.0,0.0;
e[4] = D;   point;   dragable;    0.0,5.0;
e[5] = E;   point;   dragable;    3.0,5.0;
e[6] = p;   polygon; similar;     D,E,A,B,C;
```

1.7.4 Quadrat

Quadrate lassen sich durch die Unterklasse `Square` realisieren. Dabei muß eine Quadratseite durch zwei Punkte AB angegeben werden.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`polygon`	`square`	<Punkt A [P]>, <Punkt B [P]>

Beispiel:

```
e[1] = A;   point;   dragable;   -6.0,5.0;
e[2] = B;   point;   dragable;   -3.0,5.0;
e[3] = p;   polygon; square;      A,B;
```

1.7.5 Parallelogramm

Zu drei Punkten A, B, C kann mit Hilfe der Unterklasse `Parallelogram` ein Kantenzug erzeugt werden, der ein Parallelogramm darstellt.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`polygon`	`parallelogram`	<Punkt A [P]>, <Punkt B [P]>, <Punkt C [P]>

Beispiel:

```
e[1] = A;   point;   dragable;      -6.0,5.0;
e[2] = B;   point;   dragable;      -3.0,5.0;
e[3] = C;   point;   dragable;       3.0,0.0;
e[4] = p;   polygon; parallelogram;  A,B,C;
```

1.7.6 Reguläre Polygone

Aus einer Seite AB heraus kann man mit Hilfe der Unterklasse `RegularPolygon` ein regelmäßiges n-Eck erzeugen.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`polygon`	`regularPolygon`	<Punkt A [P]>, <Punkt B [P]>, <n [i]>

Beispiel:

```
e[1] = A;   point;   dragable;        6.0,5.0;
e[2] = B;   point;   dragable;        3.0,5.0;
e[3] = p;   polygon; regularPolygon;  A,B,5;
```

1.7.7 Reguläre Sternpolygone

Aus einer Seite AB heraus kann man mit Hilfe der Unterklasse `starPolygon` ein regelmäßiges, sternförmiges n-Eck erzeugen, das eine Kante zwischen jeder i-ten Ecke hat.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`polygon`	`starPolygon`	<Punkt A [P]>, <Punkt B [P]>, <n [i]>, <i [i]>

Beispiel:

```
e[1] = A;   point;   dragable;        6.0,5.0;
e[2] = B;   point;   dragable;        3.0,5.0;
e[3] = p;   polygon; starPolygon;     A,B,5,3;
```

1.8 Punktmengen

Mit Hilfe der Unterklasse `PointSet` kann eine statische Punktmenge auf der Zeichenfläche erzeugt werden. Dazu ist die Angabe einer Bilddatei erforderlich. Die Menge aller schwarz gefärbten Bildpunkte definiert die Punktmenge. Als optionaler Parameter n kann die Größe des Arrays zum Speichern der Punktmenge begrenzt werden. Hierdurch wird Speicherplatz gespart. Wenn die Punktmenge als Bezugsobjekt für einen ziehbaren Punkt dient, kann zwischen zwei alternativen Verfahren im Zugmodus gewählt werden. Voreingestellt ist der "Gummibandmodus", d. h. ein ziehbarer Punkt kann wie an einem Gummiband innerhalb der Punktmenge gezogen werden. Wird ein dritter Parameter i angegeben, so ist der "Gummibandmodus" ausgeschaltet und ein Punkt läßt sich mit bis zu i Bildschirmpunkten Abstand zum Mauszeiger ziehen.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`line`	`pointSet`	<Dateiname [s]>
<Name [s]>	`line`	`pointSet`	<Dateiname [s]>, <n [i]>
<Name [s]>	`line`	`pointSet`	<Dateiname [s]>, <n [i]>, <i [i]>

Beispiel:

```
e[1] = Pset; line;   pointSet;   "Kurve01.gif", 10000, 10;
e[2] = A;    point;  dragable;   2.7,5.4, Pset;
```

1.9 Koordinatensysteme

Mit Hilfe der Unterklasse `CoordSystem` kann ein Koordinatensystem auf der Zeichenfläche erzeugt werden, in dem Kurven oder Punkte dargestellt werden können. Die Längen der Koordinatenachsen werden in Bildschirmpunkten durch die vier Variablen v_1 (negative x-Achse), v_2 (positive x-Achse), v_3 (negative y-Achse) und v_4 (positive y-Achse) festgelegt. Die Größe der x- und y-Einheit wird durch den Wert zweier Funktionale oder durch den Abstand von jeweils zwei Punkten, mit $dx = |P_1P_2|$ und $dx = |P_3P_4|$, bestimmt.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`point`	`coordSystem`	<Ursprung [P]>, <x-Einheit [M]>, <y-Einheit [M]>, <v_1 [i]>, <v_2 [i]>, <v_3 [i]>, <v_4 [i]>
<Name [s]>	`point`	`coordSystem`	<Ursprung [P]>, <P_1 [P]>, <P_2 [P]>, <P_3 [P]>, <P_4 [P]>, <v_1 [i]>, <v_2 [i]>, <v_3 [i]>, <v_4 [i]>
<Name [s]>	`point`	`coordSystem`	<Ursprung [P]>, <v_1 [i]>, <v_2 [i]>, <v_3 [i]>, <v_4 [i]>

Beispiel:

```
e[1] = O;   point;   dragable;     -6.0,-6.0;
e[2] = X;   point;   dragable;      6.0,-6.0;
e[3] = Y;   point;   dragable;      6.0,0.0;
e[4] = c;   point;   coordSystem;   O,X,Y,X,Y,200,200,200,200;
```

2 Funktionale

2.1 Hinweise zum Arbeiten mit Funktionalen

Im folgenden sollen einige Hinweise zum Arbeiten mit Funktionalen gegeben werden. Unter einem Funktional wird in diesem Zusammenhang eine Abbildung $F\colon\ M\ \rightarrow\ \mathbb{R}$ verstanden. Der Definitionsbereich M besteht dabei aus dem Zustandsraum einer Menge von Objekten. Der Wert eines Funktionals ist eine reelle Zahl. Durch Funktionale können die Eigenschaften von Objekten gemessen werden und es lassen sich Relationen zwischen mehreren Objekten prüfen. Funktionale haben mehrere Aufgaben:

- Sie liefern Parameterwerte für die Konstruktion geometrischer Objekte.
- Sie werden bei der Definition von Antwortwerten und Prüfschlüsseln für eine Antwortanalyse verwendet.
- Sie geben Rückmeldung über den Figurenzustand an den Schüler, indem die Werte auf der Zeichenfläche angezeigt werden.

Um letzteres zu erreichen, werden die Objektdaten um die Angabe zweier Koordinatenwerte, eines Präfixes und eines Suffixes erweitert. Die Koordinatenwerte müssen vom Typ `double` sein, Präfix und Suffix vom Typ `String`. Alle vier Parameter sind durch Kommata zu trennen. Der Kürze halber sind in den folgenden Datentabellen diese vier Zusatzparameter weggelassen.

2.2 Eigenschaften von Objekten

2.2.1 Eigenschaften von Punkten

Mit Hilfe der Unterklasse `Property` kann auf die Eigenschaften eines Punkts zugegriffen werden. Neben den x- und y-Koordinaten kann mit `defined` geprüft werden, ob die Koordinaten reelle Werte enthalten. Ist der Punkt ziehbar auf einer parametrisierten Kurve, so kann der Wert des Kurvenparameters `t` gemessen werden.
Beispiel:

```
e[1] = A;    point;    dragable;  2.0,5.0;
e[2] = m0;   measure;  property;  A,"x",10.0,5.0,"A.x = ","";
e[3] = m1;   measure;  property;  A,"y",10.0,4.0,"A.y = ","";
e[4] = m2;   measure;  property;  A,"defined";
```

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`measure`	`property`	<Punkt [P]>, <x-Koordinate [`"x"`]>
<Name [s]>	`measure`	`property`	<Punkt [P]>, <y-Koordinate [`"y"`]>
<Name [s]>	`measure`	`property`	<Punkt [P]>, <Koord. reell? [`"defined"`]>
<Name [s]>	`measure`	`property`	<Punkt [P]>, <Kurvenparameter [`"t"`]>

2.2.2 Eigenschaften von Strecken und Geraden

Auf die Eigenschaften von Strecken und Geraden kann durch die Unterklasse `Property` zugegriffen werden.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`measure`	`property`	<Gerade [L]>, <Steigung [`"slope"`]>
<Name [s]>	`measure`	`property`	<Gerade [L]>, <Abstand [`"distance"`]>
<Name [s]>	`measure`	`property`	<Gerade [L]>, <$ax + by + c = 0$ [`"Param_a"`]>
<Name [s]>	`measure`	`property`	<Gerade [L]>, <$ax + by + c = 0$ [`"Param_b"`]>
<Name [s]>	`measure`	`property`	<Gerade [L]>, <$ax + by + c = 0$ [`"Param_c"`]>
<Name [s]>	`measure`	`property`	<Gerade [L]>, <$y = mx + b$ [`"b"`]>

Beispiel:

```
e[1] = A;    point;    dragable;      2.0,5.0;
e[2] = B;    point;    dragable;      1.0,0.0;
e[3] = g;    line;     straightLine;  A,B;
e[4] = m0;   measure;  property;      g,"slope",10.0,10.0,
                                      "Steigung = ","";
```

2.2.3 Eigenschaften von Kreisen

Mit Hilfe der Unterklasse `Property` kann auf die Eigenschaften von Kreisen zugegriffen werden.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`measure`	`property`	<Kreis [C]>, <$(x - x_0)^2 + (y - y_0)^2 = r^2$ [`"x0"`]>
<Name [s]>	`measure`	`property`	<Kreis [C]>, <$(x - x_0)^2 + (y - y_0)^2 = r^2$ [`"y0"`]>
<Name [s]>	`measure`	`property`	<Kreis [C]>, <$(x - x_0)^2 + (y - y_0)^2 = r^2$ [`"r2"`]>
<Name [s]>	`measure`	`property`	<Kreis [C]>, <Radius [`"radius"`]>
<Name [s]>	`measure`	`property`	<Kreis [C]>, <Durchmesser [`"perimeter"`]>
<Name [s]>	`measure`	`property`	<Kreis [C]>, <Flächeninhalt [`"area"`]>
<Name [s]>	`measure`	`property`	<Kreis [C]>, <Umfang [`"circumference"`]>
<Name [s]>	`measure`	`property`	<Kreis [C]>, <x-Koordinate des Mittelpunkts [`"center_x"`]>
<Name [s]>	`measure`	`property`	<Kreis [C]>, <y-Koordinate des Mittelpunkts [`"center_y"`]>

Beispiel:

```
e[1] = A;    point;    dragable;     2.0,5.0;
e[2] = B;    point;    dragable;     1.0,0.0;
e[3] = k;    circle;   radius;       A,B;
e[4] = m0;   measure;  property;     k,"area",8.0,10.0,
                                     "Flächeninhalt = "," F.E.";
e[5] = m1;   measure;  property;     k,"circumference",8.0,9.0,
                                     "Umfang = "," L.E.";
```

2.2.4 Eigenschaften von Polygonen

Mit Hilfe der Unterklasse `Property` kann auf die Eigenschaften von Polygonen zugegriffen werden.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`measure`	`property`	<Polygon [PG]>, <Flächeninhalt [`"area"`]>
<Name [s]>	`measure`	`property`	<Polygon [PG]>, <Umfang [`"circumference"`]>
<Name [s]>	`measure`	`property`	<Polygon [PG]>, <Anzahl von Symmetrieachsen [`"numSymmetryAxis"`]>
<Name [s]>	`measure`	`property`	<Polygon [PG]>, <Anzahl von Drehsymmetrien [`"numSymmetryRotation"`]>
<Name [s]>	`measure`	`property`	<Polygon [PG]>, <Konvexität [`"isConvex"`]>
<Name [s]>	`measure`	`property`	<Polygon [PG]>, <Regularität [`"isRegular"`]>
<Name [s]>	`measure`	`property`	<Polygon [PG]>, <Affine Regularität [`"isAffinRegular"`]>
<Name [s]>	`measure`	`property`	<Polygon [PG]>, <Anz. inzidierender Eckpunkte [`"numIncidentVertex"`]>
<Name [s]>	`measure`	`property`	<Polygon [PG]>, <Umfang [`"circumference"`]>
<Name [s]>	`measure`	`property`	<Polygon [PG]>, <x-Koordinate des Mittelpunkts [`"center_x"`]>
<Name [s]>	`measure`	`property`	<Polygon [PG]>, <y-Koordinate des Mittelpunkts [`"center_y"`]>

Falls das Polygon ein Dreieck ist, können die besonderen Dreiecksklassen analysiert werden. Der Befehl `bestClass` liefert die beste Dreiecksklasse. Mit dem Befehl `checkClass_i` kann ein vorliegendes Dreieck gezielt auf eine bestimmte Klasse hin geprüft werden.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`measure`	`property`	<Polygon [PG]>, <beste Figurenklasse ["bestClass"]>
<Name [s]>	`measure`	`property`	<Polygon [PG]>, <gleichseitig ["checkClass_1"]>
<Name [s]>	`measure`	`property`	<Polygon [PG]>, <gleichschenklig u. rechtwinklig ["checkClass_2"]>
<Name [s]>	`measure`	`property`	<Polygon [PG]>, <gleichschenklig u. stumpfwinklig ["checkClass_3"]>
<Name [s]>	`measure`	`property`	<Polygon [PG]>, <gleichschenklig u. spitzwinklig ["checkClass_4"]>
<Name [s]>	`measure`	`property`	<Polygon [PG]>, <rechtwinklig ["checkClass_5"]>
<Name [s]>	`measure`	`property`	<Polygon [PG]>, <stumpfwinklig ["checkClass_6"]>
<Name [s]>	`measure`	`property`	<Polygon [PG]>, <spitzwinklig ["checkClass_7"]>
<Name [s]>	`measure`	`property`	<Polygon [PG]>, <gleichschenklig ["checkClass_8"]>

Falls das Polygon ein Viereck ist, können die besonderen Viereckssklassen erkannt werden. Der Befehl `bestClass` liefert die beste Viereckssklasse. Mit dem Befehl `checkClass_i` kann ein vorliegendes Viereck gezielt auf eine bestimmte Klasse hin geprüft werden.
Beispiel:

```
e[1] = A;    point;    dragable;      2.0,5.0;
e[2] = B;    point;    dragable;      1.0,0.0;
e[3] = C;    point;    dragable;      2.0,1.0;
e[4] = D;    point;    dragable;      1.0,4.0;
e[5] = p;    polygon;  polygon;       A,B,C,D;
e[6] = m0;   measure;  property;      p,"area",8.0,10.0,
                                      "Flächeninhalt = "," F.E.";
e[7] = m1;   measure;  property;      p,"circumference",8.0,9.0,
                                      "Umfang = "," L.E.";
e[8] = m2;   measure;  property;      p,"bestClass";
```

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`measure`	`property`	<Polygon [PG]>, <beste Figurenklasse ["bestClass"]>
<Name [s]>	`measure`	`property`	<Polygon [PG]>, <Quadrat ["checkClass_1"]>
<Name [s]>	`measure`	`property`	<Polygon [PG]>, <Rechteck ["checkClass_2"]>
<Name [s]>	`measure`	`property`	<Polygon [PG]>, <Raute ["checkClass_3"]>
<Name [s]>	`measure`	`property`	<Polygon [PG]>, <Parallelogramm ["checkClass_4"]>
<Name [s]>	`measure`	`property`	<Polygon [PG]>, <gleichschenkliges Trapez ["checkClass_5"]>
<Name [s]>	`measure`	`property`	<Polygon [PG]>, <schiefes Trapez ["checkClass_6"]>
<Name [s]>	`measure`	`property`	<Polygon [PG]>, <gleichschenkliger Drachen ["checkClass_7"]>
<Name [s]>	`measure`	`property`	<Polygon [PG]>, <schiefer Drachen ["checkClass_8"]>
<Name [s]>	`measure`	`property`	<Polygon [PG]>, <schräger Drachen ["checkClass_9"]>
<Name [s]>	`measure`	`property`	<Polygon [PG]>, <Sehnenviereck ["checkClass_10"]>
<Name [s]>	`measure`	`property`	<Polygon [PG]>, <Pythagoräisches Viereck ["checkClass_11"]>

2.3 Implementierte Funktionale

2.3.1 Abstand

Durch die Unterklasse `Distance` kann der euklidische Abstand zwischen zwei Objekten gemessen werden.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`measure`	`distance`	<Punkt [P]>, <Punkt [P]>
<Name [s]>	`measure`	`distance`	<Punkt [P]>, <Gerade [L]>
<Name [s]>	`measure`	`distance`	<Punkt [P]>, <Kreis [C]>

Beispiel:

```
e[1] = A;    point;    dragable;      2.0,5.0;
e[2] = B;    point;    dragable;      1.0,0.0;
e[3] = C;    point;    dragable;      2.0,1.0;
e[4] = g;    line;     straightLine;  A,B;
e[5] = m0;   measure;  distance;      C,g,8.0,10.0,
                                      "Abstand = "," L.E.";
```

2.3.2 Winkel

Durch die Unterklasse `Angle` kann ein durch drei Punkte definierter Winkel $\angle ABC$ gemessen werden.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`measure`	`angle`	<Punkt A [P]>, <Punkt B [P]>, <Punkt C [P]>

Beispiel:

```
e[1] = A;    point;    dragable;      2.0,5.0;
e[2] = B;    point;    dragable;      1.0,0.0;
e[3] = C;    point;    dragable;      2.0,1.0;
e[4] = m0;   measure;  angle;         A,B,C,8.0,10.0,
                                      "Winkel = "," °";
```

2.3.3 Streckenverhältnisse

Durch die Unterklasse `Ratio` kann das Verhältnis einer Strecke AB zu einer Strecke CD gemessen werden.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`measure`	`ratio`	<Punkt A [P]>, <Punkt B [P]>, <Punkt C [P]>, <Punkt D [P]>

Beispiel:

```
e[1] = A;    point;    dragable;      2.0,5.0;
e[2] = B;    point;    dragable;      1.0,0.0;
e[3] = C;    point;    dragable;      2.0,1.0;
e[4] = D;    point;    dragable;      2.0,3.0;
e[5] = m0;   measure;  ratio;         A,B,C,D,8.0,10.0,
                                      "AB/CD = ","";
```

2.3.4 Ähnlichkeit

Durch die Unterklasse `Similarity` kann geprüft werden, ob zwei Polygone zueinander ähnlich sind. Das Funktional liefert den Wert 1 oder 0.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`measure`	`similarity`	<Polygon [PG]>, <Polygon [PG]>

Beispiel:

```
e[1] = A;    point;    dragable;      2.0,5.0;
e[2] = B;    point;    dragable;      1.0,0.0;
e[3] = C;    point;    dragable;      2.0,1.0;
e[4] = D;    point;    dragable;      2.0,3.0;
e[5] = E;    point;    dragable;      6.0,2.0;
e[6] = F;    point;    dragable;      7.0,3.0;
e[7] = d1;   polygon;  polygon;       A,B,C;
e[8] = d2;   polygon;  polygon;       D,E,F;
e[9] = m0;   measure;  similarity;    d1,d2,8.0,10.0,
                                      "d1 ~ d2 = ","";
```

2.3.5 Kollinearität

Mit Hilfe der Unterklasse `Collinear` kann geprüft werden, ob drei Punkte zueinander kollinear sind. Das Funktional liefert den Wert 1 oder 0.
Beispiel:

```
e[1] = A;    point;    dragable;      2.0,5.0;
e[2] = B;    point;    dragable;      1.0,0.0;
e[3] = C;    point;    dragable;      2.0,1.0;
e[4] = m0;   measure;  collinear;     A,B,C,8.0,10.0,
                                      "Kollinearität = ","";
```

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`measure`	`collinear`	<Punkt [P]>, <Punkt [P]>, <Punkt [P]>

2.3.6 Kongruenz

Mit Hilfe der Unterklasse `Congruence` kann geprüft werden, ob zwei Polygone oder zwei Kreise kongruent sind. Das Funktional liefert den Wert 1 oder 0.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`measure`	`congruence`	<Polygon [PG]>, <Polygon [PG]>
<Name [s]>	`measure`	`congruence`	<Kreis [C]>, <Kreis [C]>

Beispiel:

```
e[1] = A;   point;   dragable;     1.0,3.0;
e[2] = B;   point;   dragable;     0.0,4.0;
e[3] = k1;  circle;  radius;       A,B;
e[4] = C;   point;   dragable;     2.0,3.0;
e[5] = D;   point;   dragable;     2.0,4.0;
e[6] = k2;  circle;  radius;       C,D;
e[7] = m0;  measure; congruence;   k1,k2,8.0,10.0,
                                   "Kongruenz = ","";
```

2.3.7 Inzidenz

Mit Hilfe der Unterklasse `Incidence` kann geprüft werden, ob ein Punkt P mit einem Objekt O inzidiert. Das Funktional liefert den Wert 1 oder 0.
Beispiel:

```
e[1] = A;   point;   dragable;     1.0,3.0;
e[2] = B;   point;   dragable;     0.0,4.0;
e[3] = k;   circle;  radius;       A,B;
e[4] = P;   point;   dragable;     2.0,4.0;
e[5] = m0;  measure; incidence;    P,k,8.0,10.0,
                                   "P liegt auf k = ","";
```

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`measure`	`incidence`	<Punkt P [P]>, <Punkt O [P]>
<Name [s]>	`measure`	`incidence`	<Punkt P [P]>, <Gerade O [L]>
<Name [s]>	`measure`	`incidence`	<Punkt P [P]>, <Kreis O [C]>
<Name [s]>	`measure`	`incidence`	<Punkt P [P]>, <x-Koordinatenfkt. [M]>, <y-Koordinatenfkt. [M]>

2.3.8 Inklusion

Mit Hilfe der Unterklasse `Inclusion` kann geprüft werden, ob sich ein Punkt innerhalb der Fläche eines Kreises oder eines Polygons befindet. Das Funktional liefert den Wert 1 oder 0.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`measure`	`inclusion`	<Punkt [P]>, <Kreis [C]>
<Name [s]>	`measure`	`inclusion`	<Punkt [P]>, <Polygon [PG]>

Beispiel:

```
e[1] = A;   point;   dragable;     1.0,3.0;
e[2] = B;   point;   dragable;     0.0,4.0;
e[3] = k;   circle;  radius;       A,B;
e[4] = P;   point;   dragable;     2.0,4.0;
e[5] = m0;  measure; inclusion;    P,k,8.0,10.0,
                                   "P liegt in k = ","";
```

2.3.9 Parallelität

Mit Hilfe der Unterklasse `Parallel` kann geprüft werden, ob zwei Geraden AB und CD zueinander parallel sind. Das Funktional liefert den Wert 1 oder 0.
Beispiel:

```
e[1] = A;   point;   dragable;     1.0,3.0;
e[2] = B;   point;   dragable;     0.0,4.0;
e[3] = C;   point;   dragable;     3.0,3.0;
e[4] = D;   point;   dragable;     3.0,4.0;
e[5] = g;   line;    straightLine; A,B;
```

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`measure`	`parallel`	<Gerade AB [L]>, <Gerade CD [L]>
<Name [s]>	`measure`	`parallel`	<Punkt A [P]>, <Punkt B [P]>, <Punkt C [P]>, <Punkt D [P]>

```
e[6] = h;   line;    straightLine; C,D;
e[7] = m0;  measure; parallel;     g,h,8.0,10.0,
                                   "g || h = ","";
```

2.3.10 Orthogonalität

Durch die Unterklasse `Perpendicular` kann geprüft werden, ob zwei Geraden AB und CD zueinander orthogonal sind. Das Funktional liefert den Wert 1 oder 0.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`measure`	`perpendicular`	<Gerade AB [L]>, <Gerade CD [L]>
<Name [s]>	`measure`	`perpendicular`	<Punkt A [P]>, <Punkt B [P]>, <Punkt C [P]>, <Punkt D [P]>

Beispiel:

```
e[1] = A;   point;   dragable;      1.0,3.0;
e[2] = B;   point;   dragable;      0.0,4.0;
e[3] = C;   point;   dragable;      3.0,3.0;
e[4] = D;   point;   dragable;      3.0,4.0;
e[5] = g;   line;    straightLine;  A,B;
e[6] = h;   line;    straightLine;  C,D;
e[7] = m0;  measure; perpendicular; g,h,8.0,10.0,
                                    "g orthogonal zu h = ","";
```

2.4 Umgang mit Termen

2.4.1 Termevaluation

Das Berechnen von Termen erfolgt durch die Unterklasse `Calculate`. Ein Term wird als Zeichenkette in Anführungszeichen angegeben. In dem Term können die mathematischen Operatoren `+ - * / ^` enthalten sein. Daneben gibt es die

logischen Verknüpfungen UND (`&`), ODER (`|`) und XOR (`**`). Die Negation ist als Funktion `not()` realisiert. Weitere verfügbare mathematische Funktionen sind: `sin()`, `cos()`, `tan()`, `sqrt()`, `min()`, `max()`, `asin()`, `acos()`, `atan()`, `exp()`, `log()` und `abs()`.

In einem Term gibt es außerdem mehrere alternative Möglichkeiten, Funktionale einzubinden.

1. Um auf ein bereits im Skript definiertes Funktional Bezug zu nehmen, kann der Befehl `calculate(`<Objektbezeichner>`)` verwendet werden.
2. Um auf die Eigenschaften von Objekten zuzugreifen, steht der Befehl `$(`<Objektbezeichner>`,`<Objekteigenschaft>`)` zur Verfügung. Die Objekteigenschaften entsprechen dabei den in Abschnitt "Eigenschaften von Objekten" (Seite 271-279) aufgeführten Parametern.
3. Um die in *GeoScript* implementierten Funktionale zu verwenden, sind die folgenden Befehle verfügbar: `angle()`, `area()`, `isIncident()`, `isParallel()`, `isPerpendicular()`, `isCollinear()`, `isIncluded()`, `distance()`, `XVector()`, `YVector()` und `isSimilar()`. In die Klammern sind jeweils die entsprechenden Parameter einzusetzen (Seite 279-283).
4. Um speziell auf die Koordinaten eines Punkts zuzugreifen, gibt es alternativ zu dem `$()`-Befehl die beiden Funktionale `coordinateX(`<Punktbezeichner>`)` und `coordinateY(`<Punktbezeichner>`)`.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`measure`	`calculate`	<Term [MC]>

Beispiel:

```
e[1] = A;   point;   dragable;       1.0,3.0;
e[2] = B;   point;   dragable;       3.0,3.0;
e[3] = Mx;  measure; calculate;      "($(A,x)+$(B,x))/2";
e[4] = My;  measure; calculate;      "(coordinateY(A)+
                                       coordinateY(B))/2";
e[5] = M;   point;   functionDepend; "calculate(Mx)",
                                     "calculate(My)";
```

2.4.2 Prüffunktionen

Mit Hilfe der oben aufgeführten Termevaluation lassen sich auch sog. Prüffunktionen realisieren, die nur die Werte 1 (wahr) und 0 (falsch) annehmen können. Eine Prüffunktionen wird aufgestellt, indem zwei einzelne Terme T_1 und T_2 (Objekte der Klasse `MeasureCalculate`) durch die Symbole `<`, `>`, `=` und `!=` verknüpft werden.

Bei der Formulierung einer Prüffunktion in einem Skript ist zu beachten, daß T_1 und T_2 und das Relationssymbol jeweils durch ein Leerzeichen getrennt sind. Innerhalb der Zeichenketten von T_1 und T_2 dürfen keine Leerzeichen stehen.

Alternativ kann eine Prüffunktion auch durch die Angabe lediglich eines Funktionals definiert werden, sofern sichergestellt ist, daß das Funktional nur die

Werte 1 und 0 annehmen kann. Dies ist etwa bei den Funktionalen zum Prüfen auf Ähnlichkeit, Kollinearität, Kongruenz, Inzidenz, Inklusion, Parallelität und Orthogonalität der Fall. Auch läßt sich das Interaktionselement Checkbox als ein booleschwertiges Funktional auffassen.

Verwendet werden Prüffunktionen im Zusammenhang mit Fallunterscheidungen in Termen (Seite 285), beim Begrenzen des Zustandsraums einer Figur (Seite 288), beim Ein- und Ausblenden von Objekten (Seite 288), beim Erzeugen von speziellen Figurenzuständen (Seite 288) sowie bei der Definition einer Antwortanalyse (Seite 290) vor.

Eine Prüffunktion als ein separates Objekt zu erzeugen, ist nicht vorgesehen. Daher ist an dieser Stelle auch keine Datentabelle und kein Beispiel aufgeführt.

2.4.3 Terme mit Fallunterscheidungen

Fallunterscheidungen können mit Hilfe einer Prüffunktion (Seite 284) realisiert werden. Dazu wird ähnlich wie bei der Definition eines Terms der Unterklasse `Calculate` vorgegangen. Der Term muß jedoch wie folgt aufgebaut sein:
`"if (`<Prüffkt. [MCond]>`) then (`<Term [MC]>`) else (`<Term [MC]>`)"`

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`measure`	`calculate`	<`"if ([MCond]) then ([MC]) else ([MC])"`>

Beispiel:

```
e[1] = A;   point;   dragable;       1.0,3.0;
e[2] = B;   point;   dragable;       3.0,3.0;
e[3] = P;   point;   fixed;         -3.0, 3.0;
e[4] = g;   line;    straightLine;   B,A;
e[5] = m0;  measure; calculate;     "if (distance(P,g) < 0.05)
                                     then (distance(A,B))
                                     else (distance(A,P)+
                                     distance(B,P))",
                                     -5.0,10.0,"d(A,B) = ","";
```

2.4.4 Externe Funktionsbibliotheken

Externe Funktionsbibliotheken können durch die Unterklasse `Function` eingebunden werden. Die Funktionsbibliotheken müssen dabei als Java-Klassen vorliegen. Die Objektdaten bestehen dabei aus n Zeichenketten $S_1, \ldots, S_n$ mit $1 \leq n \leq 10$. Vereinbarungsgemäß enthält S_1 den Bezeichner der Klasse, die eingebunden werden soll. Die restlichen Zeichenketten können weitere Objektbezeichner oder numerische Werte enthalten, die als Parameter dienen.
Beispiel:

```
e[1] = a;  measure;   controller; 5.0,5.0,1.0,3.0,"a = ","";
e[2] = b;  measure;   controller; 5.0,4.0,1.0,3.0,"b = ","";
e[3] = e;  line;      curve;      "t","exp(calculate(a)*t+
```

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`measure`	`function`	<S_1 [s]>, <...>, <S_n [s]>

```
                                        calculate(b))", -8.0, 8.0, 50;
e[4] = x0; measure;   calculate;  "-100.0";
e[5] = x1; measure;   calculate;  "1.0";
e[6] = m0; measure;   function;   "Functional_Integral","e",
                                  "x0","x1","50","simpson",4.0,3.0,
                                  "Flächeninhalt = "," F.E.";
```

2.5 Interaktionselemente

2.5.1 Schieberegler

Schieberegler werden durch die Unterklasse `Controller` realisiert. Mit ihnen lassen sich numerische Werte aus einem vorgegebenen Intervall einstellen, die als Parameter für die Definition von geometrischen Objekten dienen können.

Als Objektdaten sind anzugeben: ein Anfangswert a, die Schrittweite s, ein Intervall $[x_0,\ x_1]$, die Breite b des Schiebereglers in Bildschirmpunkten sowie ein Präfix und Suffix zur Beschriftung.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`measure`	`controller`	<Anfangswert a [d]>, <Schrittweite s [d]>, <Intervallwert x_0 [d]>, <Intervallwert x_1 [d]>, <Breite b [i]>, <Präfix [s]>, <Suffix [s]>

Beispiel:

```
e[1] = a; measure; controller; 1.25,0.25,-1.0,5.0,50,"a = ","";
e[2] = b; measure; controller; 2.0,0.5,-1.0,5.0,50,"b = ","";
e[3] = e; line;    curve;      "t","exp(calculate(a)*t+
                               calculate(b))", -8.0, 8.0, 50;
```

2.5.2 Checkbox

Zweistufige Schaltelemente werden durch die Unterklasse `Checkbox` realisiert. Eine Checkbox kann lediglich die Werte 1 und 0 annehmen. Als Objektdaten ist ein Anfangswert und eine Beschriftung anzugeben.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`measure`	`checkbox`	<Beschriftung [s]>, <Anfangswert [i]>

Beispiel:

```
e[1]= cb1;  measure;  checkbox;  "Hilfe zeigen",0;
e[2]= cb2;  measure;  checkbox;  "Lösung zeigen",0;
```

2.5.3 Button

Ein Schalter kann durch die Unterklasse `Button` erzeugt werden. Durch das Anklicken eines Schalters mit der Computermaus wird ein Ereignis ausgelöst, das in den Objektdaten durch die Variable e bestimmt ist. Folgende Ereignisse sind möglich:

- `action`: In Kombination mit dem `move`-Befehl (Seite 288) kann die Figur in einen bestimmten Zustand versetzt werden.
- `reset`: Die Figur wird in den Anfangszustand zurückgesetzt.
- `help`: Ein Hilfetext wird angezeigt.
- `evaluate`: Die Antwortanalyse wird aufgerufen.
- `clearTrace`: Alle Ortsspuren werden gelöscht.

Bezeichner	Klasse	Unterklasse	Objektdaten
<Name [s]>	`measure`	`button`	<Beschriftung [s]>, <Ereignis e [s]>

Beispiel:

```
e[1] = b0;  measure;  button;  "Hilfe","help";
e[2] = b1;  measure;  button;  "Auswertung","evaluate";
e[3] = b2;  measure;  button;  "Ortsspur löschen","clearTrace";
```

3 Sonderfunktionen

3.1 Beschränken des Zustandsraums der Figur

Um den Zustandsraum einer Figur zu beschränken, ist die Angabe einer Liste `limit[1..n]` erforderlich. Jeder Listeneintrag definiert durch die Angabe einer Prüffunktion (Seite 284) einen Figurenzustand, den die Figur nicht einnehmen kann.
Beispiel:

```
limit[1] = "distance(A,B) < 20"
limit[2] = "isIncident(A,S)"
limit[3] = "isIncident(B,S)"
```

3.2 Ein- und Ausblenden von Objekten

Mit Hilfe einer Liste `hidden[1..n]` können Teile der Figur beim Eintreten bestimmter Figurenzustände ein- und ausgeblendet werden. Jeder Listeneintrag besteht aus der Angabe einer Prüffunktion (Seite 284) und einer Anzahl von Objektbezeichnern. Die angegebenen Objekte werden ausgeblendet, sobald die Prüffunktion den Wert 1 annimmt.
Beispiel:

```
hidden[1] = "if (not(isIncident(A,B))) hide (k,Textbox_1)"
hidden[2] = "if (calculate(f1) = 1) hide (A,B,C,D)"
```

3.3 Erzeugen von speziellen Figurenzuständen

Mit Hilfe einer Liste `move[1..n]` kann die Figur in einen bestimmten Zustand versetzt werden. Jeder Listeneintrag besteht aus der Angabe einer Prüffunktion (Seite 284), eines Punktobjekts und Paares von Zielkoordinaten. Der spezielle Figurenzustand wird hergestellt, sobald die Prüffunktion den Wert 1 annimmt. Dann werden dem angegebenen Punktobjekt die definierten Zielkoordinaten zugewiesen.
Beispiel:

```
move[1] = "if (calculate(Button1)) move (A, -7.0, 0.0)"
move[2] = "if (calculate(Button1)) move (B,  5.0, 0.0)"
move[3] = "if (calculate(Button1)) move (S,  2.0, 0.0)"
```

3.4 Hilfen

Abrufbare Hilfetexte werden in einem Skript in drei Schritten erzeugt:

1. Die Definition einer Hilfe beginnt mit dem Befehl `<Help>`.
2. In den folgenden Zeilen können beliebig viele Hilfeinformationen stehen.
3. Das Ende einer Hilfe wird mit `</Help>` festgelegt.

Beispiel:

```
<Help>
Der Schwerpunkt zweier Punktmassen teilt ihre
Verbindungsstrecke im umgekehrten Teilverhältnis.
</Help>
```

3.5 Textfenster

Ein Textfenster auf der Zeichenfläche wird in vier Schritten erzeugt:

1. Die Definition eines Textfensters beginnt mit `<TextBox>` (früher: `<ProblemText>` oder `<TextBoxBegin>`).
2. In der zweiten Zeile muß die Position auf der Zeichenfläche festgelegt werden. Dies geschieht durch `Position = x; y; b; h;` (früher: `TextBox = x; y; b; h;`). Dabei geben die Variablen `x` und `y` die Position der linken oberen Ecke in Fensterkoordinaten an. Die Variablen `b` und `h` legen die Breite und Höhe des Fensters in Bildschirmpunkten fest. Ist `b = -1` und `h = -1`, so wird die Fenstergröße automatisch berechnet.
3. Nach der Positionsangabe können optional Farbvariablen für die Beschriftungsfarbe, Rahmenfarbe und Füllfarbe angegeben werden. In den folgenden Zeilen stehen dann beliebig viele Textinformationen. Funktionale werden durch `{<"Term" [MC]>}` eingebunden.
4. Das Ende eines Textfensters wird mit `</TextBox>` (früher: `</ProblemText>` oder `<TextBoxEnd>`) angezeigt.

Beispiel:

```
<TextBox>
Position = 20;20;200;100
{nameColor = black}
{edgeColor = 255,225,255}
{faceColor = white}
 Dieses Textfenster wird auf
 der Zeichenfläche angezeigt.
 A = ({"coordinateX(A)"}, {"coordinateY(A)"})
</TextBox>
```

3.6 Bilder

Mit Hilfe des `image`-Befehls können Bilddateien eingebunden und auf der Zeichenfläche angezeigt werden. Neben dem Dateinamen ist dazu eine Koordinatenposition anzugeben. Diese kann absolut durch die Angabe von x- und y-Koordinaten definiert sein oder es wird ein Punkt angegeben, an dessen Position die Bilddatei dargestellt wird. Wird der Punkt bewegt, so verschiebt sich auch die Grafik.
Beispiel:

```
image[1] = "Formel01.gif", 20, 100
image[2] = "Formel02.gif", 70, 120
```

3.7 Antwortanalyse

Die Definition einer Antwortanalyse in einem Skript ist unterteilt in einen Hauptabschnitt und mehrere Unterabschnitte.

In dem Hauptabschnitt wird die Gesamthöchstzahl `MAX_ANSWER` von möglichen Antwortversuchen festgelegt. Wird der Wert auf 0 gesetzt, so hat der Schüler unbegrenzt viele Antwortversuche. Ferner ist im Hauptabschnitt eine Liste `condition[1..n]` von Prüffunktionen (Seite 284) anzugeben. Der Hauptabschnitt wird durch die beiden Befehle `<Problem>` und `</Problem>` umschlossen.
Beispiel:

```
<Problem>
    MAX_ANSWER = 3
    condition[1] = "calculate(r) < 0.26"
    condition[2] = "calculate(r) > 0.24"
    condition[3] = "calculate(r) < 0.51"
    condition[4] = "calculate(r) > 0.49"
</Problem>
```

In jedem Unterabschnitt wird jeweils ein Antwortwert und eine zugehörige Liste von Antwortkommentaren definiert. Zur Beschreibung eines Antwortwerts gehört die Angabe eines Prüfschlüssels `key`, der durch eine boolesche Verknüpfung der im Hauptabschnitt definierten Prüffunktionen festgelegt wird. Die Antwortkommentare werden in einer Liste `commment[1..h]` beschrieben. In einem Skript steht pro Kommentar nur eine Zeile zur Verfügung.[11] Bei der Ausgabe eines Kommentars wird ein Zeilenumbruch durch den Befehl `/n` bewirkt. Jeder Unterabschnitt zur Definition eines Antwortwerts wird durch die beiden Befehle `<Answer i>` und `</Answer i>` umschlossen.
Beispiel:

```
<Answer 1>
  key = "condition[1] AND condition[2]"
  comment[1] = "Richtig. Der Eckenschwerpunkt teilt die
                Verbindungsstrecke zweier Punktmassen im
                umgekehrten Verhältnis der anliegenden Massen."
```

[11] In dem unten aufgeführten Beispiel sind aus Platzgründen Zeilenumbrüche eingefügt.

```
</Answer 1>

<Answer 2>
  key = "condition[3] AND condition[4]"
  comment[1] = "Ihre Antwort ist nicht richtig. Der Eckenschwer-
               punkt würde nur dann genau in der Mitte zwischen
               zwei Punktmassen liegen, wenn diese gleich groß
               wären. Versuchen Sie es noch einmal."
  comment[2] = "Ihre Antwort ist immer noch nicht richtig.
               Lösung: Der Eckenschwerpunkt teilt die Verbin-
               dungsstrecke zweier Punktmassen im umgekehrten
               Verhältnis der anliegenden Massen."
</Answer 2>

<Answer 3>
  key = "1"
  comment[1] = "Ihre Antwort ist nicht richtig. Versuchen Sie es
               noch einmal."
  comment[2] = "Ihre Antwort ist immer noch nicht richtig.
               Lösung: Der Eckenschwerpunkt teilt die Verbin-
               dungsstrecke zweier Punktmassen im umgekehrten
               Verhältnis der anliegenden Massen."
</Answer 3>
```

3.8 Animationen

Durch den `animation`-Befehl kann eine Figur automatisch bewegt werden. Dazu ist ein Punkt der Klasse `Dragable` anzugeben. Das Bezugsobjekt von diesem Punkt muß dabei entweder eine Strecke, eine Gerade, ein Kreis, ein Polygon oder eine Kurve sein und dient als Führungslinie. Ferner ist die Angabe einer Bewegungsrichtung (`1` oder `0`), einer Geschwindigkeitsverzögerung (in msec) und einer Anzahl von Animationsschritten erforderlich. Jede Animation wird durch einen (automatisch erzeugten) Schalter ein- und ausgeschaltet. Soll die Animation gleich beim Aufrufen der Figur gestartet werden, so ist als letzter Parameter der Wert `1` anzugeben, falls nicht, der Wert `0`.
Beispiel:

```
  animation = "A,0,10,200,0"
```

4 Systemeinstellungen

4.1 Variablen

Mit Hilfe der Systemvariablen können die globalen Einstellungen von *Geometria* festgelegt werden. In der Regel sind diese Variablen in der Layout-Vorlage definiert. Um Ausnahmen in einzelnen Skripten zu realisieren, können die Systemvariablen aber auch in jedem Skript überschrieben werden.

Systemvariable	Default-Wert	Beschreibung
`APPLET_WIDTH` = <[i]>	640	Breite der Zeichenfläche in Bildschirmpunkten
`APPLET_HEIGHT` = <[i]>	480	Höhe der Zeichenfläche in Bildschirmpunkten
`WORLD_X_MAX` = <[d]>	+16.0	Größter x-Wert im Weltkoordinatensystem
`WORLD_X_MIN` = <[d]>	−16.0	Kleinster x-Wert im Weltkoordinatensystem
`WORLD_Y_MAX` = <[d]>	+12.0	Größter y-Wert im Weltkoordinatensystem
`WORLD_Y_MIN` = <[d]>	−12.0	Kleinster y-Wert im Weltkoordinatensystem
`FONT` = <[s]>	`Serif`	Schriftart
`FONTSIZE` = <[i]>	10	Schriftgröße

Systemvariable	Default-Wert	Beschreibung
`GRIDSIZE` = <[i]>	10	Rasterabstand in Bildschirmpunkten
`GRIDCOLOR` = <[c]>	235, 205, 180	Farbe des Hintergrundrasters
`BACKGROUNDCOLOR` = <[c]>	235, 225, 200	Farbe der Zeichenfläche
`APPLETBGCOLOR` = <[c]>	`black`	Farbe des Applethintergrunds
`CONTROLPANELCOLOR` = <[c]>	`black`	Farbe des Hintergrunds der Interaktionselemente
`TITLE` = <[i]>	Name der Skriptdatei	Titel des Betrachterfensters
`SHOWLABEL` = <[b]>	`false`	Beschriftungen anzeigen
`SHOWGRID` = <[b]>	`false`	Hintergundraster anzeigen
`SHOWAXIS` = <[b]>	`false`	Koordinatenachsen anzeigen
`SNAPTOGRID` = <[b]>	`false`	Rasterfangmodus einschalten
`ALLPOINTSDRAGABLE` = <[b]>	`false`	Verschieben nichtziehbarer Punkte
`LANGUAGE` = <[s]>	`GERMAN`	Beschriftungssprache (`GERMAN` oder `ENGLISH`)
`USESEPARATEWINDOW` = <[b]>	`true`	Zeichenfläche in separatem Fenster darstellen

Systemvariable	Default-Wert	Beschreibung
`HIDEMENU(`<[s]>`)`		Ausblenden eines Menüs (Menübezeichner angeben)
`HIDEMENUITEM(`<[s]>`)`		Ausblenden eines Menüpunkts (Menüpunktbezeichner angeben)
`MEASURE_EXACTNESS` = <[i]>	3	Nachkommastellen bei Funktionalen
`CHECKSYMBOLS` = <[b]>	`false`	Griechische Buchstaben darstellen
`DRAGMEASURE` = <[b]>	`false`	Funktionale auf der Zeichenfläche ziehbar
`INERTUPDATEMODE` = <[b]>	`true`	Trägheitsmodus in der `update`-Methode einschalten

4.2 Farbdefinitionen

Um Farben zu definieren, sind zwei alternative Wege vorgesehen. Als erstes kann eine Farbe durch die Angabe eines Zahlentripels `r`, `g`, `b` definiert werden, mit $0 \leq$ `r`, `g`, `b` ≤ 255. Dabei bestimmt `r` den Rotanteil, `g` den Grünanteil und `b` den Blauanteil einer Farbe.

Die zweite Möglichkeit besteht darin, einen der vordefinierten Farbbezeichner zu verwenden: `black` (schwarz), `blue` (blau), `cyan` (türkisblau), `darkGray` (dunkelgrau), `gray` (grau), `green` (grün), `lightGray` (hellgrau), `magenta` (violett), `orange` (orange), `pink` (rosa), `red` (rot), `white` (weiß), `yellow` (gelb), `random` (Zufallsfarbe), `brighter` (heller als der Hintergrund) und `darker` (dunkler als der Hintergrund).

Soll eine Farbe nicht definiert werden, so ist der Wert `0` anzugeben.

4.3 Punktformen

Um Punkte auf der Zeichenfläche darzustellen, kann zwischen sechs vordefinierten Formen gewählt werden: `smallsquare` (kleines Quadrat), `square` (mittleres Quadrat), `bigsquare` (großes Quadrat), `smallcircle` (kleiner Kreis), `circle` (mittlerer Kreis) und `bigcircle` (großer Kreis). Die Angabe von `default` bewirkt, daß die in der aktuellen Layout-Vorlage festgelegte Punktform verwendet wird.

4.4 Sonderzeichen

Alle Beschriftungen oder Objektbezeichner können Buchstaben des griechischen Alphabets enthalten. Um den Parser zu veranlassen, gezielt nach Sonderzeichen zu suchen, ist es erforderlich, daß die Systemvariable `CHECKSYMBOLS` in dem betreffenden Skript auf `true` gesetzt wird.

Es können folgenden Zeichen erzeugt werden: `\alpha` (α), `\beta` (β), `\gamma` (γ), `\delta` (δ), `\epsilon` (ϵ), `\zeta` (ζ), `\eta` (η), `\theta` (θ), `\jota` (ι), `\kappa` (κ), `\lambda` (λ), `\my` (μ), `\ny` (ν), `\xi` (ξ), `\omikron` (o), `\pi` (π), `\rho` (ρ), `\sigma` (σ), `\tau` (τ), `\ypsilon` (υ), `\phi` (ϕ), `\chi` (χ), `\psi` (ψ), `\omega` (ω), `\ALPHA` (A), `\BETA` (B), `\GAMMA` (Γ), `\DELTA` (Δ), `\EPSILON` (E), `\ZETA` (Z), `\ETA` (H), `\THETA` (Θ), `\JOTA` (I), `\KAPPA` (K), `\LAMBDA` (Λ), `\MY` (M), `\NY` (N), `\XI` (Ξ), `\OMIKRON` (O), `\PI` (Π), `\RHO` (P), `\SIGMA` (Σ), `\TAU` (T), `\YPSILON` (Υ), `\PHI` (Φ), `\CHI` (X), `\PSI` (Ψ) und `\OMEGA` (Ω).

4.5 Applet-Parameter

Beim Einbinden des Applets *Geometria* in ein HTML-Dokument können folgende Parameter übergeben werden.

- `script`: Bezeichnet den Dateinamen des Skripts, das eingelesen werden soll.
- `style`: Bezeichnet den Dateinamen der Layout-Vorlage, die eingelesen werden soll.
- `backgroundColor`: Legt die Farbe des Applet-Hintergrunds fest.
- `startButton`: Legt fest, ob ein Button erzeugt werden soll, durch den man *Geometria* startet, oder ob das Applet direkt ausgeführt werden soll.
- `scriptPath`: Legt das Verzeichnis fest, in dem die Skriptdatei gespeichert ist.
- `stylePath`: Bestimmt das Verzeichnis, in dem die Layout-Vorlage abgelegt ist.
- `imagePath`: Gibt das Verzeichnis an, in dem alle Bilddateien (s. Seite 290) gespeichert sind.

Beispiel:

```
<applet code="Geometria" codebase="" archive="Geometria.jar"
        height="395" width="480">
  <param name="script"          value="demo.script">
  <param name="style"           value="default.style">
  <param name="startButton"     value="0">
  <param name="backgroundColor" value="100,23,00">
  <param name="scriptPath"      value="./script/">
  <param name="stylePath"       value="./style/">
  <param name="imagePath"       value="./image/">
</applet>
```

Abkürzungsverzeichnis

[b]	boolescher Wert	(true, false)
[c]	Farbdefinition	(Seite 294)
[d]	reelle Zahl	(z. B. -1.2)
[i]	ganze Zahl	(z. B. 1, -2)
[s]	Zeichenkette	("Text")
[C]	Kreisobjekt	(Seite 264)
[SE]	Kreisbogenobjekt	(Seite 265)
[CO]	Koordinatensystem	(Seite 270)
[CU]	Kurvenobjekt	(Seite 260)
[M]	Funktional	(Seite 271)
[MC]	Term	(Seite 283)
[MCond]	Prüffunktion	(Seite 284)
[L]	Strecken-, Strahlen- oder Geradenobjekt	(Seite 256)
[LO]	Ortslinienobjekt	(Seite 262)
[P]	Punktobjekt	(Seite 242)
[PG]	Polygonobjekt	(Seite 266)
[PS]	Punktmengenobjekt	(Seite 269)
[R]	Strahlenobjekt	(Seite 257)
[ST]	Geradenobjekt	(Seite 257)

Anhang C
Skripte aller Beispielfiguren

Beispielfigur Zugmodus (Seite 49)

```
//
// Datei: Beispielfigur_Zugmodus.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
//

// Figurenbeschreibung
// ===================

e[1]  = A; point;  free;         -5.0,1.0;
e[2]  = B; point;  free;         5.0,-1.0;
e[3]  = M; point;  midpoint;     A,B;
e[4]  = k; circle; radius;       M,M,A;     "hideLabel";
e[5]  = C; point;  circleSlider; 1.0,6.0,k;
e[6]  = a; line;   connect;      C,B;
e[7]  = b; line;   connect;      A,C;
e[8]  = c; line;   connect;      A,B;
```

Affine Abbildungen (Seite 97)

```
//
// Datei: Affine_Abbildungen.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
//

// Systemvariablen
// ===============

APPLET_WIDTH  = 640
APPLET_HEIGHT = 480

// Figurenbeschreibung
// ===================

e[1]  = O;     point;  fixed;           0.0,0.0;                        "hidden"
e[2]  = coord; point;  coordSystem;     0,300,300,200,200;              0;red;black;0
e[3]  = A;     point;  free;            3.0,3.0;
e[4]  = B;     point;  free;            -5.0,5.0;
e[5]  = C;     point;  free;            -4.0,-2.0;
e[6]  = D;     point;  free;            4.0,-2.0;
e[7]  = a1;    measure; controller;     1.0,0.25,-3.0,3.0,50,"a1 = ","";
e[8]  = a2;    measure; controller;     0.0,0.25,-3.0,3.0,50,"a2 = ","";
e[9]  = b1;    measure; controller;     0.0,0.25,-3.0,3.0,50,"b1 = ","";
e[10] = b2;    measure; controller;     -1.0,0.25,-3.0,3.0,50,"b2 = ","";
e[11] = v1;    measure; controller;     0.0,0.25,-3.0,3.0,50,"v1 = ","";
e[12] = v2;    measure; controller;     0.0,0.25,-3.0,3.0,50,"v2 = ","";
e[13] = P1;    polygon; quadrilateral;  A,B,C,D;                        0;0;red;0
e[14] = A';    point;   functionDepend;
               "coordinateX(A)*calculate(a1)+coordinateY(A)*calculate(b1)+calculate(v1)",
               "coordinateX(A)*calculate(a2)+coordinateY(A)*calculate(b2)+calculate(v2)";
e[15] = B';    point;   functionDepend;
               "coordinateX(B)*calculate(a1)+coordinateY(B)*calculate(b1)+calculate(v1)",
               "coordinateX(B)*calculate(a2)+coordinateY(B)*calculate(b2)+calculate(v2)";
e[16] = C';    point;   functionDepend;
               "coordinateX(C)*calculate(a1)+coordinateY(C)*calculate(b1)+calculate(v1)",
               "coordinateX(C)*calculate(a2)+coordinateY(C)*calculate(b2)+calculate(v2)";
e[17] = D';    point;   functionDepend;
               "coordinateX(D)*calculate(a1)+coordinateY(D)*calculate(b1)+calculate(v1)",
               "coordinateX(D)*calculate(a2)+coordinateY(D)*calculate(b2)+calculate(v2)";
e[18] = P2;    polygon; quadrilateral;  A',B',C',D';                    0;0;blue;0

// Textfenster
// ===========

<Textbox>
 Position = 10;10;-1;-1
  Visualisierung der allgemeinen Abbildung
  f: R^2 -> R^2 mit
  f(x,y) := (a1x + b1y + v1, a2x + b2y + v2).
</Textbox>

<Textbox>
 Position = 340;50;260;-1
  Das Viereck ABCD wird nach der
  Abbildungsvorschrift
  f(x,y) := ({calculate(a1)}x + {calculate(b1)}y + {calculate(v1)},
             {calculate(a2)}x + {calculate(b2)}y + {calculate(v2)})
  auf das Viereck A'B'C'D' abgebildet.
</Textbox>
```

Satz von Pythagoras (Seite 99)

```
//
// Datei: Satz_von_Pythagoras.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
//

// Figurenbeschreibung
// ===================

e[1]  = A;   point;   free;           -10.95,-1.0;
e[2]  = B;   point;   horizontal;     A,-5.65;
e[3]  = M;   point;   midpoint;       A,B;                    "hidden"
e[4]  = k;   circle;  radius;         M,M,A;                  "hidden"
e[5]  = C;   point;   circleSlider;   -9.31,3.15,k;
e[6]  = a;   line;    connect;        C,B;
e[7]  = b;   line;    connect;        A,C;
e[8]  = c;   line;    connect;        A,B;
e[9]  = P1;  point;   rotation;       A,C,1.57,1.0;           "hidden"
e[10] = P2;  point;   rotation;       C,A,-1.57,1.0;          "hidden"
e[11] = P3;  point;   rotation;       B,C,-1.57,1.0;          "hidden"
e[12] = P4;  point;   rotation;       C,B,1.57,1.0;           "hidden"
e[13] = P5;  point;   rotation;       B,A,1.57,1.0;           "hidden"
e[14] = P6;  point;   rotation;       A,B,-1.57,1.0;          "hidden"
e[15] = F;   point;   foot;           C,c;                    "hidden"
e[16] = P8;  point;   intersection;   P1,P2,C,F;              "hidden"
e[17] = p1;  polygon; hexagon;        P1,P8,C,F,C,P8;         "hidden"
e[18] = P;   point;   polygonSlider;  -11.76,4.79,p1;
e[19] = P9;  point;   translation;    P,C,A;                  "hidden"
e[20] = r1;  polygon; quadrilateral;  P,P9,A,C;               0;0;black;lightGray
e[21] = P10; point;   translation;    P,P8,C;                 "hidden"
e[22] = P11; point;   translation;    P9,P8,C;                "hidden"
e[23] = r2;  polygon; quadrilateral;  P,P9,P11,P10;           0;0;black;lightGray
e[24] = P12; point;   translation;    P,A,P5;                 "hidden"
e[25] = r3;  polygon; quadrilateral;  P,A,P5,P12;             0;0;black;lightGray
e[26] = s1;  line;    connect;        P1,P8;                  "hidden"
e[27] = s2;  line;    connect;        P8,C;                   "hidden"
e[28] = s3;  line;    connect;        C,F;                    "hidden"
e[29] = P13; point;   proportion;     P8,P1,P8,P,P8,P3,P8,P3; "hidden"
e[30] = m0;  measure; incidence;      P,s1,-100.0,-1.0,"","";
e[31] = P';  point;   functionDepend;
             "if (calculate(m0)) then (coordinateX(P13)) else (coordinateX(P))",
             "if (calculate(m0)) then (coordinateY(P13)) else (coordinateY(P))"; "hidden"
e[32] = P14; point;   translation;    P',C,B;                 "hidden"
e[33] = r4;  polygon; quadrilateral;  P',C,B,P14;             0;0;black;gray
e[34] = P15; point;   translation;    P',P8,C;                "hidden"
e[35] = P16; point;   translation;    P14,P8,C;               "hidden"
e[36] = r5;  polygon; quadrilateral;  P',P15,P16,P14;         0;0;black;gray
e[37] = P17; point;   translation;    P',B,P6;                "hidden"
e[38] = r6;  polygon; quadrilateral;  P',P17,P6,B;            0;0;black;gray
e[39] = a^2; polygon; quadrilateral;  B,P4,P3,C;
e[40] = b^2; polygon; quadrilateral;  C,P1,P2,A;
e[41] = c^2; polygon; quadrilateral;  A,P5,P6,B;

// Ein- und Ausblenden von Objekten
// ================================

hidden[1] = "if (not(isIncident(P,s1))) hide (r1,r4,Textbox_1)"
hidden[2] = "if (not(isIncident(P,s2))) hide (r2,r5,Textbox_2)"
hidden[3] = "if (not(isIncident(P,s3))) hide (r3,r6,Textbox_3)"

// Textfenster
// ===========

<TextBox>
 Position = 20;20;300;-1
  Verschiebe den Punkt P parallel zur Seite AC und
  transformiere so das hellgraue Quadrat in ein
  Parallelogramm mit gleichem Flächeninhalt.
</TextBox>
```

```
<TextBox>
 Position = 20;20;300;-1
  Verschiebe den Punkt P in Richtung auf C.
  Die Form und der Flächeninhalt der beiden
  Parallelogramme ändert sich dabei nicht.
</TextBox>

<TextBox>
 Position = 20;20;300;-1
  Verschiebe den Punkt P lotrecht auf die Seite AB.
  Die beiden Parallelogramme werden dadurch in ein
  Rechteck transformiert. Ihr Flächeninhalt ändert
  sich dabei nicht.
</TextBox>
```

Eckenschwerpunkt im Dreieck (Seite 100)

```
//
// Datei: Eckenschwerpunkt_im_Dreieck.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
//

// Figurenbeschreibung
// ===================

e[1]  = A;    point;   fixed;          -2.0,6.05;         black;red;black;smallcircle
e[2]  = B;    point;   free;           -9.0,6.0;          black;red;black;smallcircle
e[3]  = C;    point;   free;           -9.95,-1.05;       black;red;black;smallcircle
e[4]  = d;    polygon; triangle;       A,B,C;             0;0;blue;0
e[5]  = M1;   point;   midpoint;       A,B;               "hidden"
e[6]  = M2;   point;   midpoint;       C,B;               "hidden"
e[7]  = M3;   point;   midpoint;       C,A;               "hidden"
e[8]  = m4;   line;    connect;        A,B;               "hidden"
e[9]  = m1;   line;    connect;        M1,C;              0;0;lightGray;0
e[10] = m2;   line;    connect;        M2,A;              "hidden"
e[11] = m3;   line;    connect;        B,M3;              "hidden"
e[12] = S;    point;   intersection;   m1,m3;             0;lightGray;lightGray;smallcircle
e[13] = P1;   point;   midpoint;       M1,S;              0;lightGray;lightGray;smallcircle
e[14] = P2;   point;   midpoint;       C,S;               0;lightGray;lightGray;smallcircle
e[15] = P3;   point;   midpoint;       P2,S;              0;lightGray;lightGray;smallcircle
e[16] = P4;   point;   midpoint;       P2,C;              0;lightGray;lightGray;smallcircle
e[17] = p1;   polygon; quadrilateral;  A,M1,S,M1;         "hidden"
e[18] = A';   point;   polygonSlider;  p1,3.0, 3.0;       0;red;black;smallcircle
e[19] = me4;  measure; calculate;      "if (isIncident(A',m4))
              then (2*coordinateX(M1)-coordinateX(A')) else (coordinateX(A'))";
e[20] = me5;  measure; calculate;      "if (isIncident(A',m4))
              then (2*coordinateY(M1)-coordinateY(A')) else (coordinateY(A'))";
e[21] = B';   point;   functionDepend;
              "calculate(me4)","calculate(me5)";          0;red;black;smallcircle
e[22] = me6;  measure; calculate;      "if (isIncident(A',m4))
              then (coordinateX(C)) else (3*XVector(A',S)+coordinateX(A'))";
e[23] = me7;  measure; calculate;      "if (isIncident(A',m4))
              then (coordinateY(C)) else (3*YVector(A',S)+coordinateY(A'))";
e[24] = C';   point;   functionDepend;
              "calculate(me6)","calculate(me7)";          0;red;black;smallcircle
e[25] = me8;  measure; calculate;      "if (isIncident(A',B')) then (coordinateX(A'))
              else (-100.0)";
e[26] = me9;  measure; calculate;      "if (isIncident(A',B')) then (coordinateY(A'))
              else (-100.0)";
e[27] = A'';  point;   functionDepend;
              "calculate(me8)","calculate(me9)";          0;red;black;circle
e[28] = A''';point;   functionDepend; "$(S,x)","$(S,y)"; 0;red;black;bigcircle

// Ein- und Ausblenden von Objekten
// ================================

hidden[1] = "if (not(isIncident(A'',S))) hide (A''')"
hidden[2] = "if (not(isIncident(A',m1))) hide (m1,P1,P2,P3,P4,S,Textbox_3)"
hidden[3] = "if (not(isIncident(A'',S))) hide (Textbox_4)"

// Textfenster
// ===========

<Textbox>
 Textbox = 10;10;230;30
  Der Eckenschwerpunkt im Dreieck
</Textbox>

<Textbox>
 Textbox = 10;290;310;60
   Verschieben Sie die Masse in A in Richtung
   auf B.
</Textbox>

<Textbox>
 Textbox = 10;290;310;60
```

```
   Der Schwerpunkt der beiden Massen in A und
   B ist im Mittelpunkt der Seite AB. Verschieben
   Sie die Zweifach-Masse in Richtung auf C.
</Textbox>

<Textbox>
 Textbox = 10;290;310;60
   Der Schwerpunkt der Zweifach-Masse und der
   Masse in C ist der Eckenschwerpunkt des Dreiecks.
   Er teilt die Mittellinie im Verhältnis 2 : 1.
</Textbox>
```

Schachtelvolumen (Seite 102)

```
//
// Datei: Schachtelvolumen.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
//

// Systemvariablen
// ===============

APPLET_WIDTH  = 640
APPLET_HEIGHT = 480
WORLD_X_MAX   = +16.0
WORLD_X_MIN   = -16.0
WORLD_Y_MAX   = +12.0
WORLD_Y_MIN   = -12.0
MEASURE_EXACTNESS = 1

// Figurenbeschreibung
// ===================

e[1]  = A;   point;   fixed;             -4.0, 1.0;            "hidden"
e[2]  = B;   point;   fixed;             -4.0,11.0;            "hidden"
e[3]  = C;   point;   fixed;             -14.0,11.0;           "hidden"
e[4]  = D;   point;   fixed;             -14.0, 1.0;           "hidden"
e[5]  = M;   point;   midpoint;          A,B;                  "hidden"
e[6]  = a';  line;    connect;           M,A;                  "hidden"
e[7]  = P;   point;   lineSegmentSlider; -4.0,4.1,a';
e[8]  = m1;  measure; YVector;           A,P,-100.0,-100.0,"","";
e[9]  = m2;  measure; YVector;           P,A,-100.0,-100.0,"","";
e[10] = m3;  measure; calculate;         "0.0",-100.0,-100.0,"","";
e[11] = P1;  point;   translation;       B,m3,m2;              "hidden"
e[12] = P2;  point;   translation;       B,m2,m2;              "hidden"
e[13] = P3;  point;   translation;       B,m2,m3;              "hidden"
e[14] = P4;  point;   translation;       C,m1,m3;              "hidden"
e[15] = P5;  point;   translation;       C,m1,m2;              "hidden"
e[16] = P6;  point;   translation;       C,m3,m2;              "hidden"
e[17] = P7;  point;   translation;       D,m3,m1;              "hidden"
e[18] = P8;  point;   translation;       D,m1,m1;              "hidden"
e[19] = P9;  point;   translation;       D,m1,m3;              "hidden"
e[20] = P10; point;   translation;       A,m2,m3;              "hidden"
e[21] = P11; point;   translation;       A,m2,m1;              "hidden"
e[22] = p0;  polygon; quadrilateral;     A,B,C,D;              0;0;black;yellow
e[23] = p1;  polygon; quadrilateral;     P,P1,P6,P7;           0;0;black;lightGray
e[24] = p2;  polygon; quadrilateral;     P9,P10,P3,P4;         0;0;black;lightGray
e[25] = p3;  polygon; quadrilateral;     P2,P5,P8,P11;         0;0;black;gray
e[26] = Q1;  point;   free;              -0.15,1.6;            "hideLabel"
e[27] = Q2;  point;   translation;       Q1,P9,P10;            "hidden"
e[28] = Q3;  point;   translation;       Q1,A,P;               "hidden"
e[29] = Q4;  point;   translation;       Q2,A,P;               "hidden"
e[30] = Q5;  point;   rotation;          Q1,Q2,2.35619449,0.5; "hidden"
e[31] = Q6;  point;   translation;       Q5,A,P;               "hidden"
e[32] = Q7;  point;   translation;       Q3,Q2,Q5;             "hidden"
e[33] = Q8;  point;   translation;       Q1,Q2,Q5;             "hidden"
e[34] = q0;  polygon; quadrilateral;     Q1,Q2,Q5,Q8;          0;0;black;gray
e[35] = q1;  polygon; quadrilateral;     Q5,Q6,Q7,Q8;          0;0;black;lightGray
e[36] = q2;  polygon; quadrilateral;     Q1,Q3,Q7,Q8;          0;0;black;lightGray
e[37] = q3;  polygon; quadrilateral;     Q1,Q2,Q4,Q3;          0;0;black;green
e[38] = q4;  polygon; quadrilateral;     Q2,Q5,Q6,Q4;          0;0;black;green
e[39] = m4;  measure; calculate;        "distance(A,P)*distance(P,P1)^2",-2.0,10.0,
           "Schachtelvolumen = b * b * x = ","";
e[40] = m5;  measure; calculate;         "-1.0",-100.0,-100.0,"","";
e[41] = R1;  point;   translation;       P9,m3,m5;             0;black;black;smallcircle
e[42] = R2;  point;   translation;       P10,m3,m5;            0;black;black;smallcircle
e[43] = R3;  point;   fixed;             -15.0,11.0;           0;black;black;smallcircle
e[44] = R4;  point;   fixed;             -15.0, 1.0;           0;black;black;smallcircle
e[45] = R5;  point;   fixed;             -3.0, 1.0;            0;black;black;smallcircle
e[46] = R6;  point;   translation;       R5,A,P;               0;black;black;smallcircle
e[47] = x;   line;    connect;           R5,R6;                black;0;black;0
e[48] = b;   line;    connect;           R1,R2;                black;0;black;0
e[49] = a;   line;    connect;           R3,R4;                black;0;black;0
```

```
e[50] = m6;  measure; distance;          A,P,-2.0,7.0,"x = ","";
e[51] = m7;  measure; distance;          A,B,-2.0,9.0,"a = ","";
e[52] = m8;  measure; distance;          P,P1,-2.0,8.0,"b = ","";
e[53] = O;   point;   free;              4.0,-11.0;            "hidden"
e[54] = Ox;  point;   translation;       O,2.0,0.0;            "hidden"
e[55] = Oy;  point;   translation;       O,0.0,0.15;           "hidden"
e[56] = coord;point;  coordSystem;       O,O,Ox,O,Oy,10,220,10,220; 0;red;black;0
e[57] = C1;  point;   functionDepend;    "O","calculate(m4)", coord; "hidden"
e[58] = C2;  point;   functionDepend;    "calculate(m6)","O", coord; "hidden"
e[59] = C3;  point;   functionDepend;    "calculate(m6)","calculate(m4)", coord; "hideLabel"
e[60] = c1;  line;    connect;           C1,C3;                0;0;lightGray;0
e[61] = c2;  line;    connect;           C2,C3;                0;0;lightGray;0
e[62] = locus;line;   locus;             C3,P,a',150;          0;red;black;0

// Textfenster
// ===========
<ProblemText>
 Position = 20;280;-1;-1
   Aufgabe:

   Man betrachte das Quadrat mit der Seitenlänge a = 10,
   an dessen Ecken gleich große Quadrate ausgeschnitten
   werden. Durch Auffalten bekommt man eine nach oben
   offene Schachtel mit der Breite b und der Höhe x.

   Durch Verschieben von Punkt P kann das Netz der
   Schachtel verändert werden.
   Gibt es eine besondere unter den Schachteln?
</ProblemText>
```

Volladdierer (Seite 103)

```
//
// Datei: Volladdierer.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
//

// Systemvariablen
// ===============

APPLET_WIDTH  = 640
APPLET_HEIGHT = 480
showLabel = false

// Figurenbeschreibung
// ===================

e[1]   = A1;   point;   fixed;       -10.0,10.0;             "hidden"
e[2]   = A2;   point;   fixed;       10.0,10.0;              "hidden"
e[3]   = B1;   point;   fixed;       -10.0,9.0;              "hidden"
e[4]   = B2;   point;   fixed;       10.0,9.0;               "hidden"
e[5]   = C1;   point;   fixed;       -10.0,8.0;              "hidden"
e[6]   = C2;   point;   fixed;       10.0,8.0;               "hidden"
e[7]   = s1;   line;    connect;     A1,A2;                  black;0;gray;0
e[8]   = s2;   line;    connect;     B1,B2;                  black;0;gray;0
e[9]   = s3;   line;    connect;     C1,C2;                  black;0;gray;0
e[10]  = P1;   point;   fixed;       -7.5,10.0;              0;gray;gray;smallcircle
e[11]  = P2;   point;   fixed;       -7.5,-3.0;              "hidden"
e[12]  = P3;   point;   fixed;       -6.5,9.0;               0;gray;gray;smallcircle
e[13]  = P4;   point;   fixed;       -6.5,-3.0;              "hidden"
e[14]  = P5;   point;   fixed;       -2.0,10.0;              0;gray;gray;smallcircle
e[15]  = P6;   point;   fixed;       -2.0,5.0;               "hidden"
e[16]  = P7;   point;   fixed;       -1.0,9.0;               0;gray;gray;smallcircle
e[17]  = P8;   point;   fixed;       -1.0,5.0;               0;white;black;circle
e[18]  = P9;   point;   fixed;       1.0,10.0;               0;gray;gray;smallcircle
e[19]  = P10;  point;   fixed;       1.0,5.0;                0;white;black;circle
e[20]  = P11;  point;   fixed;       2.0,9.0;                0;gray;gray;smallcircle
e[21]  = P12;  point;   fixed;       2.0,5.0;                "hidden"
e[22]  = P13;  point;   fixed;       4.5,8.0;                0;gray;gray;smallcircle
e[23]  = P14;  point;   fixed;       4.5,-3.0;               "hidden"
e[24]  = P15;  point;   fixed;       7.5,8.0;                0;gray;gray;smallcircle
e[25]  = P16;  point;   fixed;       7.5,-3.0;               0;white;black;circle
e[26]  = P17;  point;   fixed;       -1.5,4.0;               "hidden"
e[27]  = P18;  point;   fixed;       -1.5,3.0;               "hidden"
e[28]  = P19;  point;   fixed;       -0.5,3.0;               "hidden"
e[29]  = P20;  point;   fixed;       -0.5,2.0;               "hidden"
e[30]  = P31;  point;   fixed;       1.5,4.0;                "hidden"
e[31]  = P32;  point;   fixed;       1.5,3.0;                "hidden"
e[32]  = P33;  point;   fixed;       0.5,3.0;                "hidden"
e[33]  = P34;  point;   fixed;       0.5,2.0;                "hidden"
e[34]  = P35;  point;   fixed;       -7.0,-4.0;              "hidden"
e[35]  = P36;  point;   fixed;       -7.0,-5.0;              "hidden"
e[36]  = P37;  point;   fixed;       -6.0,-5.0;              "hidden"
e[37]  = P38;  point;   fixed;       -6.0,-6.0;              "hidden"
e[38]  = P39;  point;   fixed;       -4.0,-4.0;              "hidden"
e[39]  = P40;  point;   fixed;       -4.0,-5.0;              "hidden"
e[40]  = P41;  point;   fixed;       -5.0,-5.0;              "hidden"
e[41]  = P42;  point;   fixed;       -5.0,-6.0;              "hidden"
e[42]  = P43;  point;   fixed;       4.0,-4.0;               "hidden"
e[43]  = P44;  point;   fixed;       4.0,-5.0;               "hidden"
e[44]  = P45;  point;   fixed;       5.0,-5.0;               "hidden"
e[45]  = P46;  point;   fixed;       5.0,-6.0;               "hidden"
e[46]  = P47;  point;   fixed;       7.0,-4.0;               "hidden"
e[47]  = P48;  point;   fixed;       7.0,-5.0;               "hidden"
e[48]  = P49;  point;   fixed;       6.0,-5.0;               "hidden"
e[49]  = P50;  point;   fixed;       6.0,-6.0;               "hidden"
e[50]  = P51;  point;   fixed;       0.0,1.0;                "hidden"
e[51]  = P52;  point;   fixed;       0.0,-1.0;               0;gray;gray;smallcircle
e[52]  = P53;  point;   fixed;       -3.5,-1.0;              "hidden"
e[53]  = P54;  point;   fixed;       -3.5,-3.0;              "hidden"
e[54]  = P55;  point;   fixed;       -4.5,-3.0;              "hidden"
```

```
e[55]  = P56;  point;   fixed;       -4.5,8.0;                0;gray;gray;smallcircle
e[56]  = P57;  point;   fixed;       3.5,-1.0;                0;gray;gray;smallcircle
e[57]  = P58;  point;   fixed;       3.5,-3.0;                0;white;black;circle
e[58]  = P59;  point;   fixed;       6.5,-1.0;                0;gray;gray;smallcircle
e[59]  = P60;  point;   fixed;       6.5,-3.0;                "hidden"
e[60]  = P61;  point;   fixed;       -5.5,-7.0;               "hidden"
e[61]  = P62;  point;   fixed;       -5.5,-9.0;               "hidden"
e[62]  = P63;  point;   fixed;       5.5,-7.0;                "hidden"
e[63]  = P64;  point;   fixed;       5.5,-9.0;                "hidden"
e[64]  = s4;   line;    connect;     P1,P2;                   black;0;gray;0
e[65]  = s5;   line;    connect;     P3,P4;                   black;0;gray;0
e[66]  = s6;   line;    connect;     P5,P6;                   black;0;gray;0
e[67]  = s7;   line;    connect;     P7,P8;                   black;0;gray;0
e[68]  = s8;   line;    connect;     P9,P10;                  black;0;gray;0
e[69]  = s9;   line;    connect;     P11,P12;                 black;0;gray;0
e[70]  = s10;  line;    connect;     P13,P14;                 black;0;gray;0
e[71]  = s11;  line;    connect;     P15,P16;                 black;0;gray;0
e[72]  = s12;  polygon; hexagon;     P17,P18,P19,P20,P19,P18; black;0;gray;0
e[73]  = s13;  polygon; hexagon;     P31,P32,P33,P34,P33,P32; black;0;gray;0
e[74]  = s14;  polygon; hexagon;     P35,P36,P37,P38,P37,P36; black;0;gray;0
e[75]  = s15;  polygon; hexagon;     P39,P40,P41,P42,P41,P40; black;0;gray;0
e[76]  = s16;  polygon; hexagon;     P43,P44,P45,P46,P45,P44; black;0;gray;0
e[77]  = s17;  polygon; hexagon;     P47,P48,P49,P50,P49,P48; black;0;gray;0
e[78]  = s18;  polygon; hexagon;     P51,P52,P53,P54,P53,P52; black;0;gray;0
e[79]  = s19;  polygon; hexagon;     P51,P52,P57,P58,P57,P52; black;0;gray;0
e[80]  = s20;  polygon; hexagon;     P51,P52,P59,P60,P59,P52; black;0;gray;0
e[81]  = s21;  line;    connect;     P55,P56;                 black;0;gray;0
e[82]  = s22;  line;    connect;     P61,P62;                 black;0;gray;0
e[83]  = s23;  line;    connect;     P63,P64;                 black;0;gray;0
e[84]  = s1';  line;    connect;     A1,A2;                   black;0;blue;0
e[85]  = s2';  line;    connect;     B1,B2;                   black;0;blue;0
e[86]  = s3';  line;    connect;     C1,C2;                   black;0;blue;0
e[87]  = s4';  line;    connect;     P1,P2;                   black;0;blue;0
e[88]  = s5';  line;    connect;     P3,P4;                   black;0;blue;0
e[89]  = s6';  line;    connect;     P5,P6;                   black;0;blue;0
e[90]  = s7';  line;    connect;     P7,P8;                   black;0;blue;0
e[91]  = s8';  line;    connect;     P9,P10;                  black;0;blue;0
e[92]  = s9';  line;    connect;     P11,P12;                 black;0;blue;0
e[93]  = s10'; line;    connect;     P13,P14;                 black;0;blue;0
e[94]  = s11'; line;    connect;     P15,P16;                 black;0;blue;0
e[95]  = s12'; polygon; hexagon;     P17,P18,P19,P20,P19,P18; black;0;blue;0
e[96]  = s13'; polygon; hexagon;     P31,P32,P33,P34,P33,P32; black;0;blue;0
e[97]  = s14'; polygon; hexagon;     P35,P36,P37,P38,P37,P36; black;0;blue;0
e[98]  = s15'; polygon; hexagon;     P39,P40,P41,P42,P41,P40; black;0;blue;0
e[99]  = s16'; polygon; hexagon;     P43,P44,P45,P46,P45,P44; black;0;blue;0
e[100] = s17'; polygon; hexagon;     P47,P48,P49,P50,P49,P48; black;0;blue;0
e[101] = s18'; polygon; hexagon;     P51,P52,P53,P54,P53,P52; black;0;blue;0
e[102] = s19'; polygon; hexagon;     P51,P52,P57,P58,P57,P52; black;0;blue;0
e[103] = s20'; polygon; hexagon;     P51,P52,P59,P60,P59,P52; black;0;blue;0
e[104] = s21'; line;    connect;     P55,P56;                 black;0;blue;0
e[105] = s22'; line;    connect;     P61,P62;                 black;0;blue;0
e[106] = s23'; line;    connect;     P63,P64;                 black;0;blue;0
e[107] = P1';  point;   translation; P1,0.0,0.0;              black;blue;blue;smallcircle
e[108] = P3';  point;   translation; P3,0.0,0.0;              black;blue;blue;smallcircle
e[109] = P5';  point;   translation; P5,0.0,0.0;              black;blue;blue;smallcircle
e[110] = P7';  point;   translation; P7,0.0,0.0;              black;blue;blue;smallcircle
e[111] = P9';  point;   translation; P9,0.0,0.0;              black;blue;blue;smallcircle
e[112] = P11'; point;   translation; P11,0.0,0.0;             black;blue;blue;smallcircle
e[113] = P56'; point;   translation; P56,0.0,0.0;             black;blue;blue;smallcircle
e[114] = P13'; point;   translation; P13,0.0,0.0;             black;blue;blue;smallcircle
e[115] = P15'; point;   translation; P15,0.0,0.0;             black;blue;blue;smallcircle
e[116] = a;    measure; controller;  0.0,1.0,0.0,1.0,10,"a = ","";
e[117] = b;    measure; controller;  0.0,1.0,0.0,1.0,10,"b = ","";
e[118] = c;    measure; controller;  0.0,1.0,0.0,1.0,10,"c = ","";
e[119] = d;    measure; calculate;
               "(calculate(a)&(not(calculate(b))))|(calculate(b)&(not(calculate(a))))";
e[120] = P52'; point;   translation; P52,0.0,0.0;             black;blue;blue;smallcircle
e[121] = P57'; point;   translation; P57,0.0,0.0;             black;blue;blue;smallcircle
e[122] = P59'; point;   translation; P59,0.0,0.0;             black;blue;blue;smallcircle
e[123] = e;    measure; calculate;
               "((calculate(a)&(calculate(b)))|(calculate(c)&(calculate(d))))",-8.0,-10.0,
```

```
               "Übertrag = ","";
e[124] = f;    measure; calculate;
               "((not(calculate(d))&(calculate(c)))|(not(calculate(c))&(calculate(d))))",
               4.0,-10.0,"Summe = ","";

// Ein- und Ausblenden von Objekten
// ================================

hidden[1] = "if (not(calculate(a))) hide (s1',P1',P5',P9',s4',s6',s8')"
hidden[2] = "if (not(calculate(b))) hide (s2',P3',P7',P11',s5',s7',s9')"
hidden[3] = "if (not(calculate(c))) hide (s3',P56',P13',P15',s10',s11',s21')"
hidden[4] = "if (not(calculate(a)&calculate(b))) hide (s14')"
hidden[5] = "if (not((calculate(a)&(not(calculate(b)))))) hide (s12')"
hidden[6] = "if (not((calculate(b)&(not(calculate(a)))))) hide (s13')"
hidden[7] = "if (not(calculate(d))) hide (s18',s19',s20',P52',P57',P59')"
hidden[8] = "if (not(calculate(a)&(calculate(b)))) hide (s14')"
hidden[9] = "if (not(calculate(c)&(calculate(d)))) hide (s15')"
hidden[10] = "if (not((calculate(a)&(calculate(b)))|(calculate(c)&(calculate(d)))))
             hide (s22')"
hidden[11] = "if (not(not(calculate(d))&(calculate(c)))) hide (s16')"
hidden[12] = "if (not(not(calculate(c))&(calculate(d)))) hide (s17')"
hidden[13] = "if (not((not(calculate(d))&(calculate(c)))|
             (not(calculate(c))&(calculate(d))))) hide (s23')"

// Bild-Dateien
// ============

image[1] = "Kombigatter.gif", 380, 300
image[2] = "Kombigatter.gif", 160, 300
image[3] = "Kombigatter.gif", 270, 140
```

Flächeninhalt umfangsgleicher Vierecke (Seite 108)

```
//
// Datei: Umfangsgleiche_Vierecke.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
//

// Figurenbeschreibung
// ===================

e[1]  = P1;      point;   fixed;          -14.0,10.0;            "hideLabel"
e[2]  = P2;      point;   horizontal;     P1,-10.0;              "hideLabel"
e[3]  = P3;      point;   horizontal;     P1,-4.0;               "hideLabel"
e[4]  = P4;      point;   horizontal;     P1,1.0;                "hideLabel"
e[5]  = P5;      point;   horizontal;     P1,5.0;                "hideLabel"
e[6]  = Länge_a; line;    connect;        P1,P2;                 "hideLabel"
e[7]  = Länge_b; line;    connect;        P3,P2;                 "hideLabel"
e[8]  = Länge_c; line;    connect;        P3,P4;                 "hideLabel"
e[9]  = Länge_d; line;    connect;        P4,P5;                 "hideLabel"
e[10] = A;       point;   free;           2.0,0.0;
e[11] = kA1;     circle;  radius;         A,P4,P5;               "hidden"
e[12] = B;       point;   circleSlider;   2.0,9.0,kA1;
e[13] = kA2;     circle;  radius;         A,P4,P3;               "hidden"
e[14] = D;       point;   circleSlider;   -2.0,0.0,kA2;
e[15] = kD;      circle;  radius;         D,P2,P3;               "hidden"
e[16] = kB;      circle;  radius;         B,P2,P1;               "hidden"
e[17] = C;       point;   intersection;   kD,kB,2;
e[18] = a;       line;    connect;        B,C;
e[19] = b;       line;    connect;        C,D;
e[20] = c;       line;    connect;        D,A;
e[21] = d;       line;    connect;        A,B;
e[22] = ma;      measure; distance;       P1,P2,-13.0,10.5,"a = ","";
e[23] = mb;      measure; distance;       P3,P2,-8.0,10.5,"b = ","";
e[24] = mc;      measure; distance;       P3,P4,-3.0,10.5,"c = ","";
e[25] = md;      measure; distance;       P4,P5, 2.0,10.5,"d = ","";
e[26] = mg;      measure; calculate;
                 "calculate(ma)+calculate(mb)+calculate(mc)+calculate(md)",-13.0,8.0,
                 "Gesamtumfang = ","";
e[27] = p;       polygon; quadrilateral;  A,B,C,D;               "hidden"
e[28] = mA;      measure; area;           p,-13.0,-1.0,"Flächeninhalt = ","";
e[29] = P6;      point;   fixed;          -14.0,-2.0;            "hidden"
e[30] = P7;      point;   fixed;          -14.0,-3.0;            "hidden"
e[31] = P8;      point;   functionDepend; "coordinateX(P6)+calculate(mA)",
                 "coordinateY(P6)";                              "hidden"
e[32] = P9;      point;   functionDepend; "coordinateX(P7)+calculate(mA)",
                 "coordinateY(P7)";                              "hidden"
e[33] = p2;      polygon; quadrilateral;  P6,P7,P9,P8;           0;black;black;green
e[34] = po1;     polygon; triangle;       A,B,C;                 "hidden"
e[35] = po2;     polygon; triangle;       B,C,D;                 "hidden"
e[36] = po3;     polygon; triangle;       C,D,A;                 "hidden"
e[37] = po4;     polygon; triangle;       D,A,B;                 "hidden"
e[38] = m0;      measure; inclusion;      D,po1,0.0,-50.0,"","";
e[39] = m1;      measure; inclusion;      A,po2,0.0,-50.0,"","";
e[40] = m2;      measure; inclusion;      B,po3,0.0,-50.0,"","";
e[41] = m3;      measure; inclusion;      C,po4,0.0,-50.0,"","";
e[42] = cond;    measure; calculate;
                 "if (calculate(m0)+calculate(m1)+calculate(m2)+calculate(m3) != 0.0)
                 then (0.0) else (1.0)";
e[43] = k;       circle;  circumcircle;   A,B,D;                 "hideLabel"

// Ein- und Ausblenden von Objekten
// ================================

hidden[1] = "if (not(isIncident(C,k))) hide (k)"

// Beschränken des Zustandsraums der Figur
// =======================================

limit[1] = "$(C,defined)"
limit[2] = "calculate(cond)"
```

Die Picksche Formel (Seite 110)

```
//
// Datei: PickscheFormel.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
//

// Systemvariablen
// ===============

APPLET_WIDTH  = 640
APPLET_HEIGHT = 480
snapToGrid    = true
showGrid      = true
GRIDSIZE      = 20
gridColor     = lightGray

// Figurenbeschreibung
// ===================

e[1]  = A;  point;   free;      -7.0,-2.0; "hideLabel"
e[2]  = B;  point;   free;      -9.0,-3.0; "hideLabel"
e[3]  = C;  point;   free;      -8.0,-4.0; "hideLabel"
e[4]  = P1; polygon; polygon;   A,B,C;     "hideLabel"
e[5]  = D;  point;   free;      -4.0,-4.0; "hideLabel"
e[6]  = E;  point;   free;      -4.0,-2.0; "hideLabel"
e[7]  = F;  point;   free;      -2.0,-2.0; "hideLabel"
e[8]  = G;  point;   free;      -2.0,-4.0; "hideLabel"
e[9]  = P2; polygon; polygon;   D,E,F,G;   "hideLabel"
e[10] = H;  point;   free;      1.0,-2.0;  "hideLabel"
e[11] = I;  point;   free;      3.0,-2.0;  "hideLabel"
e[12] = J;  point;   free;      3.0,-4.0;  "hideLabel"
e[13] = K;  point;   free;      1.0,-4.0;  "hideLabel"
e[14] = L;  point;   free;      0.0,-3.0;  "hideLabel"
e[15] = P3; polygon; polygon;   H,I,J,K,L; "hideLabel"
e[16] = i1; measure; function;  "Functional_PickscheFormel","i","P1",-9.0,-6.0,"i = ","";
e[17] = r1; measure; function;  "Functional_PickscheFormel","r","P1",-9.0,-7.0,"r = ","";
e[18] = i2; measure; function;  "Functional_PickscheFormel","i","P2",-4.0,-6.0,"i = ","";
e[19] = r2; measure; function;  "Functional_PickscheFormel","r","P2",-4.0,-7.0,"r = ","";
e[20] = i3; measure; function;  "Functional_PickscheFormel","i","P3",1.0,-6.0,"i = ","";
e[21] = r3; measure; function;  "Functional_PickscheFormel","r","P3",1.0,-7.0,"r = ","";
e[22] = sw; measure; checkbox;  "Lösung zeigen",0;
e[23] = s1; measure; calculate; "calculate(i1)+(calculate(r1)/2)-1",-10.0,-8.0,
            "i + (r/2) - 1 = ","";
e[24] = s2; measure; calculate; "calculate(i2)+(calculate(r2)/2)-1",-4.0,-8.0,
            "i + (r/2) - 1 = ","";
e[25] = s3; measure; calculate; "calculate(i3)+(calculate(r3)/2)-1",1.0,-8.0,
            "i + (r/2) - 1 = ","";
e[26] = s4; measure; calculate; "calculate(i1)+(calculate(r1)/2)-1",-9.0,-8.0,
            "f = ","";
e[27] = s5; measure; calculate; "calculate(i2)+(calculate(r2)/2)-1",-4.0,-8.0,
            "f = ","";
e[28] = s6; measure; calculate; "calculate(i3)+(calculate(r3)/2)-1",1.0,-8.0,
            "f = ","";

// Ein- und Ausblenden von Objekten
// ================================

hidden[1] = "if (not(calculate(sw))) hide (s1,s2,s3,Textbox_2)"
hidden[2] = "if (calculate(sw)) hide (s4,s5,s6)"

// Textfenster
// ===========

<Textbox>
 Position = 20;20;320;140
  G. Pick (1859-1942) hat eine Formel aufgestellt,
  mit der man den Flächeninhalt von Gittervielecken
  bestimmen kann.
  Dabei spielen die Anzahlen von Gitterpunkten auf
  dem Rand (r) und im Inneren eines Vielecks (i)
```

```
  eine Rolle. Kannst Du eine Formel für den
  Flächeninhalt f finden?
</Textbox>

<Textbox>
 Position = 20;20;320;140
  Lösung:
  Sei bei einem Vieleck
  i = Anzahl der Gitterpunkte im Inneren und
  r = Anzahl der Gitterpunkte auf dem Rand, dann
  ist der Flächeninhalt f = i + (r / 2) - 1 .
</Textbox>
```

Höhenfußpunktdreieck (Seite 113)

```
//
// Datei: Hoehenfusspunktdreieck.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
// Figurenidee aus Holland:
// Geometrie in der Sekundarstufe, 1996, S. 108ff
//

// Systemvariablen
// ===============

 showGrid = false
 showAxis = false
 dragMeasure = true

// Figurenbeschreibung
// ===================

e[1]  = A;   point;   free;         6.5,-5.7;
e[2]  = B;   point;   free;         -0.4,4.65;
e[3]  = C;   point;   free;         -8.25,-7.25;
e[4]  = p0;  polygon; triangle;     A,B,C;               0;0;black;0
e[5]  = P;   point;   foot;         A,B,C;
e[6]  = Q;   point;   foot;         B,C,A;
e[7]  = R;   point;   foot;         C,A,B;
e[8]  = h1;  line;    connect;      B,Q;                 "hideLabel"
e[9]  = h2;  line;    connect;      P,A;                 "hideLabel"
e[10] = h3;  line;    connect;      R,C;                 "hideLabel"
e[11] = H;   point;   intersection; h1,h2;
e[12] = k1;  circle;  circumcircle; P,Q,H;               0;0;lightGray;0
e[13] = k2;  circle;  circumcircle; R,Q,H;               0;0;lightGray;0
e[14] = p1;  polygon; triangle;     P,Q,R;               0;0;black;0
e[15] = sw1; measure; checkbox;     "1. Hilfe zeigen",0;
e[16] = k4;  circle;  circumcircle; A,R,P;               0;0;gray;0
e[17] = sw2; measure; checkbox;     "2. Hilfe zeigen",0;

// Ein- und Ausblenden von Objekten
// ================================

hidden[1] = "if (not(calculate(sw1))) hide (k1,k2)"
hidden[2] = "if (not(calculate(sw2))) hide (k4)"

// Textfenster
// ===========

<ProblemText>
 Position = 20;20;250;70
  In jedem spitzwinkligen Dreieck ABC
  sind die Höhen die Winkelhalbierenden
  des Höhenfußpunktdreiecks PQR.
</ProblemText>
```

Lotsumme im gleichseitigen Dreieck (Seite 115)

```
//
// Datei: Lotsumme_im_gleichseitigen_Dreieck.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
// Aufgabe aus Holland: Geometrie in der Sekundarstufe. 1996, S. 120
//

// Figurenbeschreibung
// ===================

e[1]  = A;   point;   free;         7.0,-3.0;
e[2]  = C;   point;   free;         -7.0,-3.0;
e[3]  = B;   point;   rotation;     C,A,1.047197551,1.0;
e[4]  = c;   line;    connect;      A,B;                0;0;black;0
e[5]  = a;   line;    connect;      B,C;                0;0;black;0
e[6]  = b;   line;    connect;      C,A;                0;0;black;0
e[7]  = p;   polygon; triangle;     A,B,C;              "hidden"
e[8]  = P;   point;   areaSlider;   0.0,2.0,p;
e[9]  = Ha;  point;   foot;         P,a;                "hideLabel"
e[10] = Hb;  point;   foot;         P,b;                "hideLabel"
e[11] = Hc;  point;   foot;         P,c;                "hideLabel"
e[12] = x;   line;    connect;      P,Ha;               black;0;green;0
e[13] = y;   line;    connect;      P,Hb;               black;0;blue;0
e[14] = z;   line;    connect;      P,Hc;               black;0;red;0
e[15] = a';  line;    parallel;     P,a;                black;0;lightGray;0
e[16] = b';  line;    parallel;     P,b;                black;0;lightGray;0
e[17] = c';  line;    parallel;     P,c;                black;0;lightGray;0
e[18] = S1;  point;   intersection; a',c;
e[19] = S2;  point;   intersection; a',b;
e[20] = S3;  point;   intersection; b',a;
e[21] = S4;  point;   intersection; b',c;
e[22] = S5;  point;   intersection; c',a;
e[23] = S6;  point;   intersection; c',b;
e[24] = b2'; line;    parallel;     S5,b;               black;0;lightGray;0
e[25] = S7;  point;   intersection; c,b2';
e[26] = a''; line;    connect;      S1,S2;              0;0;lightGray;0
e[27] = b''; line;    connect;      S3,S4;              0;0;lightGray;0
e[28] = c''; line;    connect;      S5,S6;              0;0;lightGray;0
e[29] = b2'';line;    connect;      S5,S7;              0;0;lightGray;0
e[30] = HB2; point;   foot;         B,b2'';
e[31] = HA2; point;   foot;         S5,b'';
e[32] = hA'; line;    connect;      S5,HA2;             0;0;green;0
e[33] = hB'; line;    connect;      B,HB2;              0;0;red;0
e[34] = P';  point;   translation;  P,0.0,0.0;          0;red;black;smallcircle
e[35] = sw1; measure; checkbox;     "1. Hilfe zeigen",0;
e[36] = sw2; measure; checkbox;     "2. Hilfe zeigen",0;

// Ein- und Ausblenden von Objekten
// ================================

hidden[1] = "if (1.0) hide (HA2,HB2,a',b',c',b2',hA',hB')"
hidden[2] = "if (not(calculate(sw1))) hide (S1,S2,S3,S4,S5,S6,S7,a'',b'',c'',b2'')"
hidden[3] = "if (not(calculate(sw2))) hide (a'',b'',c'',b2'',hA',hB')"
hidden[4] = "if (not(calculate(sw2))) hide (hA',hB')"

// Textfenster
// ===========

<ProblemText>
 Position = 100;360;280;80
   Begründe, warum die Summe der Abstände
   von P zu den Seiten des Dreiecks ABC
   stets gleich der Höhe des Dreiecks ist.
</ProblemText>
```

Ableitung (Seite 118)

```
//
// Datei: Ableitung.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
//

// Systemvariablen
// ===============

APPLET_WIDTH      = 640
APPLET_HEIGHT     = 480
WORLD_X_MAX       = +8.0
WORLD_X_MIN       = -8.0
WORLD_Y_MAX       = +6.0
WORLD_Y_MIN       = -6.0
MEASURE_EXACTNESS = 1

// Figurenbeschreibung
// ===================

e[1]  = O;     point;   fixed;          0.0,0.0;           "hidden"
e[2]  = coord; point;   coordSystem;    O,300,300,200,200; 0;red;black;0
e[3]  = curve; line;    curve;          "t","t^5/300-11*t^3/60+1.5*t",-8.0,8.0,300;
e[4]  = A;     point;   curveslider;    0.5,0.5,curve;
e[5]  = m0;    measure; coordinates;    A,0.5,-2.5,"A = ","";
e[6]  = m1;    measure; calculate;
               "coordinateX(A)^5/300-11*coordinateX(A)^3/60+1.5*coordinateX(A)",0.5,-3.5,
               "f(x) = ","";
e[7]  = m2;    measure; calculate;      "coordinateX(A)^4/60-33*coordinateX(A)^2/60+1.5",
               0.5,-4.5,"f'(x) = ","";
e[8]  = m3;    measure; calculate;      "coordinateX(A)^3/15-33*coordinateX(A)/30",
               0.5,-5.5,"f''(x) = ","";
e[9]  = A';    point;   functionDepend; "coordinateX(A)+1.0","coordinateY(A)+calculate(m2)";
e[10] = t;     line;    straightLine;   A,A';               0;0;black;0
e[11] = A'';   point;   functionDepend; "coordinateX(A)+1.0","coordinateY(A)"; "hidden"
e[12] = f'(x); line;    connect;        A',A'';             black;0;gray;0
e[13] = l;     line;    connect;        A'',A;              black;0;gray;0

// Ein- und Ausblenden von Objekten
// ================================

hidden[1] = "if (abs(calculate(m3)) > 0.01) hide (Textbox_1)"
hidden[2] = "if (abs(calculate(m2)) > 0.015) hide (Textbox_2)"

// Textfenster
// ===========

<TextBox>
 TextBox = 340;20;100;40
   Wendepunkt
</TextBox>

<TextBox>
 TextBox = 340;20;100;40
   Extrempunkt
</TextBox>

// Bild-Datei
// ==========

image[1] = "Ableitung.gif", 60, 370
```

Evolute einer Parabel (Seite 121)

```
//
// Datei: Evolute01.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
//

// Systemvariablen
// ===============

APPLET_WIDTH  = 640
APPLET_HEIGHT = 480
WORLD_X_MAX   = +4.0
WORLD_X_MIN   = -4.0
WORLD_Y_MAX   = +3.0
WORLD_Y_MIN   = -3.0

// Figurenbeschreibung
// ===================

e[1]  = O;       point;   fixed;          0.0,0.0;          "hidden"
e[2]  = coord;   point;   coordSystem;    0,300,300,200,200; 0;red;black;0
e[3]  = a;       measure; controller;     1.0,0.5,-3.0,3.0,100,"a = ","";
e[4]  = b;       measure; controller;     0.0,0.5,-3.0,3.0,100,"b = ","";
e[5]  = c;       measure; controller;     0.0,0.5,-3.0,3.0,100,"c = ","";
e[6]  = parabel; line;    curve;          "t",
                 "calculate(a)*t^2+calculate(b)*t+calculate(c)",-4.0, 4.0, 50;
e[7]  = T;       point;   curveSlider;    0.45,0.23,parabel;
e[8]  = t;       measure; property;       T,"t";
e[9]  = d;       measure; calculate;
                 "(1+(2*calculate(a)*calculate(t)+calculate(b))^2)/(2*calculate(a))";
e[10] = M;       point;   functionDepend;
                 "calculate(t)+calculate(d)*(-2*calculate(a)*calculate(t)-calculate(b))",
                 "calculate(a)*calculate(t)^2+calculate(b)*calculate(t)+calculate(c)+
                 calculate(d)";                              black;green;black;smallcircle
e[11] = k2;      circle;  radius;         M,M,T;            0;black;black;0
e[12] = n;       line;    straightline;   M,T;              0;0;lightGray;0
e[13] = ta;      line;    perpendicular;  T,n;              0;0;lightGray;0
e[14] = locus;   line;    locus;          M,T;              0;0;black;0

// Textfenster
// ===========

<TextBox>
 TextBox = 20;340;220;120
   Bewege den Punkt T entlang der
   Normalparabel und betrachte die
   Bahn von M.

   Welcher Zusammenhang besteht
   zwischen der Normalen im
   Punkt T und der Evolute?
</TextBox>

<TextBox>
 TextBox = 400;360;220;100
   Der Punkt M ist der Mittelpunkt
   des Krümmungskreises.
   Die Kurve, die durch die Bahn
   des Mittelpunkts erzeugt wird,
   nennt man EVOLUTE.
</TextBox>
```

Evolute einer Ellipse (Seite 122)

```
//
// Datei: Evolute02.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
//

// Systemvariablen
// ===============

APPLET_WIDTH  = 640
APPLET_HEIGHT = 480
WORLD_X_MAX   = +8.0
WORLD_X_MIN   = -8.0
WORLD_Y_MAX   = +6.0
WORLD_Y_MIN   = -6.0

// Figurenbeschreibung
// ===================

e[1]  = O;     point;   fixed;          0.0,0.0;           "hidden"
e[2]  = coord; point;   coordSystem;    0,300,300,200,200; 0;red;black;0
e[3]  = a;     measure; controller;     3.0,0.25,-3.0,3.0,100,"a = ","";
e[4]  = b;     measure; controller;     2.0,0.25,-3.0,3.0,100,"b = ","";
e[5]  = k1;    line;    curve;          "calculate(a)*cos(t)",
               "calculate(b)*sin(t)", 0.0, 6.30, 30;
e[6]  = T;     point;   curveslider;    -3.0,0.0,k1;
e[7]  = t;     measure; property;       T,"t";
e[8]  = M;     point;   functionDepend;
               "((calculate(a)^2-calculate(b)^2)/calculate(a))*cos(calculate(t))^3",
               "(-(calculate(a)^2-calculate(b)^2)/calculate(b))*sin(calculate(t))^3";
               black;green;black;smallcircle
e[9]  = k2;    circle;  radius;         M,M,T;             0;0;black;0
e[10] = n;     line;    straightline;   M,T;               0;0;lightGray;0
e[11] = ta;    line;    perpendicular;  T,n;               0;0;lightGray;0
e[12] = locus; line;    locus;          M,T;

// Textfenster
// ===========

<TextBox>
 TextBox = 10;380;-1;-1
   Die Ellipse hat die Parametergleichung
   x(t) = a*cos(t) und y(t) = b*sin(t)
   mit t = 0 .. 2*Pi.

   Betrachte die Bahn ihrer Evolute.
</TextBox>
```

Evolute einer Zykloide (Seite 123)

```
//
// Datei: Evolute03.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
//

// Systemvariablen
// ===============

APPLET_WIDTH  = 640
APPLET_HEIGHT = 480

// Figurenbeschreibung
// ===================

e[1]  = O;     point;   fixed;          0.0,0.0;          "hidden"
e[2]  = coord; point;   coordSystem;    0,300,300,200,200; 0;red;black;0
e[3]  = a;     measure; controller;     2.0,0.25,-3.0,3.0,100,"a = ","";
e[4]  = r;     measure; controller;     2.0,0.25,-3.0,3.0,100,"r = ","";
e[5]  = z1;    line;    curve;          "calculate(r)*t-calculate(a)*sin(t)",
               "calculate(r)-calculate(a)*cos(t)", -6.30, 6.30, 50;
e[6]  = T;     point;   curveslider;    2.0,0.0,z1;
e[7]  = t;     measure; property;       T,"t";
e[8]  = M;     point;   functionDepend;
               "(calculate(r)*calculate(t)+calculate(a)*sin(calculate(t)))",
               "(calculate(a)*cos(calculate(t))-calculate(r))";
e[9]  = k2;    circle;  radius;         M,M,T;             0;black;black;0
e[10] = n';    line;    straightline;   M,T;               0;0;lightGray;0
e[11] = ta;    line;    perpendicular;  T,n';              0;0;lightGray;0
e[12] = locus; line;    locus;          M,T;

// Textfenster
// ===========

<TextBox>
 TextBox = 20;360;-1;-1
   Die Kurve stellt eine Zykloide dar.
   Ihre Parametergleichung lautet:

   x(t) = r*t - a*sin( t )
   y(t) = r - a*cos( t )

   mit t = -2*Pi ... 2*Pi.
</TextBox>
```

Satz von Varignon (Seite 124)

```
//
// Datei: Satz_von_Varignon.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
//

// Figurenbeschreibung
// ===================

e[1]  = P;   point;   free;          4.0,-1.0;
e[2]  = Q;   point;   free;          -3.0,0.0;
e[3]  = R;   point;   free;          -5.0,-4.0;
e[4]  = S;   point;   parallelogram; P,Q,R;
e[5]  = sq;  polygon; quadrilateral; P,Q,R,S;          0;0;blue;0
e[6]  = A;   point;   free;          6.0,-3.0;
e[7]  = B;   point;   mirror;        A,P;
e[8]  = C;   point;   mirror;        B,Q;
e[9]  = D;   point;   mirror;        C,R;
e[10] = a;   line;    connect;       A,B;              0;0;gray;0
e[11] = b;   line;    connect;       B,C;              0;0;gray;0
e[12] = c;   line;    connect;       C,D;              0;0;gray;0
e[13] = d;   line;    connect;       D,A;              0;0;gray;0
e[14] = sw1; measure; checkbox;      "Lösung zeigen",0;

// Ein- und Ausblenden von Objekten
// ================================

hidden[1] = "if (not(calculate(sw1))) hide (A,B,C,D,a,b,c,d,Textbox_2)"

// Textfenster
// ===========

<Textbox>
 Position = 10;10;290;100
  Wieviele Ausgangsvierecke gibt es zu einem
  gegebenen Varignon-Parallelogramm?
  Überlegen Sie sich eine Antwort, bevor Sie
  die Lösung einblenden.
</Textbox>

<Textbox>
 Position = 10;10;290;100
  Zu einem gegebenen Varignon-Parallelogramm
  gibt es unendlich viele verschiedene Ausgangs-
  vierecke.
  Verschieben Sie den Punkt A, um das Ausgangs-
  viereck zu verändern.
</Textbox>
```

Teilverhältnis (Seite 126)

```
//
// Datei: Teilverhaeltnis.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
//

// Systemvariablen
// ===============

showGrid          = true
gridColor         = lightGray
MEASURE_EXACTNESS = 2

// Figurenbeschreibung
// ===================

e[1]  = A;  point;   fixed;        -7.0,3.0;
e[2]  = B;  point;   fixed;        5.0,3.0;
e[3]  = g;  line;    straightLine; A,B;        "hideLabel"
e[4]  = P1; point;   fixed;        -3.0,3.0;   0;lightGray;black;smallcircle
e[5]  = P2; point;   fixed;        1.0,3.0;    0;lightGray;black;smallcircle
e[6]  = T'; point;   fixed;        9.0,3.0;    black;blue;black;smallcircle
e[7]  = P4; point;   fixed;        -11.0,3.0;  0;lightGray;black;smallcircle
e[8]  = P5; point;   fixed;        -15.0,3.0;  0;lightGray;black;smallcircle
e[9]  = P6; point;   fixed;        13.0,3.0;   0;lightGray;black;smallcircle
e[10] = T;  point;   lineSlider;   0.0,0.0,g;
e[11] = t;  measure; function;     "Functional_Teilverhaeltnis","A","B","T";
e[12] = m0; measure; button;       "Hilfe","help";
e[13] = m1; measure; button;       "Auswertung","evaluate";
e[14] = m2; measure; checkbox;     "Lösung zeigen",0;

// Beschränken des Zustandsraums der Figur
// =======================================

limit[1] = "not(isIncident(A,T))&not(isIncident(B,T))"

// Ein- und Ausblenden von Objekten
// ================================

hidden[1] = "if (not(calculate(m2))) hide (P1,P2,P4,P5,P6,T',Textbox_2)"

// Textfenster
// ===========

<Textbox>
 Position = 10;10;-1;-1
  Bewegen Sie den Punkt T an die Position,
  so daß AT : TB = -4 ist.
</Textbox>

<Textbox>
 Position = 10;210;320;40
  Lösung:
  Für den Punkt T' gilt:  AT' / T'B = -4 .
</Textbox>

// Hilfen
// ======

<Help>
   Nutzen Sie das Hintergrundraster,
   um die Streckenlängen zu bestimmen.
</Help>

// Antwortanalyse
// ==============

<Problem>
 MAX_ANSWER = 4
 condition[1] = "calculate(t) > -4.1"
```

```
 condition[2] = "calculate(t) < -3.9"
 condition[3] = "calculate(t) > -5.1"
 condition[4] = "calculate(t) < -4.9"
 condition[5] = "calculate(t) > -1"
 condition[6] = "calculate(t) < 0.0"
 condition[7] = "calculate(t) > 0.0"
 condition[8] = "calculate(t) < -1.0"
</Problem>

<Answer 1>
 key = "condition[1] AND condition[2]"
 comment[1] = "Richtig. /nDer Punkt T muß außerhalb der Strecke AB liegen, /n
               weil das Teilverhältnis negativ ist. /n
               Der Wert des Teilverhälnisses soll -4 betragen, /nd. h.,
               die Strecke AT muß viermal so lang sein wie die Strecke TB."
</Answer 1>

<Answer 2>
 key = "condition[3] AND condition[4]"
 comment[1] = "Ihre Antwort ist teilweise richtig. /n
               Die Strecke TB muß kürzer sein als die Strecke AT. /n
               In der aktuellen Lage von T beträgt das Teilverhältnis jedoch AT : TB = -5 ./n
               Versuchen Sie es noch einmal."
 comment[2] = "Ihre Antwort ist immer noch nicht richtig. /n /n
               Lösung:/nDer Punkt T muß außerhalb der Strecke AB liegen, /n
               weil das Teilverhältnis negativ ist. /n
               Der Wert des Teilverhälnisses soll -4 betragen, /n
               d. h., die Strecke AT muß viermal so lang sein wie die Strecke TB."
</Answer 2>

<Answer 3>
 key = "condition[5] AND condition[6]"
 comment[1] = "Ihre Antwort ist nicht richtig. /n Wenn die Strecke AT kürzer ist
               als die Strecke TB, /ndann gilt für das Teilverhältnis -1 < t < 0 . /n
               Versuchen Sie es noch einmal."
 comment[2] = "Ihre Antwort ist immer noch nicht richtig. /n /n
               Lösung:/nDer Punkt T muß außerhalb der Strecke AB liegen, /n
               weil das Teilverhältnis negativ ist. /n
               Der Wert des Teilverhälnisses soll -4 betragen, /n
               d. h., die Strecke AT muß viermal so lang sein wie die Strecke TB."
</Answer 2>

<Answer 4>
 key = "condition[7]"
 comment[1] = "Ihre Antwort ist nicht richtig. /n
               Wenn der Punkt T zwischen A und B liegt, /n
               ist das Teilverhältnis größer Null./n
               Versuchen Sie es noch einmal."
 comment[2] = "Ihre Antwort ist immer noch nicht richtig. /n /n
               Lösung:/nDer Punkt T muß außerhalb der Strecke AB liegen, /n
               weil das Teilverhältnis negativ ist. /n
               Der Wert des Teilverhälnisses soll -4 betragen, /n
               d. h., die Strecke AT muß viermal so lang sein wie die Strecke TB."
</Answer 4>

<Answer 5>
 key = "condition[8]"
 comment[1] = "Ihre Antwort ist teilweise richtig. /n
               Die Strecke TB muß kürzer sein als die Strecke AT. /n
               In der aktuellen Lage von T ist das Teilverhältnis jedoch ungleich -4 ./n
               Versuchen Sie es noch einmal."
 comment[2] = "Ihre Antwort ist teilweise richtig. /n
               Die Strecke TB muß kürzer sein als die Strecke AT. /n
               Die Strecke AT muß viermal so lang sein wie die Strecke TB. /n
               Versuchen Sie es noch einmal."
 comment[3] = "Ihre Antwort ist immer noch nicht richtig. /n /n
               Lösung:/nDer Punkt T muß außerhalb der Strecke AB liegen, /n
               weil das Teilverhältnis negativ ist. /n
               Der Wert des Teilverhälnisses soll -4 betragen, /n
               d. h., die Strecke AT muß viermal so lang sein wie die Strecke TB."
</Answer 5>
```

Satz von Ceva (Seite 129)

```
//
// Datei: Satz_von_Ceva.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
//

// Systemvariablen
// ===============

MEASURE_EXACTNESS = 1

// Figurenbeschreibung
// ===================

e[1]  = A;    point;   free;           -7.0, -7.0;    black;red;black;circle
e[2]  = B;    point;   free;           9.0, -6.0;     black;red;black;circle
e[3]  = C;    point;   free;           0.0, 3.0;      black;red;black;circle
e[4]  = c;    line;    connect;        A,B;           0;0;blue;0
e[5]  = a;    line;    connect;        B,C;           0;0;blue;0
e[6]  = b;    line;    connect;        A,C;           0;0;blue;0
e[7]  = TA1;  point;   functionDepend; "$(A,x)+XVector(A,C)/12",
              "$(A,y)+YVector(A,C)/12"; 0;           lightGray;black;smallcircle
e[8]  = TA2;  point;   translation;    TA1,A,TA1;     0;lightGray;black;smallcircle
e[9]  = TA3;  point;   translation;    TA2,TA1,TA2;   0;lightGray;black;smallcircle
e[10] = TA4;  point;   translation;    TA3,TA2,TA3;   0;lightGray;black;smallcircle
e[11] = TA5;  point;   translation;    TA4,TA3,TA4;   0;lightGray;black;smallcircle
e[12] = TA6;  point;   translation;    TA5,TA4,TA5;   0;lightGray;black;smallcircle
e[13] = TA7;  point;   translation;    TA6,TA5,TA6;   0;lightGray;black;smallcircle
e[14] = TA8;  point;   translation;    TA7,TA6,TA7;   0;lightGray;black;smallcircle
e[15] = TA9;  point;   translation;    TA8,TA7,TA8;   0;lightGray;black;smallcircle
e[16] = TA10; point;   translation;    TA9,TA8,TA9;   0;lightGray;black;smallcircle
e[17] = TA11; point;   translation;    TA10,TA9,TA10; 0;lightGray;black;smallcircle
e[18] = TB1;  point;   functionDepend; "$(B,x)+XVector(B,A)/12",
              "$(B,y)+YVector(B,A)/12";              0;lightGray;black;smallcircle
e[19] = TB2;  point;   translation;    TB1,B,TB1;     0;lightGray;black;smallcircle
e[20] = TB3;  point;   translation;    TB2,TB1,TB2;   0;lightGray;black;smallcircle
e[21] = TB4;  point;   translation;    TB3,TB2,TB3;   0;lightGray;black;smallcircle
e[22] = TB5;  point;   translation;    TB4,TB3,TB4;   0;lightGray;black;smallcircle
e[23] = TB6;  point;   translation;    TB5,TB4,TB5;   0;lightGray;black;smallcircle
e[24] = TB7;  point;   translation;    TB6,TB5,TB6;   0;lightGray;black;smallcircle
e[25] = TB8;  point;   translation;    TB7,TB6,TB7;   0;lightGray;black;smallcircle
e[26] = TB9;  point;   translation;    TB8,TB7,TB8;   0;lightGray;black;smallcircle
e[27] = TB10; point;   translation;    TB9,TB8,TB9;   0;lightGray;black;smallcircle
e[28] = TB11; point;   translation;    TB10,TB9,TB10; 0;lightGray;black;smallcircle
e[29] = TC1;  point;   functionDepend; "$(C,x)+XVector(C,B)/12",
              "$(C,y)+YVector(C,B)/12";              0;lightGray;black;smallcircle
e[30] = TC2;  point;   translation;    TC1,C,TC1;     0;lightGray;black;smallcircle
e[31] = TC3;  point;   translation;    TC2,TC1,TC2;   0;lightGray;black;smallcircle
e[32] = TC4;  point;   translation;    TC3,TC2,TC3;   0;lightGray;black;smallcircle
e[33] = TC5;  point;   translation;    TC4,TC3,TC4;   0;lightGray;black;smallcircle
e[34] = TC6;  point;   translation;    TC5,TC4,TC5;   0;lightGray;black;smallcircle
e[35] = TC7;  point;   translation;    TC6,TC5,TC6;   0;lightGray;black;smallcircle
e[36] = TC8;  point;   translation;    TC7,TC6,TC7;   0;lightGray;black;smallcircle
e[37] = TC9;  point;   translation;    TC8,TC7,TC8;   0;lightGray;black;smallcircle
e[38] = TC10; point;   translation;    TC9,TC8,TC9;   0;lightGray;black;smallcircle
e[39] = TC11; point;   translation;    TC10,TC9,TC10; 0;lightGray;black;smallcircle
e[40] = Lc;   line;    connect;        A,TC9;         0;0;lightGray;0
e[41] = La;   line;    connect;        B,TA6;         0;0;lightGray;0
e[42] = Lb;   line;    connect;        C,TB3;         0;0;lightGray;0
e[43] = A';   point;   dragable;       0.0,0.0,a;     black;red;black;circle
e[44] = B';   point;   dragable;       0.0,0.0,b;     black;red;black;circle
e[45] = C';   point;   dragable;       0.0,0.0,c;     black;red;black;circle
e[46] = e;    line;    connect;        A,A';          0;0;black;0
e[47] = f;    line;    connect;        B,B';          0;0;black;0
e[48] = g;    line;    connect;        C,C';          0;0;black;0
e[49] = t;    measure; calculate;
              "ratio(B,A',A',C)*ratio(C,B',B',A)*ratio(A,C',C',B)";
e[50] = tA';  measure; function;      "Functional_Teilverhaeltnis","B","C","A'";
e[51] = tB';  measure; function;      "Functional_Teilverhaeltnis","C","A","B'";
e[52] = tC';  measure; function;      "Functional_Teilverhaeltnis","A","B","C'";
```

```
e[53] = m0;   measure; button;        "Hilfe","help";
e[54] = m1;   measure; button;        "Auswertung","evaluate";
e[55] = m2;   measure; checkbox;      "Lösung zeigen",0;

// Ein- und Ausblenden von Objekten
// ================================

hidden[1] = "if (not(calculate(m2))) hide (La,Lb,Lc,Textbox_2)"

// Textfenster
// ===========

<Textbox>
 Position = 10;10;270;100
  Verschieben Sie die Punkte A', B', C' so,
  daß sich folgende Teilverhältnisse einstellen:
  AC' : C'B = 3 : 1
  BA' : A'C = 1 : 3
  CB' : B'A = 1 : 1 .
</Textbox>

<Textbox>
 Position = 10;310;320;40
  Lösung: Die drei grauen Ecktransversalen entsprechen
  den geforderten Teilverhältnissen.
</Textbox>

// Hilfen
// ======

<Help>
   Nutzen Sie die Hilfspunkte auf den Dreiecksseiten,
   um die Teilverhältnisse zu bestimmen.
</Help>

// Antwortanalyse
// ==============

<Problem>
 MAX_ANSWER = 4
 condition[1] = "abs(calculate(tC')-3) < 0.1"
 condition[2] = "abs(calculate(tA')-0.3) < 0.05"
 condition[3] = "abs(calculate(tB')-1) < 0.1"
 condition[4] = "abs(calculate(t)-1) < 0.1"
</Problem>

<Answer 1>
 key = "condition[1] AND condition[2] AND condition[3]"
 comment[1] = "Richtig. /n /n
              Jede der Dreiecksseiten ist in zwölf gleich große Einheiten unterteilt. /n
              Die Lage der Punkte A', B' und C' läßt sich dann wie folgt bestimmen: /n /n
              AC' : C'B  =  3 : 1  =  9 Einheiten : 3 Einheiten /n /n
              BA' : A'C  =  1 : 3  =  3 Einheiten : 9 Einheiten /n /n
              CB' : B'A  =  1 : 1  =  6 Einheiten : 6 Einheiten ."
</Answer 1>

<Answer 2>
 key = "condition[4]"
 comment[1] = "Ihre Antwort ist nicht richtig. /n
              Die drei Ecktransversalen schneiden sich zwar in einem Punkt, /n
              aber die eingestellten Teilverhältnisse stimmen noch nicht. /n
              Versuchen Sie es noch einmal."
 comment[2] = "Ihre Antwort ist immer noch nicht richtig.  /n /n Lösung:/n
              Jede der Dreiecksseiten ist in zwölf gleich große Einheiten unterteilt. /n
              Die Lage der Punkte A', B' und C' läßt sich dann wie folgt bestimmen: /n /n
              AC' : C'B  =  3 : 1  =  9 Einheiten : 3 Einheiten /n /n
              BA' : A'C  =  1 : 3  =  3 Einheiten : 9 Einheiten /n /n
              CB' : B'A  =  1 : 1  =  6 Einheiten : 6 Einheiten ."
</Answer 2>

<Answer 3>
```

```
 key = "not(condition[1])"
 comment[1] = "Ihre Antwort ist nicht richtig. /n
               Die Lage von C' stimmt noch nicht. /n
               Versuchen Sie es noch einmal."
 comment[2] = "Ihre Antwort ist immer noch nicht richtig.  /n /nLösung:/n
               Jede der Dreiecksseiten ist in zwölf gleich große Einheiten unterteilt. /n
               Die Lage der Punkte A', B' und C' läßt sich dann wie folgt bestimmen: /n /n
               AC' : C'B  =  3 : 1  =  9 Einheiten : 3 Einheiten /n /n
               BA' : A'C  =  1 : 3  =  3 Einheiten : 9 Einheiten /n /n
               CB' : B'A  =  1 : 1  =  6 Einheiten : 6 Einheiten ."
</Answer 3>

<Answer 4>
 key = "not(condition[2])"
 comment[1] = "Ihre Antwort ist nicht richtig. /n
               Die Lage von A' stimmt noch nicht. /n
               Versuchen Sie es noch einmal."
 comment[2] = "Ihre Antwort ist immer noch nicht richtig.  /n /nLösung:/n
               Jede der Dreiecksseiten ist in zwölf gleich große Einheiten unterteilt. /n
               Die Lage der Punkte A', B' und C' läßt sich dann wie folgt bestimmen: /n /n
               AC' : C'B  =  3 : 1  =  9 Einheiten : 3 Einheiten /n /n
               BA' : A'C  =  1 : 3  =  3 Einheiten : 9 Einheiten /n /n
               CB' : B'A  =  1 : 1  =  6 Einheiten : 6 Einheiten ."
</Answer 4>

<Answer 5>
 key = "not(condition[3])"
 comment[1] = "Ihre Antwort ist nicht richtig. /n
               Die Lage von B' stimmt noch nicht. /n
               Versuchen Sie es noch einmal."
 comment[2] = "Ihre Antwort ist immer noch nicht richtig.  /n /nLösung:/n
               Jede der Dreiecksseiten ist in zwölf gleich große Einheiten unterteilt. /n
               Die Lage der Punkte A', B' und C' läßt sich dann wie folgt bestimmen: /n /n
               AC' : C'B  =  3 : 1  =  9 Einheiten : 3 Einheiten /n /n
               BA' : A'C  =  1 : 3  =  3 Einheiten : 9 Einheiten /n /n
               CB' : B'A  =  1 : 1  =  6 Einheiten : 6 Einheiten ."
</Answer 5>

<Answer 6>
 key = "not(condition[4])"
 comment[1] = "Ihre Antwort ist nicht richtig. /n
               Die drei Ecktransversalen schneiden sich nicht in einem Punkt. /n
               Versuchen Sie es noch einmal."
 comment[2] = "Ihre Antwort ist immer noch nicht richtig.  /n /nLösung:/n
               Jede der Dreiecksseiten ist in zwölf gleich große Einheiten unterteilt. /n
               Die Lage der Punkte A', B' und C' läßt sich dann wie folgt bestimmen: /n /n
               AC' : C'B  =  3 : 1  =  9 Einheiten : 3 Einheiten /n /n
               BA' : A'C  =  1 : 3  =  3 Einheiten : 9 Einheiten /n /n
               CB' : B'A  =  1 : 1  =  6 Einheiten : 6 Einheiten ."
</Answer 6>
```

Dudeney (Seite 130)

```
//
// Datei: Dudeney.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
//

// Systemvariablen
// ===============

WORLD_X_MAX        = +8.0
WORLD_X_MIN        = -8.0
WORLD_Y_MAX        = +6.0
WORLD_Y_MIN        = -6.0
allPointsDragable  = true
showLabel          = false

// Figurenbeschreibung
// ===================

e[1]  = A;   point;   fixed;         1.07,0.45;                 black;red;black;smallsquare
e[2]  = A';  point;   translation;   A,0.5,0.0;                 "hidden"
e[3]  = kA;  circle;  radius;        A,A';                      "hidden"
e[4]  = A''; point;   circleSlider;  kA,1.07,0.95;
e[5]  = A1;  point;   translation;   A,0.87738267,2.00796157;   "hidden"
e[6]  = A2;  point;   translation;   A,-1.75476535,-1.00398078; "hidden"
e[7]  = A3;  point;   translation;   A,0.87738267,-1.00398078;  "hidden"
e[8]  = A1'; point;   rotation;      A1,A,A',A,A'';
e[9]  = A2'; point;   rotation;      A2,A,A',A,A'';
e[10] = A3'; point;   rotation;      A3,A,A',A,A'';
e[11] = a;   polygon; triangle;      A1',A2',A3';               0;0;blue;green
e[12] = a';  line;    connect;       A,A'';                     0;0;blue;0
e[13] = B;   point;   fixed;         -1.67,0.35;                black;red;black;smallsquare
e[14] = B';  point;   translation;   B,0.5,0.0;                 "hidden"
e[15] = kB;  circle;  radius;        B,B';                      "hidden"
e[16] = B''; point;   circleSlider;  kB,-2.37,-0.1;
e[17] = B1;  point;   translation;   B,0.317135744,2.26578602;  "hidden"
e[18] = B2;  point;   translation;   B,-0.983094590,0.74611465; "hidden"
e[19] = B3;  point;   translation;   B,-0.983094590,-1.50595034;"hidden"
e[20] = B4;  point;   translation;   B,1.649053436,-1.50595034; "hidden"
e[21] = B1'; point;   rotation;      B1,B,B',B,B'';
e[22] = B2'; point;   rotation;      B2,B,B',B,B'';
e[23] = B3'; point;   rotation;      B3,B,B',B,B'';
e[24] = B4'; point;   rotation;      B4,B,B',B,B'';
e[25] = b;   polygon; quadrilateral; B1',B2',B3',B4';           0;0;blue;green
e[26] = b';  line;    connect;       B,B'';                     0;0;blue;0
e[27] = C;   point;   fixed;         -1.47,-2.30;               black;red;black;smallsquare
e[28] = C';  point;   translation;   C,0.5,0.0;                 "hidden"
e[29] = kC;  circle;  radius;        C,C';                      "hidden"
e[30] = C''; point;   circleSlider;  kC,-0.97,-2.3;
e[31] = C1;  point;   translation;   C,2.08612308,2.28533245;   "hidden"
e[32] = C2;  point;   translation;   C,-1.69935514,0.99298786;  "hidden"
e[33] = C3;  point;   translation;   C,-1.69935514,-1.63916015; "hidden"
e[34] = C4;  point;   translation;   C,1.31258720,-1.63916015;  "hidden"
e[35] = C1'; point;   rotation;      C1,C,C',C,C'';
e[36] = C2'; point;   rotation;      C2,C,C',C,C'';
e[37] = C3'; point;   rotation;      C3,C,C',C,C'';
e[38] = C4'; point;   rotation;      C4,C,C',C,C'';
e[39] = c;   polygon; quadrilateral; C1',C2',C3',C4';           0;0;blue;green
e[40] = c';  line;    connect;       C,C'';                     0;0;blue;0
e[41] = D;   point;   fixed;         1.13, -2.30;               black;red;black;smallsquare
e[42] = D';  point;   translation;   D,0.5,0.0;                 "hidden"
e[43] = kD;  circle;  radius;        D,D';                      "hidden"
e[44] = D''; point;   circleSlider;  kD,1.13,-1.8;
e[45] = D1;  point;   translation;   D,2.29124676,0.576737278;  "hidden"
e[46] = D2;  point;   translation;   D,-1.64113159,1.30913090;  "hidden"
e[47] = D3;  point;   translation;   D,-1.64113159,-0.942934092;"hidden"
e[48] = D4;  point;   translation;   D,0.99101642,-0.942934092; "hidden"
e[49] = D1'; point;   rotation;      D1,D,D',D,D'';
e[50] = D2'; point;   rotation;      D2,D,D',D,D'';
e[51] = D3'; point;   rotation;      D3,D,D',D,D'';
```

```
e[52] = D4'; point;   rotation;      D4,D,D',D,D'';
e[53] = d;   polygon; quadrilateral; D1',D2',D3',D4';          0;0;blue;green
e[54] = d';  line;    connect;       D,D'';                    0;0;blue;0

// Textfenster
// ===========

<Textbox>
 Textbox = 10;10;-1;-1
  Ernest Dudeney demonstrierte 1905 vor der Royal Society
  in London die Zerlegung des gleichseitigen Dreiecks in
  ein Quadrat. Wie kann man die Teilfiguren wieder zu einem
  Dreieck und einem Quadrat zusammensetzen?
</Textbox>
```

Satz von Holditch (Seite 132)

```
//
// Datei:    Satz_von_Holditch.script
// Autor:    Timo Ehmke (ehmke@uni-flensburg.de)
//

// Systemvariablen
// ===============

APPLET_WIDTH  = 640
APPLET_HEIGHT = 480
WORLD_X_MAX   = +16.0
WORLD_X_MIN   = -16.0
WORLD_Y_MAX   = +12.0
WORLD_Y_MIN   = -12.0

// Figurenbeschreibung
// ===================

e[1]  = Pset; line;   pointSet;          "./pics/Eilinie 640x480.gif"; 0;blue;blue;0
e[2]  = O;    point;  curveSlider;       0.0,-2.0,Pset;        black;red;black;circle
e[3]  = O';   point;  fixed;             -14.0,10.0;
e[4]  = S';   point;  horizontal;        O',-2.0;
e[5]  = s';   line;   connect;           O',S';                0;0;blue;0
e[6]  = X';   point;  lineSegmentSlider; s',-9.0,14.0;
e[7]  = k;    circle; radius;            O,O',S';              "hidden"
e[8]  = S;    point;  intersection;      k,Pset;               black;blue;black;circle
e[9]  = s;    line;   connect;           O,S;                  black;0;black;0
e[10] = X;    point;  cutoff;            O,O,S,O',X';          black;green;black;smallcircle
e[11] = x;    line;   connect;           X,O;                  black;0;black;0
e[12] = y;    line;   connect;           X,S;                  black;0;black;0

// Textfenster
// ===========

<TextBox>
 TextBox = 310;10;-1;-1
 Satz von Holditch

 Die Stange mit der Länge s kann durch Verschieben
 von Punkt O entlang der gegebenen Eilinie bewegt
 werden. Man betrachte die Bahnkurve des Punkts X.
</TextBox>

<TextBox>
 TextBox = 20;400;-1;-1
  Von welchen Faktoren hängt die Fläche
  des Ringgebiets (überraschenderweise)
  nur ab?
</TextBox>
```

Würfelnetze (Seite 134)

```
//
// Datei: Wuerfelnetze.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
//

// Systemvariablen
// ===============

APPLET_WIDTH  = 640
APPLET_HEIGHT = 480
WORLD_X_MAX   = +8.0
WORLD_X_MIN   = -8.0
WORLD_Y_MAX   = +6.0
WORLD_Y_MIN   = -6.0
gridSize      = 40
snapToGrid    = true
showGrid      = true

// Figurenbeschreibung
// ===================

e[1]  = A;    point;   free;        -1.0,0.0;   "hidden"
e[2]  = B;    point;   free;        1.0,0.0;    "hidden"
e[3]  = C;    point;   free;        -3.0,0.0;   "hidden"
e[4]  = D;    point;   free;        -1.0,-2.0;  "hidden"
e[5]  = E;    point;   free;        3.0,0.0;    "hidden"
e[6]  = F;    point;   free;        -1.0,2.0;   "hidden"
e[7]  = A';   point;   translation; A,1.0,1.0;  "hidden"
e[8]  = A'';  point;   translation; A,-1.0,1.0; "hidden"
e[9]  = SqA;  polygon; square;      A',A'';     0;0;black;green
e[10] = A'''; point;   translation; A,0.0,0.0;  0;red;black;smallcircle
e[11] = B';   point;   translation; B,1.0,1.0;  "hidden"
e[12] = B'';  point;   translation; B,-1.0,1.0; "hidden"
e[13] = SqB;  polygon; square;      B',B'';     0;0;black;green
e[14] = B'''; point;   translation; B,0.0,0.0;  0;red;black;smallcircle
e[15] = C';   point;   translation; C,1.0,1.0;  "hidden"
e[16] = C'';  point;   translation; C,-1.0,1.0; "hidden"
e[17] = SqC;  polygon; square;      C',C'';     0;0;black;green
e[18] = C'''; point;   translation; C,0.0,0.0;  0;red;black;smallcircle
e[19] = D';   point;   translation; D,1.0,1.0;  "hidden"
e[20] = D'';  point;   translation; D,-1.0,1.0; "hidden"
e[21] = SqD;  polygon; square;      D',D'';     0;0;black;green
e[22] = D'''; point;   translation; D,0.0,0.0;  0;red;black;smallcircle
e[23] = E';   point;   translation; E,1.0,1.0;  "hidden"
e[24] = E'';  point;   translation; E,-1.0,1.0; "hidden"
e[25] = SqE;  polygon; square;      E',E'';     0;0;black;green
e[26] = E'''; point;   translation; E,0.0,0.0;  0;red;black;smallcircle
e[27] = F';   point;   translation; F,1.0,1.0;  "hidden"
e[28] = F'';  point;   translation; F,-1.0,1.0; "hidden"
e[29] = SqF;  polygon; square;      F',F'';     0;0;black;green
e[30] = F'''; point;   translation; F,0.0,0.0;  0;red;black;smallcircle
e[31] = m0;   measure; function;    "Functional_Wuerfelnetz",
              "A","B","C","D","E","F",-50.0,-5.0,"NetzAnalyse = ","";
e[32] = m1;   measure; button;      "Auswertung","evaluate";

// Textfenster
// ===========

<Textbox>
 Position = 20;20;-1;-1
  Verschieben Sie die sechs Quadrate so, daß
  gültige Würfelnetze entstehen.
</Textbox>

// Antwortanalyse
// ==============

<Problem>
 MAX_ANSWER = 0
```

```
 condition[1] = "calculate(m0) = 1"
 condition[2] = "calculate(m0) = -1"
 condition[3] = "calculate(m0) < -1"
</Problem>

<Answer 1>
 key = "condition[1]"
 comment[1] = "Richtig. /n /nAus diesem Netz kann /nein Würfel gefaltet werden."
 </Answer 1>

<Answer 2>
 key = "condition[2]"
 comment[1] = "Ihre Antwort ist nicht richtig. /nDas Netz ist nicht zusammenhängend. /n
              Versuchen Sie es noch einmal."
</Answer 2>

<Answer 3>
 key = "condition[3]"
 comment[1] = "Ihre Antwort ist nicht richtig. /nDieses Netz läßt sich nicht zu /n
              einem Würfel falten. /nVersuchen Sie es noch einmal."
</Answer 3>
```

Haus der Vierecke (Seite 135)

```
//
// Datei: Haus_der_Vierecke.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
// Aufgabenidee findet sich u. a. bei
// Neubrand [u. a.]: Tendenzen der Geometriedidaktik der letzten 20 Jahre. S. 212
// Journal für Mathematikdidaktik, Jg. 17 (1996), Heft 3/4
//

// Systemvariablen
// ===============

APPLET_WIDTH  = 640
APPLET_HEIGHT = 480
showGrid      = true
snapToGrid    = true

// Figurenbeschreibung
// ===================

e[1]  = A;   point;   free;     -6.0,4.5;
e[2]  = B;   point;   free;     -6.0,8.0;
e[3]  = C;   point;   free;     -9.5,7.0;
e[4]  = D;   point;   free;     -11.0,4.0;
e[5]  = p;   polygon; polygon;  A,B,C,D;     0;0;blue;yellow
e[6]  = e;   line;    connect;  A,C;         0;0;lightGray;0
e[7]  = f;   line;    connect;  B,D;         0;0;lightGray;0
e[8]  = w1;  sector;  angle;    B,A,D,0,0;   0;0;black;0
e[9]  = w2;  sector;  angle;    C,B,A,0,0;   0;0;black;0
e[10] = w3;  sector;  angle;    D,C,B,1,1;   0;0;black;0
e[11] = w4;  sector;  angle;    A,D,C,1,1;   0;0;black;0
e[12] = sw1; measure; checkbox; "Diagonalen zeigen",0;
e[13] = sw2; measure; checkbox; "Winkel zeigen",0;
e[14] = M1;  point;   midpoint; A,C;         0;lightGray;lightGray;smallcircle
e[15] = M2;  point;   midpoint; B,D;         0;lightGray;lightGray;smallcircle
e[16] = A1;  point;   fixed;    8.5,6.5;     "hidden"
e[17] = B1;  point;   fixed;    6.5,6.5;     "hidden"
e[18] = C1;  point;   fixed;    4.0,3.5;     "hidden"
e[19] = D1;  point;   fixed;    9.5,3.5;     "hidden"
e[20] = p1;  polygon; polygon;  A1,B1,C1,D1; 0;0;black;0
e[21] = A2;  point;   fixed;    1.5,-2.5;    "hidden"
e[22] = B2;  point;   fixed;    2.5,0.5;     "hidden"
e[23] = C2;  point;   fixed;    -1.5,0.5;    "hidden"
e[24] = D2;  point;   fixed;    -2.5,-2.5;   "hidden"
e[25] = p2;  polygon; polygon;  A2,B2,C2,D2; 0;0;black;0
e[26] = A3;  point;   fixed;    8.0,-2.5;    "hidden"
e[27] = B3;  point;   fixed;    12.0,-2.5;   "hidden"
e[28] = C3;  point;   fixed;    11.0,0.5;    "hidden"
e[29] = D3;  point;   fixed;    9.0,0.5;     "hidden"
e[30] = p3;  polygon; polygon;  A3,B3,C3,D3; 0;0;black;0
e[31] = A4;  point;   fixed;    1.5,-8.0;    "hidden"
e[32] = B4;  point;   fixed;    1.5,-11.0;   "hidden"
e[33] = C4;  point;   fixed;    -1.5,-11.0;  "hidden"
e[34] = D4;  point;   fixed;    -1.5,-8.0;   "hidden"
e[35] = p4;  polygon; polygon;  A4,B4,C4,D4; 0;0;black;0
e[36] = A5;  point;   fixed;    -5.0,-6.5;   "hidden"
e[37] = B5;  point;   fixed;    -6.5,-9.0;   "hidden"
e[38] = C5;  point;   fixed;    -8.0,-6.5;   "hidden"
e[39] = D5;  point;   fixed;    -6.5,-4.0;   "hidden"
e[40] = p5;  polygon; polygon;  A5,B5,C5,D5; 0;0;black;0
e[41] = A6;  point;   fixed;    -9.0,0.0;    "hidden"
e[42] = B6;  point;   fixed;    -11.0,-3.0;  "hidden"
e[43] = C6;  point;   fixed;    -13.0,0.0;   "hidden"
e[44] = D6;  point;   fixed;    -11.0,1.0;   "hidden"
e[45] = p6;  polygon; polygon;  A6,B6,C6,D6; 0;0;black;0
e[46] = A7;  point;   fixed;    1.0,11.0;    "hidden"
e[47] = B7;  point;   fixed;    -1.5,10.0;   "hidden"
e[48] = C7;  point;   fixed;    -1.5,8.0;    "hidden"
e[49] = D7;  point;   fixed;    2.0,6.5;     "hidden"
e[50] = p7;  polygon; polygon;  A7,B7,C7,D7; 0;0;black;0
```

```
e[51] = A8;  point;   fixed;    8.5,-5.5;    "hidden"
e[52] = B8;  point;   fixed;    8.5,-7.5;    "hidden"
e[53] = C8;  point;   fixed;    4.5,-7.5;    "hidden"
e[54] = D8;  point;   fixed;    4.5,-5.5;    "hidden"
e[55] = p8;  polygon; polygon;  A8,B8,C8,D8; 0;0;black;0
e[56] = S1;  point;   fixed;    5.0,6.0;     "hidden"
e[57] = S2;  point;   fixed;    3.0,8.0;     "hidden"
e[58] = s1;  line;    connect;  S1,S2;       "hideLabel"
e[59] = S3;  point;   fixed;    -10.0,1.0;   "hidden"
e[60] = S4;  point;   fixed;    -8.0,3.0;    "hidden"
e[61] = s2;  line;    connect;  S3,S4;       "hideLabel"
e[62] = S5;  point;   fixed;    -5.0,3.0;    "hidden"
e[63] = S6;  point;   fixed;    -3.0,1.0;    "hidden"
e[64] = s3;  line;    connect;  S5,S6;       "hideLabel"
e[65] = S7;  point;   fixed;    3.0,1.0;     "hidden"
e[66] = S8;  point;   fixed;    5.0,3.0;     "hidden"
e[67] = s4;  line;    connect;  S7,S8;       "hideLabel"
e[68] = S9;  point;   fixed;    10.0,1.0;    "hidden"
e[69] = S10; point;   fixed;    8.0,3.0;     "hidden"
e[70] = s5;  line;    connect;  S9,S10;      "hideLabel"
e[71] = S11; point;   fixed;    -3.0,-3.0;   "hidden"
e[72] = S12; point;   fixed;    -5.0,-5.0;   "hidden"
e[73] = s6;  line;    connect;  S11,S12;     "hideLabel"
e[74] = S13; point;   fixed;    -10.0,-3.0;  "hidden"
e[75] = S14; point;   fixed;    -8.0,-5.0;   "hidden"
e[76] = s7;  line;    connect;  S13,S14;     "hideLabel"
e[77] = S15; point;   fixed;    3.0,-3.0;    "hidden"
e[78] = S16; point;   fixed;    5.0,-5.0;    "hidden"
e[79] = s8;  line;    connect;  S15,S16;     "hideLabel"
e[80] = S17; point;   fixed;    8.0,-5.0;    "hidden"
e[81] = S18; point;   fixed;    10.0,-3.0;   "hidden"
e[82] = s9;  line;    connect;  S17,S18;     "hideLabel"
e[83] = S19; point;   fixed;    3.0,-10.0;   "hidden"
e[84] = S20; point;   fixed;    5.0,-8.0;    "hidden"
e[85] = s10; line;    connect;  S19,S20;     "hideLabel"
e[86] = S21; point;   fixed;    -5.0,-8.0;   "hidden"
e[87] = S22; point;   fixed;    -3.0,-10.0;  "hidden"
e[88] = s11; line;    connect;  S21,S22;     "hideLabel"
e[89] = S23; point;   fixed;    -5.0,6.0;    "hidden"
e[90] = S24; point;   fixed;    -3.0,8.0;    "hidden"
e[91] = s12; line;    connect;  S23,S24;     "hideLabel"
e[92] = b1;  measure; button;  "Hilfe","help";
e[93] = b2;  measure; button;  "Auswerten","evaluate";

// Ein- und Ausblenden von Objekten
// ================================

hidden[1] = "if (not(calculate(sw1))) hide (e,f,M1,M2)"
hidden[2] = "if (not(calculate(sw2))) hide (w1,w2,w3,w4)"

// Hilfen
// =======

<Help>
   Betrachte die gegenüberliegenden Winkel
   im Viereck.
</Help>

// Antwortanalyse
// ==============

<Problem>
 MAX_ANSWER = 0
 condition[1] = "$(p,"bestClass") = 9"
 condition[2] = "$(p,"bestClass") = 8"
 condition[3] = "$(p,"bestClass") = 1"
 condition[4] = "$(p,"bestClass") = 2"
 condition[5] = "$(p,"bestClass") = 3"
 condition[6] = "$(p,"bestClass") = 4"
 condition[7] = "$(p,"bestClass") = 5"
 condition[8] = "$(p,"bestClass") = 6"
```

```
 condition[9] = "$(p,"bestClass") = 7"
 condition[10] = "$(p,"bestClass") = 0"
</Problem>

<Answer 1>
 key = "condition[1]"
 comment[1] = "Richtig. /nDas Viereck ist ein 'Schräger Drache',/n
              d. h. eine Diagonale wird durch die zweite halbiert./n
              Möglich wäre auch ein 'Schiefer Drache' gewesen./n
              Bei diesem gibt es zwei gegenüberliegende gleich große Winkel."
</Answer 1>

<Answer 2>
 key = "condition[2]"
 comment[1] = "Richtig. /nDas Viereck ist ein 'Schiefer Drache',/n
              d. h. zwei gegenüberliegende Winkel sind gleich groß./n
              Möglich wäre auch ein 'Schräger Drache' gewesen./n
              Bei diesem wird eine Diagonale wird durch die zweite halbiert."
</Answer 2>

<Answer 3>
 key = "condition[3]"
 comment[1] = "Ihre Antwort ist nicht richtig. /n
              Das Viereck stellt ein Quadrat dar./n
              Versuchen Sie es noch einmal."
 comment[2] = "Ihre Antwort ist immer noch nicht richtig. /n
              Das Viereck stellt ein Quadrat dar./n
              Dies ist nicht der gesuchte Viereckstyp./n
              Versuchen Sie es noch einmal."
 comment[3] = "Ihre Antwort ist immer noch nicht richtig. /n
              Das Viereck stellt ein Quadrat dar./n
              Dies ist nicht der gesuchte Viereckstyp./n
              In die Lücke würde ein 'Schräger Drache' oder /nein 'Schiefer Drache' passen."
</Answer 3>

<Answer 4>
 key = "condition[4]"
 comment[1] = "Ihre Antwort ist nicht richtig. /n
              Das Viereck stellt ein Rechteck dar./n
              Versuchen Sie es noch einmal."
 comment[2] = "Ihre Antwort ist immer noch nicht richtig. /n
              Das Viereck stellt ein Rechteck dar./n
              Dies ist nicht der gesuchte Viereckstyp./n
              Versuchen Sie es noch einmal."
 comment[3] = "Ihre Antwort ist immer noch nicht richtig. /n
              Das Viereck stellt ein Rechteck dar./n
              Dies ist nicht der gesuchte Viereckstyp./n
              In die Lücke würde ein 'Schräger Drache' oder /nein 'Schiefer Drache' passen."
</Answer 4>

<Answer 5>
 key = "condition[5]"
 comment[1] = "Ihre Antwort ist nicht richtig. /n
              Das Viereck stellt eine Raute dar./n
              Versuchen Sie es noch einmal."
 comment[2] = "Ihre Antwort ist immer noch nicht richtig. /n
              Das Viereck stellt eine Raute dar./n
              Dies ist nicht der gesuchte Viereckstyp./n
              Versuchen Sie es noch einmal."
 comment[3] = "Ihre Antwort ist immer noch nicht richtig. /n
              Das Viereck stellt eine Raute dar./n
              Dies ist nicht der gesuchte Viereckstyp./n
              In die Lücke würde ein 'Schräger Drache' oder /nein 'Schiefer Drache' passen."
</Answer 5>

<Answer 6>
 key = "condition[6]"
 comment[1] = "Ihre Antwort ist nicht richtig. /n
              Das Viereck stellt ein Parallelogramm dar./n
              Versuchen Sie es noch einmal."
 comment[2] = "Ihre Antwort ist immer noch nicht richtig. /n
```

```
              Das Viereck stellt ein Parallelogramm dar./n
              Dies ist nicht der gesuchte Viereckstyp./n
              Versuchen Sie es noch einmal."
 comment[3] = "Ihre Antwort ist immer noch nicht richtig. /n
              Das Viereck stellt ein Parallelogramm dar./n
              Dies ist nicht der gesuchte Viereckstyp./n
              In die Lücke würde ein 'Schräger Drache' oder /nein 'Schiefer Drache' passen."
</Answer 6>

<Answer 7>
 key = "condition[7]"
 comment[1] = "Ihre Antwort ist nicht richtig. /n
              Das Viereck stellt ein gleichschenkliges Trapez dar./n
              Versuchen Sie es noch einmal."
 comment[2] = "Ihre Antwort ist immer noch nicht richtig. /n
              Das Viereck stellt ein gleichschenkliges Trapez dar./n
              Dies ist nicht der gesuchte Viereckstyp./n
              Versuchen Sie es noch einmal."
 comment[3] = "Ihre Antwort ist immer noch nicht richtig. /n
              Das Viereck stellt ein gleichschenkliges Trapez dar./n
              Dies ist nicht der gesuchte Viereckstyp./n
              In die Lücke würde ein 'Schräger Drache' oder /nein 'Schiefer Drache' passen."
</Answer 7>

<Answer 8>
 key = "condition[8]"
 comment[1] = "Ihre Antwort ist nicht richtig. /n
              Das Viereck stellt ein schiefes Trapez dar./n
              Versuchen Sie es noch einmal."
 comment[2] = "Ihre Antwort ist immer noch nicht richtig. /n
              Das Viereck stellt ein schiefes Trapez dar./n
              Dies ist nicht der gesuchte Viereckstyp./n
              Versuchen Sie es noch einmal."
 comment[3] = "Ihre Antwort ist immer noch nicht richtig. /n
              Das Viereck stellt ein schiefes Trapez dar./n
              Dies ist nicht der gesuchte Viereckstyp./n
              In die Lücke würde ein 'Schräger Drache' oder /nein 'Schiefer Drache' passen."
</Answer 8>

<Answer 9>
 key = "condition[9]"
 comment[1] = "Ihre Antwort ist nicht richtig. /n
              Das Viereck stellt einen gleichschenkligen Drachen dar./n
              Versuchen Sie es noch einmal."
 comment[2] = "Ihre Antwort ist immer noch nicht richtig. /n
              Das Viereck stellt einen gleichschenkligen Drachen dar./n
              Dies ist nicht der gesuchte Viereckstyp./n
              Versuchen Sie es noch einmal."
 comment[3] = "Ihre Antwort ist immer noch nicht richtig. /n
              Das Viereck stellt einen gleichschenkligen Drachen dar./n
              Dies ist nicht der gesuchte Viereckstyp./n
              In die Lücke würde ein 'Schräger Drache' oder /nein 'Schiefer Drache' passen."
</Answer 9>

<Answer 10>
 key = "condition[10]"
 comment[1] = "Ihre Antwort ist nicht richtig. /n
              Das Viereck kann nicht klassifiziert werden./n
              Versuchen Sie es noch einmal."
 comment[2] = "Ihre Antwort ist immer noch nicht richtig. /n
              Das Viereck kann nicht klassifiziert werden./n
              Versuchen Sie es noch einmal."
 comment[3] = "Ihre Antwort ist immer noch nicht richtig. /n
              Das Viereck kann nicht klassifiziert werden./n
              In die Lücke würde ein 'Schräger Drache' oder /nein 'Schiefer Drache' passen."
</Answer 10>
```

Gleichseitiges Dreieck im Quadrat (Seite 138)

```
//
// Datei: Gleichseitiges_Dreieck_im_Quadrat.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
// Aufgabenidee findet sich u. a. bei
// Hölzl: "Im Zugmodus der Cabri-Geometrie", Weinheim 1994, S. 80f
//

// Systemvariablen
// ===============

APPLET_WIDTH  = 640
APPLET_HEIGHT = 480
WORLD_X_MAX   = +16.0
WORLD_X_MIN   = -16.0
WORLD_Y_MAX   = +12.0
WORLD_Y_MIN   = -12.0

// Figurenbeschreibung
// ===================

e[1]  = P1; point;   fixed;      -11.0,-6.0;     "hidden"
e[2]  = P2; point;   fixed;      1.0,-6.0;       "hidden"
e[3]  = q;  polygon; square;     P1,P2;          0;0;black;yellow
e[4]  = A;  point;   areaSlider; -2.2,-1.0,q;
e[5]  = B;  point;   areaSlider; 3.1,-0.9,q;
e[6]  = C;  point;   rotation;   A,B,1.04719551;
e[7]  = d;  polygon; triangle;   A,B,C;          0;0;black;lightGray
e[8]  = m0; measure; property;   d,"area",-1.0,8.0,"Dreiecksfläche = "," FE";
e[9]  = m1; measure; button;     "Hilfe","help";
e[10] = m2; measure; button;     "Auswertung","evaluate";

// Beschränken des Zustandsraums der Figur
// =======================================
limit[1] = "isIncluded(C,q)"

// Textfenster
// ===========

<Textbox>
 Position = 20;20;-1;-1
   Aufgabe:
   Das gleichseitige Dreieck soll dem
   Quadrat so einbeschrieben werden, daß
   der Flächeninhalt möglichst groß wird.
</Textbox>

// Hilfen
// ======

<Help>
   Variiere das Dreieck durch Verschieben
   der beiden roten Eckpunkte.
   Achte auf die Änderung des Flächeninhalts.
</Help>

// Antwortanalyse
// ==============

<Problem>
 MAX_ANSWER = 0
 condition[1] = "area(d) > 66.4"
 condition[2] = "area(d) > 66.0"
 condition[3] = "area(d) = 62.4"
 condition[4] = "area(d) < 62.4"
 condition[5] = "area(d) > 62.4"
</Problem>

<Answer 1>
 key = "condition[1]"
```

```
 comment[1] = "Richtig. /nIn dieser Lage hat das Dreieck /nden größten Flächeninhalt."
</Answer 1>

<Answer 2>
 key = "condition[2]"
 comment[1] = "Ihre Antwort ist nicht richtig. /n
              Ihre Lösung ist aber schon nahe dran. /n
              Die Dreiecksfläche läßt sich jedoch noch etwas vergrößern. /n
              Versuchen Sie es noch einmal."
</Answer 2>

<Answer 3>
 key = "condition[3]"
 comment[1] = "Ihre Antwort ist nicht richtig. /n
              Auf den ersten Blick scheint dies /nzwar eine gute Lösung zu sein, /n
              aber versuchen Sie das Dreieck noch /n
              anders einzupassen."
</Answer 3>

<Answer 4>
 key = "condition[4]"
 comment[1] = "Ihre Antwort ist nicht richtig. /n
              Das ist auch bestimmt nicht ihre beste Lösung. /n
              Versuchen Sie es noch einmal."
</Answer 4>

<Answer 5>
 key = "condition[5] AND (NOT(condition[2]))"
 comment[1] = "Ihre Antwort ist nicht richtig. /n
              Die Dreiecksfläche läßt sich noch etwas vergrößern. /n
              Versuchen Sie es noch einmal."
</Answer 5>
```

Verhältnis zweier Quadrate (Seite 140)

```
//
// Datei: Verhaeltnis_zweier_Quadrate.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
//

// Figurenbeschreibung
// ===================

e[1]  = A;   point;   free;           -4.0,5.0;
e[2]  = B;   point;   free;           -14.0,5.0;
e[3]  = q1;  polygon; square;         A,B;             0;0;black;green
e[4]  = S;   point;   functionDepend; "$(q1,center_x)","$(q1,center_y)";
e[5]  = M;   point;   midpoint;       A,B;             "hidden"
e[6]  = k;   circle;  radius;         S,M;             0;0;black;yellow
e[7]  = k2;  circle;  radius;         S,M;             0;0;black;green
e[8]  = R;   point;   circleSlider;   -9.0,5.0,k;
e[9]  = R';  point;   rotation;       R,S,1.570796327; "hidden"
e[10] = q2;  polygon; square;         R',R;            0;0;black;lightGray
e[11] = P1;  point;   mirror;         R,S;             "hidden"
e[12] = P2;  point;   mirror;         R',S;            "hidden"
e[13] = d1;  line;    connect;        R,P1;            0;0;black;0
e[14] = d2;  line;    connect;        R',P2;           0;0;black;0
e[15] = sw1; measure; checkbox;       "Lösung zeigen",0;

// Ein- und Ausblenden von Objekten
// ================================

hidden[1] = "if (not(calculate(sw1))) hide (d1,d2,k2)"

// Textfenster
// ===========

<ProblemText>
 Position = 20;20;-1;-1
   Aufgabe:

   Wie ist der Flächeninhalt des kleinen
   Quadrats von dem Flächeninhalt des
   großen Quadrats abhängig?
</ProblemText>
```

Dreieck mit Kreisen und Quadraten (Seite 142)

```
//
// Datei: Dreieck_mit_Kreisen_und_Quadraten.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
//

// Figurenbeschreibung
// ===================

e[1]  = G1; point;   fixed;          6.0,0.0;         "hidden"
e[2]  = G2; point;   fixed;          -6.0,0.0;        "hidden"
e[3]  = g;  line;    straightline;   G1,G2;           "hidden"
e[4]  = A;  point;   free;           4.0,9.0;         black;red;black;smallcircle
e[5]  = B;  point;   lineSlider;     -10.0,0.0,g;     black;red;black;smallcircle
e[6]  = C;  point;   foot;           A,g;             black;blue;blue;smallcircle
e[7]  = p1; polygon; triangle;       A,B,C;           0;0;black;green
e[8]  = q;  measure; angle;          C,B,A,-1400.0,10.0,"<CBA = ","";
e[9]  = t1; measure; calculate;      "distance(A,C)/(sin(calculate(q)*3.141592654/180)+
           (1/cos(calculate(q)*3.141592654/180)))";
e[10] = BD; measure; calculate;      "distance(A,B)*calculate(t1)/distance(A,C)";
e[11] = D;  point;   functionDepend; "coordinateX(B)+calculate(BD)",
           "coordinateY(B)";                      "hidden"
e[12] = E;  point;   proportion;     A,B,B,C,B,D,B,A; "hidden"
e[13] = F;  point;   proportion;     B,C,D,C,A,C,C,A; "hidden"
e[14] = G;  point;   proportion;     A,B,A,C,A,F,A,B; "hidden"
e[15] = p2; polygon; quadrilateral;  E,D,F,G;         0;0;black;lightGray
e[16] = I;  point;   proportion;     A,C,C,F,E,D,E,D; "hidden"
e[17] = H;  point;   proportion;     C,B,C,D,E,B,E,B; "hidden"
e[18] = K;  point;   proportion;     A,B,A,G,D,B,D,B; "hidden"
e[19] = L;  point;   proportion;     A,B,B,H,B,C,B,C; "hidden"
e[20] = p3; polygon; quadrilateral;  K,I,H,L;         0;0;black;lightGray
e[21] = M;  point;   proportion;     C,A,C,F,L,H,L,H; "hidden"
e[22] = N;  point;   proportion;     A,B,A,G,H,B,H,B; "hidden"
e[23] = O;  point;   proportion;     A,B,A,E,H,B,H,B; "hidden"
e[24] = P;  point;   proportion;     B,C,B,D,B,L,B,C; "hidden"
e[25] = p4; polygon; quadrilateral;  M,N,O,P;         0;0;black;lightGray
e[26] = a;  line;    angleBisector;  G,A,F;           "hidden"
e[27] = b;  line;    angleBisector;  A,F,G;           "hidden"
e[28] = Q;  point;   intersection;   a,b;             "hidden"
e[29] = R;  point;   foot;           Q,A,C;           "hidden"
e[30] = k1; circle;  radius;         Q,R;             0;0;black;red
e[31] = c;  line;    angleBisector;  I,D,K;           "hidden"
e[32] = d;  line;    angleBisector;  K,I,D;           "hidden"
e[33] = S;  point;   intersection;   c,d;             "hidden"
e[34] = T;  point;   foot;           S,B,C;           "hidden"
e[35] = k2; circle;  radius;         S,T;             0;0;black;red
e[36] = e;  line;    angleBisector;  H,M,N;           "hidden"
e[37] = f;  line;    angleBisector;  N,H,M;           "hidden"
e[38] = U;  point;   intersection;   e,f;             "hidden"
e[39] = V;  point;   foot;           U,B,A;           "hidden"
e[40] = k3; circle;  radius;         U,V;             0;0;black;red

// Textfenster
// ===========

<Textbox>
 TextBox = 10;10;-1;-1
   Wie verhalten sich die Radien
   der drei Kreise zueinander?
</Textbox>
```

Dreieck mit Gerade und zwei Kreisen (Seite 142)

```
//
// Datei: Dreieck_mit_Gerade_und_zwei_Kreisen.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
//

// Figurenbeschreibung
// ===================

e[1]  = A;  point;   free;          -4.13,-4.67;
e[2]  = B;  point;   free;          4.13,-4.67;
e[3]  = C;  point;   rotation;      A,B,1.04719,1.0;
e[4]  = D;  point;   free;          9.47,0.4;
e[5]  = D'; point;   mirror;        D,C;    "hidden"
e[6]  = g;  line;    straightLine;  A,B;    "hideLabel"
e[7]  = h;  line;    straightLine;  C,D;    "hideLabel"
e[8]  = b;  line;    connect;       A,C;    "hideLabel"
e[9]  = c;  line;    connect;       C,B;    "hideLabel"
e[10] = wC'; line;   angleBisector; D',C,A; "hidden"
e[11] = wC; line;    angleBisector; B,C,D;  "hidden"
e[12] = c'; line;    parallel;      A,c;    "hidden"
e[13] = b'; line;    parallel;      B,b;    "hidden"
e[14] = E;  point;   intersection;  c',wC'; "hidden"
e[15] = F;  point;   intersection;  b',wC;  "hidden"
e[16] = G;  point;   foot;          E,g;    "hidden"
e[17] = H;  point;   foot;          F,g;    "hidden"
e[18] = k1; circle;  radius;        E,G;    0;0;black;green
e[19] = k2; circle;  radius;        F,H;    0;0;black;blue
e[20] = dr; polygon; triangle;      A,B,C;  0;0;black;red

// Textfenster
// ===========

<TextBox>
 TextBox = 10;10;-1;-1
   Wie groß ist die Summe der
   beiden Radien?
</TextBox>
```

Drei Kreise auf einer Geraden (Seite 143)

```
//
// Datei: Drei_Kreise_auf_einer_Geraden.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
//

// Figurenbeschreibung
// ===================

e[1]  = G1;  point;   fixed;          6.0,0.0;   "hidden"
e[2]  = G2;  point;   fixed;          -6.0,0.0;  "hidden"
e[3]  = t;   line;    straightline;   G1,G2;     0;0;black;0
e[4]  = M1;  point;   free;           -5.5,5.1;  0;red;black;smallcircle
e[5]  = A;   point;   foot;           M1,t;      "hidden"
e[6]  = r1'; measure; distance;       M1,A,9.0,11.0,"r1 = ","";
e[7]  = C;   point;   lineSlider;     1.5,0.0,t; 0;red;black;smallcircle
e[8]  = r3'; measure; calculate;      "(distance(A,C)^2)/(4*calculate(r1'))",
             9.0,9.0,"r3 = ","";
e[9]  = M3;  point;   functionDepend;
             "coordinateX(C)","coordinateY(C)+calculate(r3')"; "hidden"
e[10] = k1;  circle;  radius;         M1,A;      0;0;black;green
e[11] = k3;  circle;  radius;         M3,C;      0;0;black;blue
e[12] = r2'; measure; calculate;
             "(calculate(r1')*calculate(r3'))/(sqrt(calculate(r1'))+
             sqrt(calculate(r3')))^2",9.0,10.0,"r2 = ","";
e[13] = AB;  measure; calculate;      "2*sqrt(calculate(r1')*calculate(r2'))";
e[14] = B;   point;   functionDepend; "coordinateX(A)+calculate(AB)",
             "coordinateY(A)";  "hidden"
e[15] = M2;  point;   functionDepend; "coordinateX(B)",
             "coordinateY(B)+calculate(r2')"; "hidden"
e[16] = k2;  circle;  radius;         M2,B;      0;0;black;red
e[17] = m0;  measure; checkbox;       "Hilfe zeigen",0;
e[18] = r1;  line;    connect;        M1,A;      black;0;black;0
e[19] = r2;  line;    connect;        M2,B;      black;0;black;0
e[20] = r3;  line;    connect;        M3,C;      black;0;black;0
e[21] = m1;  measure; checkbox;       "Lösung zeigen",0;
e[22] = m2;  measure; calculate;      "1/sqrt(calculate(r2'))",9.0,8.0,"1/sqrt(r2) = ","";
e[23] = m3;  measure; calculate;      "(1/sqrt(calculate(r1')))+(1/sqrt(calculate(r3')))",
             9.0,7.0,"1/sqrt(r1) + 1/sqrt(r3) = ","";

// Ein- und Ausblenden von Objekten
// ================================

hidden[1] = "if (not(calculate(m0))) hide (r1',r2',r3',r1,r2,r3)"
hidden[2] = "if (not(calculate(m1))) hide (m2,m3)"

// Textfenster
// ===========

<Textbox>
 TextBox = 10;10;-1;-1
  Wie verhalten sich die Radien der drei
  Kreise zueinander?
</Textbox>
```

Ellipse mit Tangente und Normale (Seite 143)

```
//
// Datei: Ellipse_mit_Tangente_und_Normale.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
//

// Systemvariablen
// ===============

MEASURE_EXACTNESS = 2

// Figurenbeschreibung
// ===================
e[1]  = a;       measure; controller;     7.5,0.5,0.0,10.0,150,"a = ","";
e[2]  = b;       measure; controller;     5.0,0.5,0.0,10.0,150,"b = ","";
e[3]  = a2;      measure; calculate;      "calculate(a)^2",-14.0,7.0,"a2 = ", "";
e[4]  = b2;      measure; calculate;      "calculate(b)^2",-14.0,6.0,"b2 = ", "";
e[5]  = ellipse; line;    curve;          "calculate(a)*cos(t)",
                 "calculate(b)*sin(t)", 0.0, 6.30, 50; 0;0;blue;0
e[6]  = P;       point;   curveSlider;    6.3,2.7,ellipse; black;red;blue;circle
e[7]  = m0;      measure; property;       P,"t",-14.0, 5.0, "t = ", "";
e[8]  = Tang;    point;   functionDepend; "coordinateX(P)-calculate(a)*sin(calculate(m0))",
                 "coordinateY(P)+calculate(b)*cos(calculate(m0))"; "hidden"
e[9]  = t;       line;    straightline;   Tang,P; 0;0;black;0
e[10] = N1;      point;   rotation;       Tang,P,-1.570796327,0.25; "hidden"
e[11] = N2;      point;   rotation;       Tang,P,-1.570796327,1.0; "hidden"
e[12] = n;       line;    straightLine;   P,N1;   0;0;black;0
e[13] = n';      line;    ray;            N1,N2;  "hidden"
e[14] = alpha;   measure; XVector;        N1,P,-14.0,4.0,"alpha = ", "";
e[15] = beta;    measure; YVector;        N1,P,-14.0,3.0,"beta = ", "";
e[16] = A;       measure; calculate;
                 "(calculate(alpha)^2/calculate(a2))+(calculate(beta)^2/calculate(b2))",
                 -14.0,2.0,"A = ", "";
e[17] = B;       measure; calculate;
                 "2*( ($(P,x)/calculate(a2))*calculate(alpha)+($(P,y)/calculate(b2))*
                 calculate(beta))",-14.0,1.0,"B = ", "";
e[18] = C;       measure; calculate;
                 "($(P,x)^2/calculate(a2))+($(P,y)^2/calculate(b2))-1",
                 -14.0,0.0,"C = ", "";
e[19] = D;       measure; calculate;      "sqrt((calculate(B)^2)-
                 (4*calculate(A)*calculate(C)))",-14.0,-1.0,"sqrt(D) = ", "";
e[20] = mP;      measure; coordinates;    P,-14.0,-4.0,"P = ", "";
e[21] = s1;      measure; calculate;
                 "(-(calculate(B))+calculate(D))/(2*calculate(A))",-14.0,-2.0,"s1 = ", "";
e[22] = s2;      measure; calculate;
                 "(-(calculate(B))-calculate(D))/(2*calculate(A))",-14.0,-3.0,"s2 = ", "";
e[23] = I1       point;   functionDepend; "$(P,x)+calculate(s1)*(calculate(alpha))",
                 "$(P,y)+calculate(s1)*(calculate(beta))";
e[24] = Q;       point;   functionDepend; "$(P,x)+calculate(s2)*(calculate(alpha))",
                 "$(P,y)+calculate(s2)*(calculate(beta))";
e[25] = sw1;     measure; switch;         10.0,-80.0,0.0,"Berechnung zeigen = ","";
e[26] = sw2;     measure; checkbox;       "Hilfe zeigen",0;
e[27] = d;       measure; distance;       P,Q,-14.0,5.0,"Abstand PQ = ","";

// Ein- und Ausblenden von Objekten
// ================================

hidden[1] = "if (not(calculate(sw1))) hide (a2,b2,m0,alpha,beta,A,B,C,D,mP,s1,s2)"
hidden[2] = "if (not(calculate(sw2))) hide (d)"

// Textfenster
// ===========

<Textbox>
 TextBox = 10;10;-1;-1
   Finde den kleinsten Wert
   für den Abstand zwischen
   P und Q.
</Textbox>
```

Doppelspiegelung eines Fünfecks (Seite 144)

```
//
// Datei: Doppelspiegelung_eines_Fuenfecks.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
//

// Figurenbeschreibung
// ===================

e[1]  = A;  point;    free;         -11.4,7.35;
e[2]  = B;  point;    free;         -13.65,5.95;
e[3]  = C;  point;    free;         -14.25,3.35;
e[4]  = D;  point;    free;         -12.05,2.8;
e[5]  = E;  point;    free;         -9.95,4.4;
e[6]  = O;  polygon;  pentagon;     A,B,C,D,E;         0;0;black;yellow
e[7]  = S1; point;    free;         -5.15,5.5;         "hideLabel"
e[8]  = S2; point;    free;         -12.95,1.0;        "hideLabel"
e[9]  = s1; line;     straightline; S1,S2;             0;0;black;0
e[10] = A';  point;   mirror;       A,s1;
e[11] = B';  point;   mirror;       B,s1;
e[12] = C';  point;   mirror;       C,s1;
e[13] = D';  point;   mirror;       D,s1;
e[14] = E';  point;   mirror;       E,s1;
e[15] = B1;  polygon; pentagon;     A',B',C',D',E';     0;0;blue;lightGray
e[16] = S3;  point;   free;         -8.95,-5.85;       "hideLabel"
e[17] = S4;  point;   free;         -3.05,3.15;        "hideLabel"
e[18] = s2;  line;    straightline; S3,S4;              0;0;black;0
e[19] = A''; point;   mirror;       A',s2;
e[20] = B''; point;   mirror;       B',s2;
e[21] = C''; point;   mirror;       C',s2;
e[22] = D''; point;   mirror;       D',s2;
e[23] = E''; point;   mirror;       E',s2;
e[24] = B2;  polygon; pentagon;     A'',B'',C'',D'',E''; 0;0;blue;yellow

// Textfenster
// ===========

<Textbox>
 TextBox = 10;10;-1;-1
 Die beiden Geraden sollen so verschoben werden,
 daß die Fünfecke ABCDE und A''B''C''D''E'' zur
 Deckung kommen.
</Textbox>
```

Aufgabe zur Geradenspiegelung (Seite 145)

```
//
// Datei: Aufgabe_zur_Geradenspiegelung.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
//

// Systemvariablen
// ===============

snapToGrid = true

// Figurenbeschreibung
// ===================

e[1]  = A;  point;   fixed;        -6.0,0.0;
e[2]  = B;  point;   fixed;        -3.0,6.0;
e[3]  = C;  point;   fixed;        -11.0, 6.0;
e[4]  = P1; polygon; triangle;     A,B,C;     0;0;black;yellow
e[5]  = S1; point;   fixed;        -2.0,2.0;  "hidden"
e[6]  = S2; point;   fixed;        -5.0,-1.0; "hidden"
e[7]  = g;  line;    straightline; S1,S2;     "hidden"
e[8]  = A'; point;   mirror;       A,g;       black;black;0;smallsquare
e[9]  = B'; point;   free;         6.0,-1.0;
e[10] = C'; point;   free;         4.0,-5.0;
e[11] = P2; polygon; triangle;     A',B',C';  0;0;black;lightGray
e[12] = G1; point;   free;         -8.0,-3.0; "hideLabel"
e[13] = G2; point;   free;         2.0,-6.0;  "hideLabel"
e[14] = s;  line;    straightline; G1,G2;
e[15] = MA; point;   midpoint;     A,A';      "hidden"
e[16] = MB; point;   midpoint;     B,B';      "hidden"
e[17] = MC; point;   midpoint;     C,C';      "hidden"
e[18] = ZB; point;   mirror;       B,g;       "hidden"
e[19] = ZC; point;   mirror;       C,g;       "hidden"
e[20] = m2; measure; button;       "Hilfe","help";
e[21] = m3; measure; button;       "Auswertung","evaluate";

// Textfenster
// ===========

<ProblemText>
 Position = 10;10;-1;-1
 Verändere die Lage der Geraden s, so
 daß A durch eine Spiegelung an s in A' übergeht.
 Die Punkte B' und C' sollen die Bildpunkte zu B
 und C bzgl. der Spiegelung an s werden.
</ProblemText>

// Hilfen
// ======

<Help>
     Die Gerade s muß durch den
     Mittelpunkt zwischen A und
     A' verlaufen.
</Help>

// Antwortanalyse
// ==============

<Problem>
 MAX_ANSWER = 0
 condition[1] = "isIncident(ZB,B')"
 condition[2] = "isIncident(ZC,C')"
 condition[3] = "isIncident(G1,g)"
 condition[4] = "isIncident(G2,g)"
</Problem>

<Answer 1>
 key = "condition[1] AND condition[2] AND condition[3] AND condition[4]"
 comment[1] = "Richtig. /n /nIn dieser Lage gehen die beiden Dreiecke /n
```

```
             durch eine Spiegelung an der Geraden s /nineinander über."
</Answer 1>

<Answer 2>
 key = "NOT(condition[1]) AND NOT(condition[2]) AND NOT(condition[3] AND condition[4])"
 comment[1] = "Ihre Antwort ist nicht richtig. /n
              Weder die Lage der Geraden noch die /n
              Koordinaten der Bildpunkte sind korrekt. /n
              Versuchen Sie es noch einmal."
</Answer 2>

<Answer 3>
 key = "NOT(condition[1]) AND NOT(condition[2]) AND condition[3] AND condition[4]"
 comment[1] = "Ihre Antwort ist nicht richtig. /n
              Die Lage der Geraden s stimmt, /n
              aber die Bildpunkte sind noch nicht korrekt./n
              Versuchen Sie es noch einmal."
</Answer 3>

<Answer 4>
 key = "(NOT(condition[1])) AND condition[2] AND condition[3] AND condition[4]"
 comment[1] = "Ihre Antwort ist teilweise richtig. /n
              Nur der Bildpunkt B' stimmt noch nicht. /n
              Versuchen Sie es noch einmal."
</Answer 4>

<Answer 5>
 key = "(NOT(condition[2])) AND condition[1] AND condition[3] AND condition[4]"
 comment[1] = "Ihre Antwort ist teilweise richtig. /n
              Nur der Bildpunkt C' stimmt noch nicht. /n
              Versuchen Sie es noch einmal."
</Answer 5>

<Answer 6>
 key = "1"
 comment[1] = "Ihre Antwort ist nicht richtig. /n
              Die Lage der Geraden s und der beiden /n
              Bildpunkte B' und C' stimmt noch nicht./n
              Versuchen Sie es noch einmal."
</Answer 6>
```

Drehstreckung eines Dreiecks (Seite 146)

```
//
// Datei: Drehstreckung_eines_Dreiecks.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
//

// Systemvariablen
// ===============

snapToGrid = true

// Figurenbeschreibung
// ===================

e[1]  = m0; measure; calculate; "1.5";
e[2]  = m1; measure; calculate; "-1.570796327";
e[3]  = A;  point;   fixed;     -1.0,  1.0;
e[4]  = B;  point;   fixed;     -2.0, -2.0;
e[5]  = C;  point;   fixed;     2.0, -2.0;
e[6]  = Z;  point;   fixed;     2.0,  2.0;
e[7]  = P1; polygon; triangle;  A,B,C;      0;0;black;yellow
e[8]  = A'; point;   free;      -8.0,  1.0;
e[9]  = B'; point;   free;      -6.0, -2.0;
e[10] = C'; point;   free;      -4.0,  2.0;
e[11] = P2; polygon; triangle;  A',B',C';   0;0;black;green
e[12] = A1; point;   rotation;  A,Z,-1.570796327,1.0; "hidden"
e[13] = B1; point;   rotation;  B,Z,-1.570796327,1.0; "hidden"
e[14] = C1; point;   rotation;  C,Z,-1.570796327,1.0; "hidden"
e[15] = A2; point;   rotation;  A,Z,m1,m0;  "hidden"
e[16] = B2; point;   rotation;  B,Z,m1,m0;  "hidden"
e[17] = C2; point;   rotation;  C,Z,m1,m0;  "hidden"
e[18] = m2; measure; button;    "Hilfe","help";
e[19] = m3; measure; button;    "Auswertung","evaluate";

// Textfenster
// ===========

<ProblemText>
 Position = 140;10;-1;-1
  Das Dreieck ABC soll um das Drehzentrum Z um 90°
  gedreht werden.
  Außerdem findet eine Streckung um den Faktor 1.5
  statt. Man bewege das Dreieck A'B'C' an die
  entsprechende Position.
</ProblemText>

// Hilfen
// ======

<Help>
     Führe zuerst mit dem Dreieck A'B'C'
     die Drehung durch und dann die
     Streckung.
</Help>

// Antwortanaylse
// ==============

<Problem>
 MAX_ANSWER = 0
 condition[1] = "isIncident(A2,A')"
 condition[2] = "isIncident(B2,B')"
 condition[3] = "isIncident(C2,C')"
 condition[4] = "isIncident(A1,A')"
 condition[5] = "isIncident(B1,B')"
 condition[6] = "isIncident(C1,C')"
</Problem>

<Answer 1>
 key = "condition[1] AND condition[2] AND condition[3]"
```

```
 comment[1] = "Richtig. /n /nDie Drehstreckung wurde korrekt durchgeführt."
</Answer 1>

<Answer 2>
 key = "condition[4] AND condition[5] AND condition[6]"
 comment[1] = "Ihre Antwort ist nicht richtig. /n
               Die Drehung wurde zwar korrekt durchgeführt, /n
               es fehlt aber noch die Streckung./n
               Versuchen Sie es noch einmal."
</Answer 2>

<Answer 3>
 key = "not(condition[1]) AND condition[2] AND condition[3]"
 comment[1] = "Ihre Antwort ist teilweise richtig. /n
               Die Bildpunkte B' und C' stimmen, /n
               aber A' ist noch nicht korrekt plaziert. /n
               Versuchen Sie es noch einmal."
</Answer 3>

<Answer 4>
 key = "not(condition[2]) AND condition[1] AND condition[3]"
 comment[1] = "Ihre Antwort ist teilweise richtig. /n
               Die Bildpunkte A' und C' stimmen, /n
               aber B' ist noch nicht korrekt plaziert. /n
               Versuchen Sie es noch einmal."
</Answer 4>

<Answer 5>
 key = "not(condition[3]) AND condition[2] AND condition[1]"
 comment[1] = "Ihre Antwort ist teilweise richtig. /n
               Die Bildpunkte A' und B' stimmen, /n
               aber C' ist noch nicht korrekt plaziert. /n
               Versuchen Sie es noch einmal."
</Answer 5>

<Answer 6>
 key = "not(condition[1]) AND NOT(condition[2]) AND condition[3]"
 comment[1] = "Ihre Antwort ist teilweise richtig. /n
               Die Bildpunkte A' und B' stimmen nicht, /n
               nur C' ist korrekt plaziert. /n
               Versuchen Sie es noch einmal."
</Answer 6>

<Answer 7>
 key = "not(condition[3]) AND NOT(condition[2]) AND condition[1]"
 comment[1] = "Ihre Antwort ist teilweise richtig. /n
               Die Bildpunkte B' und C' stimmen nicht, /n
               nur A' ist korrekt plaziert. /n
               Versuchen Sie es noch einmal."
</Answer 7>

<Answer 8>
 key = "not(condition[1]) AND NOT(condition[3]) AND condition[2]"
 comment[1] = "Ihre Antwort ist teilweise richtig. /n
               Die Bildpunkte A' und C' stimmen nicht, /n
               nur B' ist korrekt plaziert. /n
               Versuchen Sie es noch einmal."
</Answer 8>

<Answer 9>
 key = "1"
 comment[1] = "Ihre Antwort ist nicht richtig. /n
               Die Bildpunkte A', B' und C' sind nicht korrekt plaziert. /n
               Versuchen Sie es noch einmal."
</Answer 9>
```

Nicht-affine Abbildung (Seite 147)

```
//
// Datei: Nicht-affine_Abbildung.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
//

// Systemvariablen
// ===============

snapToGrid = true

// Figurenbeschreibung
// ===================

e[1]  = O;     point;   fixed;          0.0,0.0;  "hidden"
e[2]  = Ox;    point;   fixed;          1.0,0.0;  "hidden"
e[3]  = Oy;    point;   fixed;          0.0,1.0;  "hidden"
e[4]  = coord; point;   coordSystem;    O,O,Ox,O,Oy,300,300,220,220; 0;red;black;0
e[5]  = A;     point;   free;           -7.0,-1.0;
e[6]  = B;     point;   free;           -4.0,2.0;
e[7]  = C;     point;   free;           1.0,1.0;
e[8]  = P1;    polygon; triangle;       A,B,C;     0;0;black;yellow
e[9]  = A';    point;   functionDepend; "abs(coordinateX(A))","3*abs(coordinateY(A))";
e[10] = B';    point;   functionDepend; "abs(coordinateX(B))","3*abs(coordinateY(B))";
e[11] = C';    point;   functionDepend; "abs(coordinateX(C))","3*abs(coordinateY(C))";
e[12] = P2;    polygon; triangle;       A',B',C';  0;0;black;lightGray

// Textfenster
// ===========

<Textbox>
 Position = 80;30;-1;-1
 Das Dreieck ABC wird durch eine
 Funktion f auf das Dreieck A'B'C'
 abgebildet.
 Wie lautet die Abbildungsvorschrift?
</Textbox>
```

Generieren eines Funktionsgraphen (Seite 148)

```
//
// Datei: Generieren_eines_Funktionsgraphen.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
//

// Systemvariablen
// ===============

WORLD_X_MAX       = +8.0
WORLD_X_MIN       = -8.0
WORLD_Y_MAX       = +6.0
WORLD_Y_MIN       = -6.0
MEASURE_EXACTNESS = 2

// Figurenbeschreibung
// ===================

e[1]  = O;     point;   fixed;          0.0,0.0;  "hidden"
e[2]  = Ox;    point;   fixed;          1.0,0.0;  "hidden"
e[3]  = Oy;    point;   fixed;          0.0,1.0;  "hidden"
e[4]  = coord; point;   coordSystem;    O,O,Ox,O,Oy,300,300,220,220; 0;red;black;0
e[5]  = a;     measure; controller;     1.0,0.25,-5.0,5.0,150,"a = ","";
e[6]  = b;     measure; controller;     -1.0,0.25,-5.0,5.0,150,"b = ","";
e[7]  = c;     measure; controller;     1.0,0.25,-5.0,5.0,150,"c = ","";
e[8]  = x;     point;   horizontal;     O,1.0;
e[9]  = f(x);  point;   functionDepend; "coordinateX(x)",
               "calculate(a)*coordinateX(x)+calculate(b)/coordinateX(x)+calculate(c)";
e[10] = m0;    measure; coordinateX;    x,-12.0,6.0,"x = ","";
e[11] = m1;    measure; coordinateY;    f(x),-12.0,5.0,"f(x) = ","";
e[12] = Y;     point;   functionDepend; "0","calculate(a)*coordinateX(x)+
               calculate(b)/coordinateX(x)+calculate(c)"; "hidden"
e[13] = y;     line;    connect;        f(x),x;   0;0;lightGray;0
e[14] = z;     line;    connect;        f(x),Y;   0;0;lightGray;0

// Textfenster
// ===========

<ProblemText>
 Position = 40;20;-1;-1
  Erzeuge den Graphen der Funktion

  f(x) = ax + b/x + c
       = {"calculate(a)"} x + ({"calculate(b)"}) / x + ({"calculate(c)"}) .
</ProblemText>
```

Untersuchen besonderer Punkte (Seite 149)

```
//
// Datei: Untersuchen_besonderer_Punkte.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
//

// Systemvariablen
// ===============

WORLD_X_MAX       = +8.0
WORLD_X_MIN       = -8.0
WORLD_Y_MAX       = +6.0
WORLD_Y_MIN       = -6.0
MEASURE_EXACTNESS = 2

// Figurenbeschreibung
// ===================

e[1]  = O;     point;   fixed;          0.0,0.0;  "hidden"
e[2]  = Ox;    point;   fixed;          1.0,0.0;  "hidden"
e[3]  = Oy;    point;   fixed;          0.0,1.0;  "hidden"
e[4]  = coord; point;   coordSystem;    O,O,Ox,O,Oy,300,300,220,220; 0;red;black;0
e[5]  = a;     measure; controller;     -0.5,0.25,-5.0,5.0,150,"a = ","";
e[6]  = b;     measure; controller;     -1.5,0.25,-5.0,5.0,150,"b = ","";
e[7]  = c;     measure; controller;     -0.5,0.25,-5.0,5.0,150,"c = ","";
e[8]  = fkt;   line;    curve;          "t","calculate(a)*t+calculate(b)/t+calculate(c)",
               -8.0, 8.0, 70;
e[9]  = P;     point;   curveSlider;    1.0,1.0,fkt;
e[10] = m0;    measure; coordinateX;    P,-12.0,6.0,"x = ","";
e[11] = m1;    measure; coordinateY;    P,-12.0,5.0,"f(x) = ","";
e[12] = X;     point;   functionDepend; "coordinateX(P)","0"; "hidden"
e[13] = Y;     point;   functionDepend; "0","coordinateY(P)"; "hidden"
e[14] = y;     line;    connect;        Y,P;   0;0;lightGray;0
e[15] = x;     line;    connect;        X,P;   0;0;lightGray;0

// Textfenster
// ===========

<ProblemText>
 Position = 40;20;-1;-1
  Untersuche die speziellen Punkte
  des Funktionsgraphen
  f(x) = ax + b/x + c
       = {"calculate(a)"} x + ({"calculate(b)"}) / x + ({"calculate(c)"}) .
</ProblemText>
```

Untersuchen der Funktionsgleichung (Seite 150)

```
//
// Datei: Untersuchen_der_Funktionsgleichung.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
//

// Systemvariablen
// ===============

WORLD_X_MAX       = +8.0
WORLD_X_MIN       = -8.0
WORLD_Y_MAX       = +6.0
WORLD_Y_MIN       = -6.0
MEASURE_EXACTNESS = 1

// Figurenbeschreibung
// ===================

e[1]  = O;     point;   fixed;       0.0,0.0;                            "hidden"
e[2]  = Ox;    point;   fixed;       1.0,0.0;                            "hidden"
e[3]  = Oy;    point;   fixed;       0.0,1.0;                            "hidden"
e[4]  = fkt0;  line;    curve;       "t","-1.5/t-1", -8.0, 8.0, 50;      0;0;gray;0
e[5]  = fkt1;  line;    curve;       "t","-1.5*(-2/t)+1", -8.0, 8.0, 50; 0;0;gray;0
e[6]  = fkt2;  line;    curve;       "t","1.5*t-2", -8.0, 8.0, 50;       0;0;gray;0
e[7]  = coord; point;   coordSystem; O,O,Ox,O,Oy,300,300,220,220;        0;red;black;0
e[8]  = P1;    point;   free;        2.0,2.0;
e[9]  = P2;    point;   free;        3.0,2.0;
e[10] = P3;    point;   free;        1.0,-2.0;
e[11] = curve; line;    curve;       "Curve_Funktionsgraph","P1","P2","P3",
               "-8.0","8.0","150"; black;red;blue;0
e[12] = m0;    measure; function;    "Functional_Funktionsparameter",
               "P1","P2","P3","a",6.0,5.0,"a = ","";
e[13] = m1;    measure; function;    "Functional_Funktionsparameter",
               "P1","P2","P3","b",6.0,4.0,"b = ","";
e[14] = m2;    measure; function;    "Functional_Funktionsparameter",
               "P1","P2","P3","c",6.0,3.0,"c = ","";

// Textfenster
// ===========

<ProblemText>
 Position = 40;20;-1;-1
  Variiere den Graphen der Funktion
  f(x) = ax + b/x + c durch Verschieben
  der Punkte P1, P2 und P3.
  Welche Parameterwerte haben die
  drei festen Funktionsgraphen?
</ProblemText>
```

Untersuchen des Graphen (Seite 151)

```
//
// Datei: Untersuchen_des_Graphen.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
//

// Systemvariablen
// ===============

WORLD_X_MAX       = +8.0
WORLD_X_MIN       = -8.0
WORLD_Y_MAX       = +6.0
WORLD_Y_MIN       = -6.0
MEASURE_EXACTNESS = 2

// Figurenbeschreibung
// ===================

e[1]  = O;     point;   fixed;       0.0,0.0;                          "hidden"
e[2]  = Ox;    point;   fixed;       1.0,0.0;                          "hidden"
e[3]  = Oy;    point;   fixed;       0.0,1.0;                          "hidden"
e[4]  = fkt0;  line;    curve;       "t","-1.5/t+2", -8.0, 8.0, 50;    0;0;gray;0
e[5]  = fkt1;  line;    curve;       "t","1.5*(2/t)+1", -8.0, 8.0, 50; 0;0;gray;0
e[6]  = fkt2;  line;    curve;       "t","0.5*t-2", -8.0, 8.0, 50;     0;0;gray;0
e[7]  = coord; point;   coordSystem; O,O,Ox,O,Oy,300,300,220,220;      0;red;black;0
e[8]  = a;     measure; controller;  1.0,0.25,-5.0,5.0,150,"a = ","";
e[9]  = b;     measure; controller;  -1.0,0.25,-5.0,5.0,150,"b = ","";
e[10] = c;     measure; controller;  1.0,0.25,-5.0,5.0,150,"c = ","";
e[11] = fkt;   line;    curve;       "t","calculate(a)*t+calculate(b)/t+calculate(c)",
               -8.0,8.0,70;

// Textfenster
// ===========

<ProblemText>
 Position = 40;20;-1;-1
  Variiere den Graphen der Funktion
  f(x) = ax + b/x + c durch Verändern
  der Parameter a, b und c.
  Welche Parameterwerte haben die
  drei festen Funktionsgraphen?
</ProblemText>
```

Modulares Generieren des Funktionsgraphen (Seite 152)

```
//
// Datei: Modulares_Generieren_des_Funktionsgraphen.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
//

// Systemvariablen
// ===============

WORLD_X_MAX       = +8.0
WORLD_X_MIN       = -8.0
WORLD_Y_MAX       = +6.0
WORLD_Y_MIN       = -6.0
MEASURE_EXACTNESS = 2

// Figurenbeschreibung
// ===================

e[1]  = O;        point;   fixed;          0.0,0.0;                    "hidden"
e[2]  = Ox;       point;   fixed;          1.0,0.0;                    "hidden"
e[3]  = Oy;       point;   fixed;          0.0,1.0;                    "hidden"
e[4]  = coord;    point;   coordSystem;    O,O,Ox,O,Oy,300,300,220,220; 0;red;black;0
e[5]  = a;        measure; controller;     1.0,0.25,-5.0,5.0,130,"a = ","";
e[6]  = b;        measure; controller;     -1.5,0.25,-5.0,5.0,130,"b = ","";
e[7]  = c;        measure; controller;     1.0,0.25,-5.0,5.0,130,"c = ","";
e[8]  = fkt0;     line;    curve;          "t","calculate(a)*t", -8.0, 8.0, 10;
                  0;0;lightGray;0
e[9]  = fkt1;     line;    curve;          "t",
                  "calculate(b)/t+calculate(c)", -8.0, 8.0, 70;         0;0;gray;0
e[10] = fkt;      line;    curve;          "t",
                  "calculate(a)*t+calculate(b)/t+calculate(c)", -8.0, 8.0, 70;
e[11] = P;        point;   horizontal;     O,-1.0;
e[12] = ax;       point;   functionDepend; "coordinateX(P)",
                  "calculate(a)*coordinateX(P)";
e[13] = b/x+c;    point;   functionDepend; "coordinateX(P)",
                  "calculate(b)/coordinateX(P)+calculate(c)";
e[14] = ax+b/x+c; point;   functionDepend; "coordinateX(P)",
                  "calculate(a)*coordinateX(P)+calculate(b)/coordinateX(P)+calculate(c)";
e[15] = m0;       measure; coordinateX;    P,5.0,2.0,"x = ","";
e[16] = m1;       measure; coordinateY;    ax,5.0,1.5,"ax = ","";
e[17] = m2;       measure; coordinateY;    b/x+c,5.0,1.0,"b/x+c = ","";
e[18] = m3;       measure; coordinateY;    ax+b/x+c,5.0,0.5,"f(x) = ","";
e[19] = s1;       line;    connect;        ax,P;   "hideLabel"
e[20] = s2;       line;    connect;        ax,b/x+c;                0;0;lightGray;0
e[21] = s3;       line;    connect;        ax+b/x+c,b/x+c;          "hideLabel"
e[22] = sw1;      measure; checkbox;       "f(x)=ax+b/x+c zeigen",0;

// Ein- und Ausblenden von Objekten
// ================================

hidden[1] = "if (not(calculate(sw1))) hide (fkt)"

// Textfenster
// ===========

<ProblemText>
 Position = 40;20;-1;-1
  Verschiebe den Punkt P entlang der
  x-Achse und betrachte die Summe
  der beiden Funktionen
  f1(x) = a x und
  f2(x) = b / x + c .
</ProblemText>
```

Generieren der Umkehrfunktion (Seite 153)

```
//
// Datei: Generieren_der_Umkehrfunktion.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
//

// Systemvariablen
// ===============

WORLD_X_MAX       = +8.0
WORLD_X_MIN       = -8.0
WORLD_Y_MAX       = +6.0
WORLD_Y_MIN       = -6.0
MEASURE_EXACTNESS = 2

// Figurenbeschreibung
// ===================

e[1]  = O;     point;   fixed;          0.0,0.0;                   "hidden"
e[2]  = Ox;    point;   fixed;          1.0,0.0;                   "hidden"
e[3]  = Oy;    point;   fixed;          0.0,1.0;                   "hidden"
e[4]  = A;     point;   fixed;          1.0,1.0;                   "hidden"
e[5]  = coord; point;   coordSystem;    O,O,Ox,O,Oy,300,300,220,220; 0;red;black;0
e[6]  = w;     line;    straightLine;   O,A;   0;0;black;0
e[7]  = a;     measure; controller;     0.5,0.25,-5.0,5.0,130,"a = ","";
e[8]  = b;     measure; controller;     -2.5,0.25,-5.0,5.0,130,"b = ","";
e[9]  = c;     measure; controller;     -0.5,0.25,-5.0,5.0,130,"c = ","";
e[10] = fkt;   line;    curve;          "t","calculate(a)*t+calculate(b)/t+calculate(c)",
               -8.0, 8.0, 70;
e[11] = P;     point;   curveSlider;    1.0,1.0,fkt;
e[12] = X;     point;   functionDepend; "coordinateX(P)","0";       "hidden"
e[13] = Y;     point;   functionDepend; "0","coordinateY(P)";       "hidden"
e[14] = y;     line;    connect; Y,P;   0;0;lightGray;0
e[15] = x;     line;    connect; X,P;   0;0;lightGray;0
e[16] = P';    point;   functionDepend; "coordinateY(P)","coordinateX(P)";
e[17] = X';    point;   functionDepend; "coordinateX(P')","0";      "hidden"
e[18] = Y';    point;   functionDepend; "0","coordinateY(P')";      "hidden"
e[19] = y';    line;    connect;        Y',P';                     0;0;lightGray;0
e[20] = x';    line;    connect;        X',P';                     0;0;lightGray;0
e[21] = m4;    measure; coordinates;    P,5.0,1.0,"P = ","";
e[22] = m5;    measure; coordinates;    P',5.0,1.5,"P' = ","";
e[23] = m6;    measure; button;         "Ortsspuren löschen","cleartrace";

// Textfenster
// ===========

<ProblemText>
 Position = 40;20;220;140
  Der Punkt P' ist der Bildpunkt der
  Spiegelung von P an der 1. Winkel-
  halbierenden.
  Erzeuge die Umkehrfunktion von
  f(x) = {"calculate(a)"} x + ({"calculate(b)"}) / x + ({"calculate(c)"}) ,
  indem Du die Ortsspur von P'
  aufzeichnest.
</ProblemText>
```

Untersuchen von Tangenten (Seite 153)

```
//
// Datei: Untersuchen_von_Tangenten.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
//

// Systemvariablen
// ===============

WORLD_X_MAX       = +8.0
WORLD_X_MIN       = -8.0
WORLD_Y_MAX       = +6.0
WORLD_Y_MIN       = -6.0
MEASURE_EXACTNESS = 2

// Figurenbeschreibung
// ===================

e[1]  = O;     point;   fixed;          0.0,0.0;                    "hidden"
e[2]  = Ox;    point;   fixed;          1.0,0.0;                    "hidden"
e[3]  = Oy;    point;   fixed;          0.0,1.0;                    "hidden"
e[4]  = coord; point;   coordSystem;    O,O,Ox,O,Oy,300,300,220,220; 0;red;black;0
e[5]  = a;     measure; controller;     -0.5,0.25,-5.0,5.0,150,"a = ","";
e[6]  = b;     measure; controller;     -1.5,0.25,-5.0,5.0,150,"b = ","";
e[7]  = c;     measure; controller;     -1.0,0.25,-5.0,5.0,150,"c = ","";
e[8]  = fkt;   line;    curve;          "t",
               "calculate(a)*t+calculate(b)/t+calculate(c)", -8.0, 8.0, 70;
e[9]  = P;     point;   curveSlider;    -1.73,0.75,fkt;
e[10] = X;     point;   functionDepend; "coordinateX(P)","0";        "hidden"
e[11] = Y;     point;   functionDepend; "0","coordinateY(P)";        "hidden"
e[12] = y;     line;    connect; Y,P;                                0;0;lightGray;0
e[13] = x;     line;    connect; X,P;                                0;0;lightGray;0
e[14] = m4;    measure; coordinates;    P,5.0,1.5,"P = ","";
e[15] = T;     point;   functionDepend; "coordinateX(P)+1",
               "coordinateY(P)+(calculate(a))-(calculate(b)/
               (coordinateX(P)*coordinateX(P)))"; "hidden"
e[16] = t;     line;    straightLine;   P,T;   0;0;black;0
e[17] = m7;    measure; calculate;
               "(calculate(a))-(calculate(b)/(coordinateX(P)*coordinateX(P)))",
               5.0,1.0,"f'(x) = ","";

// Textfenster
// ===========

<ProblemText>
 Position = 40;20;220;70
  Experimentiere mit der Tangente
  durch den Punkt P.
  Gibt es besondere Lagen?
</ProblemText>
```

Algebraische Kurve (Seite 155)

```
//
// Datei: Algebraische_Kurve.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
// Kurvendiskussion aus
// Schupp & Dabrock: Höhere Kurven, Mannheim 1995, S. 362-366
//

// Systemvariablen
// ===============

WORLD_X_MAX       = +4.0
WORLD_X_MIN       = -4.0
WORLD_Y_MAX       = +3.0
WORLD_Y_MIN       = -3.0
MEASURE_EXACTNESS = 2

// Figurenbeschreibung
// ===================

e[1] = O;     point;   fixed;       0.0,0.0;                              "hidden"
e[2] = Ox;    point;   fixed;       1.0,0.0;                              "hidden"
e[3] = Oy;    point;   fixed;       0.0,1.0;                              "hidden"
e[4] = coord; point;   coordSystem; O,O,Ox,O,Oy,300,300,220,220;          0;red;black;0
e[5] = a;     measure; controller;  -2.0,0.25,-5.0,5.0,150,"a = ","";
e[6] = b;     measure; controller;  5.0,0.25,-5.0,5.0,150,"b = ","";
e[7] = curve; line;    curve;       "Curve_Demo3","a","b","330","-4.0","4.0";
              black;red;blue;0
e[8] = m0;    measure; calculate;   "if (calculate(a) < 0) then (0.0) else (1.0)";
e[9] = m1;    measure; calculate;   "if (calculate(b) < 0) then (0.0) else (1.0)";

// Beschränken des Zustandsraums der Figur
// =======================================

limit[1] = "calculate(m0)|calculate(m1)"
```

Exponentialfunktion (Seite 156)

```
//
// Datei: Exponentialfunktion.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
//

// Systemvariablen
// ===============

WORLD_X_MAX       = +8.0
WORLD_X_MIN       = -8.0
WORLD_Y_MAX       = +6.0
WORLD_Y_MIN       = -6.0
MEASURE_EXACTNESS = 2

// Figurenbeschreibung
// ===================

e[1]  = O;     point;   fixed;          0.0,0.0;  "hidden"
e[2]  = Ox;    point;   fixed;          1.0,0.0;  "hidden"
e[3]  = Oy;    point;   fixed;          0.0,1.0;  "hidden"
e[4]  = a;     measure; controller;     1.0,0.25,-5.0,5.0,150,"a = ","";
e[5]  = b;     measure; controller;     -1.0,0.25,-5.0,5.0,150,"b = ","";
e[6]  = P1;    point;   horizontal;     -1.0,0;
e[7]  = P2;    point;   horizontal;     1.0,0;
e[8]  = P1';   point;   functionDepend; "coordinateX(P1)",
               "exp(calculate(a)*coordinateX(P1)+calculate(b))"; "hidden"
e[9]  = P2';   point;   functionDepend; "coordinateX(P2)",
               "exp(calculate(a)*coordinateX(P2)+calculate(b))";  "hidden"
e[10] = F1;    point;   functionDepend; "coordinateX(P1)+1*XVector(P1,P2)/20",
               "0.0";       "hidden"
e[11] = F1';   point;   functionDepend; "coordinateX(F1)",
               "exp(calculate(a)*coordinateX(F1)+calculate(b))"; "hidden"
e[12] = F2;    point;   functionDepend; "coordinateX(P1)+2*XVector(P1,P2)/20",
               "0.0"; "hidden"
e[13] = F2';   point;   functionDepend; "coordinateX(F2)",
               "exp(calculate(a)*coordinateX(F2)+calculate(b))"; "hidden"
e[14] = F3;    point;   functionDepend; "coordinateX(P1)+3*XVector(P1,P2)/20",
               "0.0"; "hidden"
e[15] = F3';   point;   functionDepend; "coordinateX(F3)",
               "exp(calculate(a)*coordinateX(F3)+calculate(b))"; "hidden"
e[16] = F4;    point;   functionDepend; "coordinateX(P1)+4*XVector(P1,P2)/20",
               "0.0"; "hidden"
e[17] = F4';   point;   functionDepend; "coordinateX(F4)",
               "exp(calculate(a)*coordinateX(F4)+calculate(b))"; "hidden"
e[18] = F5;    point;   functionDepend; "coordinateX(P1)+5*XVector(P1,P2)/20",
               "0.0"; "hidden"
e[19] = F5';   point;   functionDepend; "coordinateX(F5)",
               "exp(calculate(a)*coordinateX(F5)+calculate(b))"; "hidden"
e[20] = F6;    point;   functionDepend; "coordinateX(P1)+6*XVector(P1,P2)/20",
               "0.0"; "hidden"
e[21] = F6';   point;   functionDepend; "coordinateX(F6)",
               "exp(calculate(a)*coordinateX(F6)+calculate(b))"; "hidden"
e[22] = F7;    point;   functionDepend; "coordinateX(P1)+7*XVector(P1,P2)/20",
               "0.0"; "hidden"
e[23] = F7';   point;   functionDepend; "coordinateX(F7)",
               "exp(calculate(a)*coordinateX(F7)+calculate(b))"; "hidden"
e[24] = F8;    point;   functionDepend; "coordinateX(P1)+8*XVector(P1,P2)/20",
               "0.0"; "hidden"
e[25] = F8';   point;   functionDepend; "coordinateX(F8)",
               "exp(calculate(a)*coordinateX(F8)+calculate(b))"; "hidden"
e[26] = F9;    point;   functionDepend; "coordinateX(P1)+9*XVector(P1,P2)/20",
               "0.0"; "hidden"
e[27] = F9';   point;   functionDepend; "coordinateX(F9)",
               "exp(calculate(a)*coordinateX(F9)+calculate(b))"; "hidden"
e[28] = F10;   point;   functionDepend; "coordinateX(P1)+10*XVector(P1,P2)/20",
               "0.0"; "hidden"
e[29] = F10';  point;   functionDepend; "coordinateX(F10)",
               "exp(calculate(a)*coordinateX(F10)+calculate(b))"; "hidden"
e[30] = F11;   point;   functionDepend; "coordinateX(P1)+11*XVector(P1,P2)/20",
```

```
                "0.0"; "hidden"
e[31] = F11';   point;   functionDepend; "coordinateX(F11)",
                "exp(calculate(a)*coordinateX(F11)+calculate(b))"; "hidden"
e[32] = F12;    point;   functionDepend; "coordinateX(P1)+12*XVector(P1,P2)/20",
                "0.0"; "hidden"
e[33] = F12';   point;   functionDepend; "coordinateX(F12)",
                "exp(calculate(a)*coordinateX(F12)+calculate(b))"; "hidden"
e[34] = F13;    point;   functionDepend; "coordinateX(P1)+13*XVector(P1,P2)/20",
                "0.0"; "hidden"
e[35] = F13';   point;   functionDepend; "coordinateX(F13)",
                "exp(calculate(a)*coordinateX(F13)+calculate(b))"; "hidden"
e[36] = F14;    point;   functionDepend; "coordinateX(P1)+14*XVector(P1,P2)/20",
                "0.0"; "hidden"
e[37] = F14';   point;   functionDepend; "coordinateX(F14)",
                "exp(calculate(a)*coordinateX(F14)+calculate(b))"; "hidden"
e[38] = F15;    point;   functionDepend; "coordinateX(P1)+15*XVector(P1,P2)/20",
                "0.0"; "hidden"
e[39] = F15';   point;   functionDepend; "coordinateX(F15)",
                "exp(calculate(a)*coordinateX(F15)+calculate(b))"; "hidden"
e[40] = F16;    point;   functionDepend; "coordinateX(P1)+16*XVector(P1,P2)/20",
                "0.0"; "hidden"
e[41] = F16';   point;   functionDepend; "coordinateX(F16)",
                "exp(calculate(a)*coordinateX(F16)+calculate(b))"; "hidden"
e[42] = F17;    point;   functionDepend; "coordinateX(P1)+17*XVector(P1,P2)/20",
                "0.0"; "hidden"
e[43] = F17';   point;   functionDepend; "coordinateX(F17)",
                "exp(calculate(a)*coordinateX(F17)+calculate(b))"; "hidden"
e[44] = F18;    point;   functionDepend; "coordinateX(P1)+18*XVector(P1,P2)/20",
                "0.0"; "hidden"
e[45] = F18';   point;   functionDepend; "coordinateX(F18)",
                "exp(calculate(a)*coordinateX(F18)+calculate(b))"; "hidden"
e[46] = F19;    point;   functionDepend; "coordinateX(P1)+19*XVector(P1,P2)/20",
                "0.0"; "hidden"
e[47] = F19';   point;   functionDepend; "coordinateX(F19)",
                "exp(calculate(a)*coordinateX(F19)+calculate(b))"; "hidden"
e[48] = p1;     polygon; quadrilateral;  P1,F1,F1',P1';     0;0;0;lightGray
e[49] = p2;     polygon; quadrilateral;  F1,F2,F2',F1';     0;0;0;lightGray
e[50] = p3;     polygon; quadrilateral;  F2,F3,F3',F2';     0;0;0;lightGray
e[51] = p4;     polygon; quadrilateral;  F3,F4,F4',F3';     0;0;0;lightGray
e[52] = p5;     polygon; quadrilateral;  F4,F5,F5',F4';     0;0;0;lightGray
e[53] = p6;     polygon; quadrilateral;  F5,F6,F6',F5';     0;0;0;lightGray
e[54] = p7;     polygon; quadrilateral;  F6,F7,F7',F6';     0;0;0;lightGray
e[55] = p8;     polygon; quadrilateral;  F7,F8,F8',F7';     0;0;0;lightGray
e[56] = p9;     polygon; quadrilateral;  F8,F9,F9',F8';     0;0;0;lightGray
e[57] = p10;    polygon; quadrilateral;  F9,F10,F10',F9';   0;0;0;lightGray
e[58] = p11;    polygon; quadrilateral;  F10,F11,F11',F10'; 0;0;0;lightGray
e[59] = p12;    polygon; quadrilateral;  F11,F12,F12',F11'; 0;0;0;lightGray
e[60] = p13;    polygon; quadrilateral;  F12,F13,F13',F12'; 0;0;0;lightGray
e[61] = p14;    polygon; quadrilateral;  F13,F14,F14',F13'; 0;0;0;lightGray
e[62] = p15;    polygon; quadrilateral;  F14,F15,F15',F14'; 0;0;0;lightGray
e[63] = p16;    polygon; quadrilateral;  F15,F16,F16',F15'; 0;0;0;lightGray
e[64] = p17;    polygon; quadrilateral;  F16,F17,F17',F16'; 0;0;0;lightGray
e[65] = p18;    polygon; quadrilateral;  F17,F18,F18',F17'; 0;0;0;lightGray
e[66] = p19;    polygon; quadrilateral;  F18,F19,F19',F18'; 0;0;0;lightGray
e[67] = p20;    polygon; quadrilateral;  F19,P2,P2',F19';   0;0;0;lightGray
e[68] = coord;  point;   coordSystem;    0,0,0x,0,0y,300,300,220,220; 0;red;black;0
e[69] = x0;     measure; coordinateX;    P1, -600.0, 6.0,"","";
e[70] = x1;     measure; coordinateX;    P2, -600.0, 6.0,"","";;
e[71] = s1;     line;    connect;        P1,P1'; 0;0;black;0
e[72] = s2;     line;    connect;        P2,P2'; 0;0;black;0
e[73] = e_fkt;  line;    curve;          "t","exp(calculate(a)*t+calculate(b))",
                -8.0, 8.0, 50;
e[74] = m0;     measure; function;       "Functional_Integral","e_fkt",
                "x0","x1","50","simpson",4.0,3.0,"Flächeninhalt = "," F.E.";

// Bild-Datei einbinden
// ====================

image[1] = "Aufgabe_Exponentialfunktion.gif", 20,40
```

Flächeninhalt unter einer Kurve (Seite 157)

```
//
// Datei:    Flaecheninhalt_unter_einer_Kurve.script
// Autor:    Timo Ehmke (ehmke@uni-flensburg.de)
//

// Systemvariablen
// ===============

WORLD_X_MAX       = +8.0
WORLD_X_MIN       = -8.0
WORLD_Y_MAX       = +6.0
WORLD_Y_MIN       = -6.0
MEASURE_EXACTNESS = 2

// Figurenbeschreibung
// ===================

e[1]  = O;     point;   fixed;          0.0,0.0;                     "hidden"
e[2]  = Ox;    point;   fixed;          1.0,0.0;                     "hidden"
e[3]  = Oy;    point;   fixed;          0.0,1.0;                     "hidden"
e[4]  = a;     measure; controller;     1.0,0.25,-5.0,5.0,150,"a = ","";
e[5]  = b;     measure; controller;     -3.0,0.25,-5.0,5.0,150,"b = ","";
e[6]  = P1;    point;   horizontal;     -1.56,0;
e[7]  = P2;    point;   horizontal;     2.56,0;
e[8]  = P1';   point;   functionDepend; "coordinateX(P1)",
               "abs(calculate(a)-coordinateX(P1)^2)-coordinateX(P1)+calculate(b)";
               "hidden"
e[9]  = P2';   point;   functionDepend; "coordinateX(P2)",
               "abs(calculate(a)-coordinateX(P2)^2)-coordinateX(P2)+calculate(b)";
               "hidden"
e[10] = F1;    point;   functionDepend; "coordinateX(P1)+1*XVector(P1,P2)/20",
               "0.0";  "hidden"
e[11] = F1';   point;   functionDepend; "coordinateX(F1)",
               "abs(calculate(a)-coordinateX(F1)^2)-coordinateX(F1)+calculate(b)";
               "hidden"
e[12] = F2;    point;   functionDepend; "coordinateX(P1)+2*XVector(P1,P2)/20",
               "0.0";  "hidden"
e[13] = F2';   point;   functionDepend; "coordinateX(F2)",
               "abs(calculate(a)-coordinateX(F2)^2)-coordinateX(F2)+calculate(b)";
               "hidden"
e[14] = F3;    point;   functionDepend; "coordinateX(P1)+3*XVector(P1,P2)/20",
               "0.0";  "hidden"
e[15] = F3';   point;   functionDepend; "coordinateX(F3)",
               "abs(calculate(a)-coordinateX(F3)^2)-coordinateX(F3)+calculate(b)";
               "hidden"
e[16] = F4;    point;   functionDepend; "coordinateX(P1)+4*XVector(P1,P2)/20",
               "0.0";  "hidden"
e[17] = F4';   point;   functionDepend; "coordinateX(F4)",
               "abs(calculate(a)-coordinateX(F4)^2)-coordinateX(F4)+calculate(b)";
               "hidden"
e[18] = F5;    point;   functionDepend; "coordinateX(P1)+5*XVector(P1,P2)/20",
               "0.0";  "hidden"
e[19] = F5';   point;   functionDepend; "coordinateX(F5)",
               "abs(calculate(a)-coordinateX(F5)^2)-coordinateX(F5)+calculate(b)";
               "hidden"
e[20] = F6;    point;   functionDepend; "coordinateX(P1)+6*XVector(P1,P2)/20",
               "0.0";  "hidden"
e[21] = F6';   point;   functionDepend; "coordinateX(F6)",
               "abs(calculate(a)-coordinateX(F6)^2)-coordinateX(F6)+calculate(b)";
               "hidden"
e[22] = F7;    point;   functionDepend; "coordinateX(P1)+7*XVector(P1,P2)/20",
               "0.0";  "hidden"
e[23] = F7';   point;   functionDepend; "coordinateX(F7)",
               "abs(calculate(a)-coordinateX(F7)^2)-coordinateX(F7)+calculate(b)";
               "hidden"
e[24] = F8;    point;   functionDepend; "coordinateX(P1)+8*XVector(P1,P2)/20",
               "0.0";  "hidden"
e[25] = F8';   point;   functionDepend; "coordinateX(F8)",
               "abs(calculate(a)-coordinateX(F8)^2)-coordinateX(F8)+calculate(b)";
```

```
                "hidden"
e[26] = F9;     point;   functionDepend; "coordinateX(P1)+9*XVector(P1,P2)/20",
                "0.0";  "hidden"
e[27] = F9';    point;   functionDepend; "coordinateX(F9)",
                "abs(calculate(a)-coordinateX(F9)^2)-coordinateX(F9)+calculate(b)";
                "hidden"
e[28] = F10;    point;   functionDepend; "coordinateX(P1)+10*XVector(P1,P2)/20",
                "0.0";  "hidden"
e[29] = F10';   point;   functionDepend; "coordinateX(F10)",
                "abs(calculate(a)-coordinateX(F10)^2)-coordinateX(F10)+calculate(b)";
                "hidden"
e[30] = F11;    point;   functionDepend; "coordinateX(P1)+11*XVector(P1,P2)/20",
                "0.0";  "hidden"
e[31] = F11';   point;   functionDepend; "coordinateX(F11)",
                "abs(calculate(a)-coordinateX(F11)^2)-coordinateX(F11)+calculate(b)";
                "hidden"
e[32] = F12;    point;   functionDepend; "coordinateX(P1)+12*XVector(P1,P2)/20",
                "0.0";  "hidden"
e[33] = F12';   point;   functionDepend; "coordinateX(F12)",
                "abs(calculate(a)-coordinateX(F12)^2)-coordinateX(F12)+calculate(b)";
                "hidden"
e[34] = F13;    point;   functionDepend; "coordinateX(P1)+13*XVector(P1,P2)/20",
                "0.0";  "hidden"
e[35] = F13';   point;   functionDepend; "coordinateX(F13)",
                "abs(calculate(a)-coordinateX(F13)^2)-coordinateX(F13)+calculate(b)";
                "hidden"
e[36] = F14;    point;   functionDepend; "coordinateX(P1)+14*XVector(P1,P2)/20",
                "0.0";  "hidden"
e[37] = F14';   point;   functionDepend; "coordinateX(F14)",
                "abs(calculate(a)-coordinateX(F14)^2)-coordinateX(F14)+calculate(b)";
                "hidden"
e[38] = F15;    point;   functionDepend; "coordinateX(P1)+15*XVector(P1,P2)/20",
                "0.0";  "hidden"
e[39] = F15';   point;   functionDepend; "coordinateX(F15)",
                "abs(calculate(a)-coordinateX(F15)^2)-coordinateX(F15)+calculate(b)";
                "hidden"
e[40] = F16;    point;   functionDepend; "coordinateX(P1)+16*XVector(P1,P2)/20",
                "0.0";  "hidden"
e[41] = F16';   point;   functionDepend; "coordinateX(F16)",
                "abs(calculate(a)-coordinateX(F16)^2)-coordinateX(F16)+calculate(b)";
                "hidden"
e[42] = F17;    point;   functionDepend; "coordinateX(P1)+17*XVector(P1,P2)/20",
                "0.0";  "hidden"
e[43] = F17';   point;   functionDepend; "coordinateX(F17)",
                "abs(calculate(a)-coordinateX(F17)^2)-coordinateX(F17)+calculate(b)";
                "hidden"
e[44] = F18;    point;   functionDepend; "coordinateX(P1)+18*XVector(P1,P2)/20",
                "0.0";  "hidden"
e[45] = F18';   point;   functionDepend; "coordinateX(F18)",
                "abs(calculate(a)-coordinateX(F18)^2)-coordinateX(F18)+calculate(b)";
                "hidden"
e[46] = F19;    point;   functionDepend; "coordinateX(P1)+19*XVector(P1,P2)/20",
                "0.0";  "hidden"
e[47] = F19';   point;   functionDepend; "coordinateX(F19)",
                "abs(calculate(a)-coordinateX(F19)^2)-coordinateX(F19)+calculate(b)";
                "hidden"
e[48] = p1;     polygon; quadrilateral;  P1,F1,F1',P1';              0;0;0;lightGray
e[49] = p2;     polygon; quadrilateral;  F1,F2,F2',F1';              0;0;0;lightGray
e[50] = p3;     polygon; quadrilateral;  F2,F3,F3',F2';              0;0;0;lightGray
e[51] = p4;     polygon; quadrilateral;  F3,F4,F4',F3';              0;0;0;lightGray
e[52] = p5;     polygon; quadrilateral;  F4,F5,F5',F4';              0;0;0;lightGray
e[53] = p6;     polygon; quadrilateral;  F5,F6,F6',F5';              0;0;0;lightGray
e[54] = p7;     polygon; quadrilateral;  F6,F7,F7',F6';              0;0;0;lightGray
e[55] = p8;     polygon; quadrilateral;  F7,F8,F8',F7';              0;0;0;lightGray
e[56] = p9;     polygon; quadrilateral;  F8,F9,F9',F8';              0;0;0;lightGray
e[57] = p10;    polygon; quadrilateral;  F9,F10,F10',F9';            0;0;0;lightGray
e[58] = p11;    polygon; quadrilateral;  F10,F11,F11',F10';          0;0;0;lightGray
e[59] = p12;    polygon; quadrilateral;  F11,F12,F12',F11';          0;0;0;lightGray
e[60] = p13;    polygon; quadrilateral;  F12,F13,F13',F12';          0;0;0;lightGray
e[61] = p14;    polygon; quadrilateral;  F13,F14,F14',F13';          0;0;0;lightGray
e[62] = p15;    polygon; quadrilateral;  F14,F15,F15',F14';          0;0;0;lightGray
```

```
e[63] = p16;   polygon; quadrilateral;  F15,F16,F16',F15';          0;0;0;lightGray
e[64] = p17;   polygon; quadrilateral;  F16,F17,F17',F16';          0;0;0;lightGray
e[65] = p18;   polygon; quadrilateral;  F17,F18,F18',F17';          0;0;0;lightGray
e[66] = p19;   polygon; quadrilateral;  F18,F19,F19',F18';          0;0;0;lightGray
e[67] = p20;   polygon; quadrilateral;  F19,P2,P2',F19';            0;0;0;lightGray
e[68] = coord; point;   coordSystem;    O,O,Ox,O,Oy,300,300,220,220; 0;red;black;0
e[69] = fkt;   line;    curve;          "t","abs(calculate(a)-t^2)-t+calculate(b)",
               -4.0, 4.0, 500;
e[70] = s1;    line;    connect;        P1,P1';                     0;0;black;0
e[71] = s2;    line;    connect;        P2,P2';                     0;0;black;0
e[72] = x0;    measure; coordinateX;    P1, -600.0, 6.0,"","";
e[73] = x1;    measure; coordinateX;    P2, -600.0, 6.0,"","";;
e[74] = m0;    measure; function;       "Functional_Integral","fkt","x0","x1",
               "50","trapezium",4.0,3.0,"Flächeninhalt = "," F.E.";

// Textfenster
// ===========

<ProblemText>
 Position = 20;40;-1;-1
  Untersuche die Funktion
  f(x) = | a - x^2 | - x + b .
</ProblemText>
```

Archimedische Spirale (Seite 158)

```
//
// Datei: Archimedische_Spirale.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
//

// Figurenbeschreibung
// ===================

e[1] = O;     point;   free;        3.0,0.0;                    "hidden"
e[2] = Ox;    point;   translation; O,1.0,0.0;                  "hidden"
e[3] = Oy;    point;   translation; O,0.0,1.0;                  "hidden"
e[4] = coord; point;   coordSystem; O,O,Ox,O,Oy,220,220,220,220; 0;red;black;0
e[5] = a;     measure; controller;  0.3,0.1,-1.0,5.0,150,"a = ","";
e[6] = fkt;   line;    curve;       "calculate(a)*t",coord,10.0,50;

// Bilddateien einlesen
// ====================

image[1] = "Archimedische_Spirale.gif",20,40
```

Logarithmische Spirale (Seite 158)

```
//
// Datei: Logarithmische_Spirale.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
//

// Figurenbeschreibung
// ===================

e[1] = O;     point;   free;        3.0,0.0;                    "hidden"
e[2] = Ox;    point;   translation; O,1.0,0.0;                  "hidden"
e[3] = Oy;    point;   translation; O,0.0,1.0;                  "hidden"
e[4] = coord; point;   coordSystem; O,O,Ox,O,Oy,220,220,220,220; 0;red;black;0
e[5] = a;     measure; controller;  0.8,0.1,-1.0,5.0,150,"a = ","";
e[6] = c;     measure; controller;  1.1,0.1,-1.0,5.0,150,"c = ","";
e[7] = fkt;   line;    curve;       "calculate(c)^(calculate(a)*t)", coord, 30.0, 500;

// Bilddateien einlesen
// ====================

image[1] = "Logarithmische_Spirale.gif",20,40
```

Hyperbolische Spirale (Seite 159)

```
//
// Datei: Hyperbolische_Spirale.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
//

// Figurenbeschreibung
// ===================

e[1] = O;     point;   free;        3.0,0.0;                    "hidden"
e[2] = Ox;    point;   translation; O,1.0,0.0;                  "hidden"
e[3] = Oy;    point;   translation; O,0.0,1.0;                  "hidden"
e[4] = coord; point;   coordSystem; O,O,Ox,O,Oy,220,220,220,220; 0;red;black;0
e[5] = a;     measure; controller;  2.0,0.1,-1.0,5.0,150,"a = ","";
e[6] = fkt;   line;    curve;       "calculate(a)/t", coord, 15.0, 500;

// Bilddateien einlesen
// ====================

image[1] = "Hyperbolische_Spirale.gif",20,20
```

Fermatsche Spirale (Seite 159)

```
//
// Datei: Fermatsche_Spirale.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
//

// Figurenbeschreibung
// ===================

e[1] = O;     point;   free;        3.0,0.0;                    "hidden"
e[2] = Ox;    point;   translation; O,1.0,0.0;                  "hidden"
e[3] = Oy;    point;   translation; O,0.0,1.0;                  "hidden"
e[4] = coord; point;   coordSystem; O,O,Ox,O,Oy,220,220,220,220; 0;red;black;0
e[5] = a;     measure; controller;  1.5,0.1,-1.0,5.0,150,"a = ","";
e[6] = fkt;   line;    curve;       "calculate(a)*SQRT(t)", coord, 80.0, 500;

// Bilddateien einlesen
// ====================

image[1] = "Fermatsche_Spirale.gif",20,20
```

Kissoide (Seite 161)

```
//
// Datei: Kissoide.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
//

// Figurenbeschreibung
// ===================

e[1]  = G1;    point;   free;          -18.0,0.0;
e[2]  = G2;    point;   free;          18.0,0.0;
e[3]  = h;     line;    straightLine;  G1,G2;
e[4]  = O;     point;   lineSlider;    -6.0,0.0,h;
e[5]  = M;     point;   lineSlider;    0.0,0.0,h;
e[6]  = P;     point;   lineSlider;    3.0,0.0,h;
e[7]  = g;     line;    perpendicular; O,h;         0;0;black;0
e[8]  = k;     circle;  radius;        M,P;         0;0;black;0
e[9]  = B;     point;   circleSlider;  0.0,3.3,k;
e[10] = l;     line;    straightLine;  P,B;
e[11] = G;     point;   intersection;  l,g;
e[12] = A;     point;   translation;   P,B,G;
e[13] = locus; line;    locus;         A,B,k,150;
e[14] = sw1;   measure; checkbox;      "Kissoide zeigen",0;

// Ein- und Ausblenden von Objekten
// ================================

hidden[1] = "if (not(calculate(sw1))) hide (locus,T)"
```

Bezier-Kurve von drei Punkten (Seite 162)

```
//
// Datei: Bezier-Kurve_von_drei_Punkten.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
//

// Figurenbeschreibung
// ===================

e[1]  = sw1;   measure; checkbox;   "Bézier-Kurve zeigen",0;
e[2]  = A;     point;   dragable;   -11.73,-7.8;
e[3]  = B;     point;   dragable;   -7.13,5.2;
e[4]  = C;     point;   dragable;   10.13,7.93;
e[5]  = s1;    line;    connect;    A,B;                "hidelabel"
e[6]  = s2;    line;    connect;    C,B;                "hidelabel"
e[7]  = P1;    point;   dragable;   -9.89,-2.58,s1;
e[8]  = P2;    point;   proportion; A,B,A,P1,B,C,B,C;
e[9]  = s3;    line;    connect;    P1,P2;              "hidelabel"
e[10] = P3;    point;   proportion; A,B,A,P1,P1,P2,P1,P2;
e[11] = locus; line;    locus;      P3,P1,s1,50;        0;0;black;0

// Ein- und Ausblenden von Objekten
// ================================

hidden[1] = "if (not(calculate(sw1))) hide (locus)"
```

Bezier-Kurve von vier Punkten (Seite 162)

```
//
// Datei: Bezier-Kurve_von_vier_Punkten.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
//

// Figurenbeschreibung
// ===================

e[1]  = sw1;   measure; checkbox;          "Hilfskonstruktion zeigen",0;
e[2]  = A;     point;   dragable;          -4.93,-5.27;
e[3]  = B;     point;   dragable;          -10.53,1.93;
e[4]  = C;     point;   dragable;          0.13,8.67;
e[5]  = D;     point;   dragable;          9.47,-6.13;
e[6]  = s1;    line;    connect;           A,B;                  "hidelabel"
e[7]  = s2;    line;    connect;           C,B;                  "hidelabel"
e[8]  = s3;    line;    connect;           C,D;                  "hidelabel"
e[9]  = P1;    point;   lineSegmentSlider; -7.41,-2.08,s1;       "hidelabel"
e[10] = P2;    point;   proportion;        A,B,A,P1,B,C,B,C;     "hidelabel"
e[11] = P3;    point;   proportion;        A,B,A,P1,C,D,C,D;     "hidelabel"
e[12] = s4;    line;    connect;           P1,P2;                "hidelabel"
e[13] = s5;    line;    connect;           P3,P2;                "hidelabel"
e[14] = P4;    point;   proportion;        A,B,A,P1,P1,P2,P1,P2; "hidelabel"
e[15] = P5;    point;   proportion;        A,B,A,P1,P2,P3,P2,P3; "hidelabel"
e[16] = s6;    line;    connect;           P4,P5;   "hidelabel"
e[17] = P6;    point;   proportion;        A,B,A,P1,P4,P5,P4,P5; "hidelabel"
e[18] = locus; line;    locus;             P6,P1,s1,50;          0;0;black;0

// Ein- und Ausblenden von Objekten
// ================================

hidden[1] = "if (not(calculate(sw1))) hide (P1,P2,P3,P4,P5,P6,s4,s5,s6)"
```

Bezier-Kurve von sechs Punkten (Seite 162)

```
//
// Datei: Bezier-Kurve_von_sechs_Punkten.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
//

// Figurenbeschreibung
// ===================

e[1]  = sw1;   measure; checkbox;          "Hilfskonstruktion zeigen",0;
e[2]  = A;     point;   dragable;          -14.13,0.27;
e[3]  = B;     point;   dragable;          -13.6,6.0;
e[4]  = C;     point;   dragable;          -3.4,9.53;
e[5]  = D;     point;   dragable;          1.13,8.4;
e[6]  = E;     point;   dragable;          3.33,2.67;
e[7]  = F;     point;   dragable;          -3.73,-1.07;
e[8]  = s1;    line;    connect;           A,B;                       "hidelabel"
e[9]  = s2;    line;    connect;           C,B;                       "hidelabel"
e[10] = s3;    line;    connect;           C,D;                       "hidelabel"
e[11] = s4;    line;    connect;           E,D;                       "hidelabel"
e[12] = s5;    line;    connect;           E,F;                       "hidelabel"
e[13] = P1;    point;   lineSegmentSlider; -13.83,3.52,s1;            "hidelabel"
e[14] = P2;    point;   proportion;        A,B,A,P1,B,C,B,C;          "hidelabel"
e[15] = P3;    point;   proportion;        A,B,A,P1,C,D,C,D;          "hidelabel"
e[16] = P4;    point;   proportion;        A,B,A,P1,D,E,D,E;          "hidelabel"
e[17] = P5;    point;   proportion;        A,B,A,P1,E,F,E,F;          "hidelabel"
e[18] = s6;    line;    connect;           P1,P2;                     "hidelabel"
e[19] = s7;    line;    connect;           P2,P3;                     "hidelabel"
e[20] = s8;    line;    connect;           P3,P4;                     "hidelabel"
e[21] = s9;    line;    connect;           P4,P5;                     "hidelabel"
e[22] = P6;    point;   proportion;        A,B,A,P1,P1,P2,P1,P2;      "hidelabel"
e[23] = P7;    point;   proportion;        A,B,A,P1,P2,P3,P2,P3;      "hidelabel"
e[24] = P8;    point;   proportion;        A,B,A,P1,P3,P4,P3,P4;      "hidelabel"
e[25] = P9;    point;   proportion;        A,B,A,P1,P4,P5,P4,P5;      "hidelabel"
e[26] = s10;   line;    connect;           P6,P7;                     "hidelabel"
e[27] = s11;   line;    connect;           P7,P8;                     "hidelabel"
e[28] = s12;   line;    connect;           P8,P9;                     "hidelabel"
e[29] = P10;   point;   proportion;        A,B,A,P1,P6,P7,P6,P7;      "hidelabel"
e[30] = P11;   point;   proportion;        A,B,A,P1,P7,P8,P7,P8;      "hidelabel"
e[31] = P12;   point;   proportion;        A,B,A,P1,P8,P9,P8,P9;      "hidelabel"
e[32] = s13;   line;    connect;           P10,P11;                   "hidelabel"
e[33] = s14;   line;    connect;           P11,P12;                   "hidelabel"
e[34] = P13;   point;   proportion;        A,B,A,P1,P10,P11,P10,P11;  "hidelabel"
e[35] = P14;   point;   proportion;        A,B,A,P1,P11,P12,P11,P12;  "hidelabel"
e[36] = s15;   line;    connect;           P13,P14;                   "hidelabel"
e[37] = P15;   point;   proportion;        A,B,A,P1,P13,P14,P13,P14;  "hidelabel"
e[38] = locus; line;    locus;             P15,P1,s1,350;             0;0;black;0


// Ein- und Ausblenden von Objekten
// ================================

hidden[1] = "if (not(calculate(sw1))) hide (P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11,
                                            P12,P13,P14,P15)"
hidden[2] = "if (not(calculate(sw1))) hide (s6,s7,s8,s9,s10,s11,s12,s13,s14,s15)"
```

Koch-Kurve (Seite 164)

```
//
// Datei: Koch-Kurve.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
//

// Figurenbeschreibung
// ===================

e[1] = A;     point;   dragable;   -5.0,0.0;
e[2] = B;     point;   dragable;   5.0,0.0;
e[3] = stufe; measure; controller; 1.0,1.0,1.0,7.0,100,"",".Stufe";
e[4] = k;     line;    curve;      "Curve_Kochkurve","A","B","stufe";
```

Sierpinski-Dreieck (Seite 164)

```
//
// Datei: Sierpinski-Dreieck.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
//

// Figurenbeschreibung
// ===================

e[1] = A;     point;   dragable;   -10.67,-8.0;
e[2] = B;     point;   dragable;   10.67,-8.0;
e[3] = C;     point;   rotation;   B,A,-1.047197551,1.0;
e[4] = stufe; measure; controller; 1.0,1.0,1.0,7.0,100,"",".Stufe";
e[5] = k;     line;    curve;      "Curve_Sierpinski","A","B","C","stufe";
```

Fraktaler Baum (Seite 165)

```
//
// Datei: Fraktaler_Baum.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
//

// Figurenbeschreibung
// ===================

e[1] = A;     point;   dragable;   -2.6,-9.67;
e[2] = B;     point;   dragable;   -3.13,-3.13;
e[3] = C;     point;   dragable;   -2.13,-4.0;
e[4] = D;     point;   dragable;   -4.13,-4.87;
e[5] = E;     point;   dragable;   -2.67,-6.33;
e[6] = stufe; measure; controller; 1.0,1.0,1.0,6.0,100,"",".Stufe";
e[7] = k;     line;    curve;      "Curve_Baeume","A","B","C","D","E","stufe";
```

Ellipse in der Taxi-Metrik (Seite 167)

```
//
// Datei: Ellipse_in_der_Taxi-Metrik.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
//

// Systemvariablen
// ===============

showLabel         = true
showGrid          = true
snapToGrid        = true
gridColor         = lightGray
MEASURE_EXACTNESS = 2

// Figurenbeschreibung
// ===================

e[1]  = M1; point;   fixed;         -4.0,2.0;
e[2]  = M2; point;   fixed;         4.0, 2.0;
e[3]  = A;  point;   free;          7.0,4.0;
e[4]  = B;  point;   free;          1.0,6.0;
e[5]  = C;  point;   free;          -7.0,5.0;
e[6]  = D;  point;   free;          -8.0,1.0;
e[7]  = E;  point;   free;          -4.0,-2.0;
e[8]  = F;  point;   free;          6.0,-2.0;
e[9]  = p;  polygon; hexagon;       A,B,C,D,E,F; 0;0;blue;0
e[10] = P;  point;   polygonSlider; p,1.0,-2.0;
e[11] = m0; measure; calculate;     "abs(XVector(M1,P))+abs(YVector(M1,P))",
            -11.0,9.0,"d(M1,P) = ","";
e[12] = m1; measure; calculate;     "abs(XVector(M2,P))+abs(YVector(M2,P))",
            -11.0,8.0,"d(M2,P) = ","";
e[13] = m2; measure; calculate;     "calculate(m0)+calculate(m1)",-11.0,7.0,
            "d(M1,P) + d(M2,P) = ","";

// Textfenster
// ===========

<Textbox>
 Position = 20;300;-1;-1
   Aufgabe:
   Die sechs beweglichen Punkte A,B,C,D,E und F
   bilden ein Sechseck auf dem der Punkt P bewegt
   werden kann.
   Kann man das Sechseck so verschieben, daß die
   Summe von der beiden Abstände |P M1| und |P M2|
   gemessen in der Taxi-Metrik konstant bleibt,
   egal an welcher Stelle sich P befindet?
</Textbox>

<Textbox>
 Textbox = 320;20;-1;-1
   Taxi-Metrik:
   d(A,B) := |Ax-Bx| + |Ay-By|
</Textbox>
```

Abstandssumme in der Taxi-Metrik (Seite 168)

```
//
// Datei: Abstandssumme_in_der_Taxi-Metrik.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
// Aufgabe aus Heinz-Jörg Claus "Extremwertaufgaben"
// Darmstadt: Wissenschaftliche Buchgesellschaft 1992, S. 7
//

// Systemvariablen
// ===============

showLabel  = true
showGrid   = true
snapToGrid = true
gridColor  = lightGray

// Figurenbeschreibung
// ===================

e[1] = P1; point;   fixed;  -12.0,5.0;
e[2] = P2; point;   fixed;  -9.0,1.0;
e[3] = P3; point;   fixed;  -1.0,0.0;
e[4] = P4; point;   fixed;  0.0,-7.0;
e[5] = P5; point;   fixed;  8.0,-4.0;
e[6] = T;  point;   free;   -3.0,-3.0;
e[7] = m2; measure; button; "Hilfe","help";
e[8] = m3; measure; button; "Auswertung","evaluate";

// Textfenster
// ===========

<Textbox>
 Position = 20;20;-1;-1
 In der Stadt Orthopolis wollen sich die fünf
 Bewohner der Häuser P1 ... P5 an dem Ort mit
 der kürzesten Wegesumme treffen.
 Bewege den Punkt T an diese Stelle.
</Textbox>

// Hilfen
// ======

<Help>
   Betrachte immer Paare von
   zwei Punkten auf einmal.

   In welchem Bereich liegt ein
   Punkt mit der kürzesten Wegesumme?
</Help>

// Antwortanalyse
// ==============

<Problem>
 MAX_ANSWER = 3
 condition[1] = "isIncident(T,P3)"
 condition[2] = "isIncident(T,P1)"
 condition[3] = "isIncident(T,P2)"
 condition[4] = "isIncident(T,P4)"
 condition[5] = "isIncident(T,P5)"
 condition[6] = "distance(T,P3) < 1.8"
 condition[7] = "distance(T,P3) > 1.8"
/Problem>

<Answer 1>
 key = "condition[1]"
 comment[1] = "Richtig. /n
              Bei einer ungeraden Anzahl von Punkten,/n
              fällt der Punkt mit der kürzesten/n
              Wegesumme mit dem 'mittlersten' Punkt/n
```

```
             zusammen.
</Answer 1>

<Answer 2>
 key = "condition[2] | condition[3] | condition[4] | condition[5]"
 comment[1] = "Ihre Antwort ist nicht richtig. /n
              Der Punkt T hat noch nicht die richtige Position./n
              Versuchen Sie es noch einmal."
 comment[2] = "Ihre Antwort ist immer noch nicht richtig. /n
              Der Punkt T hat noch nicht die richtige Position./n
              Versuchen Sie es noch einmal."
 comment[3] = "Ihre Antwort ist immer noch nicht richtig. /n
              Bei einer ungeraden Anzahl von Punkten,/n
              fällt der Punkt mit der kürzesten/n
              Wegesumme mit dem 'mittlersten' Punkt/nzusammen.
</Answer 2>

<Answer 3>
 key = "condition[6]"
 comment[1] = "Ihre Lösung ist schon ganz gut,/n
              läßt sich aber noch etwas verbessern./n
              Versuchen Sie es noch einmal."
 comment[2] = "Ihre Antwort ist immer noch nicht richtig. /n
              Bei einer ungeraden Anzahl von Punkten,/n
              fällt der Punkt mit der kürzesten/n
              Wegesumme mit dem 'mittlersten' Punkt/n
              zusammen.
</Answer 3>

<Answer 4>
 key = "condition[7]"
 comment[1] = "Ihre Antwort ist nicht richtig. /n
              Der Punkt T hat noch nicht die richtige Position./n
              Es läßt sich eine Lage mit einer kürzeren Wegesumme finden./n
              Versuchen Sie es noch einmal."
 comment[2] = "Ihre Antwort ist immer noch nicht richtig. /n
              Der Punkt T hat immer noch nicht die richtige Position./n
              Versuchen Sie es noch einmal."
 comment[3] = "Ihre Antwort ist immer noch nicht richtig. /n
              Bei einer ungeraden Anzahl von Punkten,/n
              fällt der Punkt mit der kürzesten/n
              Wegesumme mit dem 'mittlersten' Punkt/nzusammen.
</Answer 4>
```

Kreis in der Maximum-Metrik (Seite 168)

```
//
// Datei: Kreis_in_der_Maximum-Metrik.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
// Aufgabe aus Heinz-Jörg Claus "Extremwertaufgaben in metrischen Räumen"
// MU 5/82, Seite 27-58
//

// Systemvariablen
// ===============

showLabel         = true
showGrid          = true
snapToGrid        = true
gridColor         = lightGray
MEASURE_EXACTNESS = 2

// Figurenbeschreibung
// ===================

e[1]  = M;  point;   fixed;      0.0,3.0;
e[2]  = A;  point;   free;       3.0,6.0;
e[3]  = B;  point;   free;       -3.0,4.0;
e[4]  = C;  point;   free;       -2.0,0.0;
e[5]  = D;  point;   free;       3.0,-1.0;
e[6]  = p;  polygon; octagon;    A,B,C,D,A,D,C,B; 0;0;blue;0
e[7]  = P;  point;   areaSlider; p,0.0,5.0;
e[8]  = m0; measure; calculate;  "max(abs(XVector(M,P)),abs(YVector(M,P)))",
           -5.0,10.0,"d(P,M) = ","";
e[9]  = P1; point;   fixed;      -3.0, 6.0; "hidden"
e[10] = P2; point;   fixed;      -3.0, 0.0; "hidden"
e[11] = P3; point;   fixed;      3.0, 0.0; "hidden"
e[12] = P4; point;   fixed;      3.0, 6.0; "hidden"
e[13] = m6; measure; button;     "Hilfe","help";
e[14] = m7; measure; button;     "Auswertung","evaluate";

// Textfenster
// ===========

<Textbox>
 Position = 20;330;300;140
   Die vier beweglichen Punkte A,B,C und D bilden
   ein Viereck auf dem der Punkt P bewegt werden
   kann.
   Kann man das Viereck so verschieben, daß
   der Abstand von P zu M gemessen in der
   Maximum-Metrik immer konstant = 3 bleibt,
   egal an welcher Stelle sich P befindet?
</Textbox>

<Textbox>
 Textbox = 320;20;-1;-1
   Maximum-Metrik:

   d(A,B) := max( |Ax-Bx|, |Ay-By| )
</Textbox>

// Hilfen
// ======

<Help>
   Betrachten Sie die Definition der
   Maximum-Metrik. Wie können zwei
   Punkte zueinander im Koordinaten-
   system verschoben werden, damit
   sie denselben Abstand behalten?
</Help>

// Antwortanalyse
// ==============
```

```
<Problem>
 MAX_ANSWER = 3
 condition[1] = "isIncident(P1,A) | isIncident(P2,A) | isIncident(P3,A) | isIncident(P4,A)"
 condition[2] = "isIncident(P1,B) | isIncident(P2,B) | isIncident(P3,B) | isIncident(P4,B)"
 condition[3] = "isIncident(P1,C) | isIncident(P2,C) | isIncident(P3,C) | isIncident(P4,C)"
 condition[4] = "isIncident(P1,D) | isIncident(P2,D) | isIncident(P3,D) | isIncident(P4,D)"
 condition[5] = "NOT( isIncident(A,B) | isIncident(A,C) | isIncident(A,D) |
                     isIncident(C,B) | isIncident(D,B) | isIncident(C,D) )"
</Problem>

<Answer 1>
 key = "condition[1] & condition[2] & condition[3] & condition[4] & condition[5]"
 comment[1] = "Richtig. /n
               Weil das Viereck die Menge aller Punkte ist,/n
               die von M denselben Abstand in der Maximum-Metrik besitzen, /n
               spricht man auch von einem Kreis um M."
</Answer 1>

<Answer 2>
 key = "not(condition[1])"
 comment[1] = "Ihre Antwort ist nicht richtig. /n
               Der Punkt A hat noch nicht die richtige Position./n
               Versuchen Sie es noch einmal."
 comment[2] = "Ihre Antwort ist immer noch nicht richtig. /n
               Der Punkt A hat noch nicht die richtige Position./n
               Versuchen Sie es noch einmal."
 comment[3] = "Ihre Antwort ist immer noch nicht richtig. /n
               Der Abstand von A zu M muß, sowohl auf der horizontalen /n
               als auch auf der vertikalen Koordinatenachse, 3 betragen."
</Answer 2>

<Answer 3>
 key = "not(condition[2])"
 comment[1] = "Ihre Antwort ist nicht richtig. /n
               Der Punkt B hat noch nicht die richtige Position./n
               Versuchen Sie es noch einmal."
 comment[2] = "Ihre Antwort ist immer noch nicht richtig. /n
               Der Punkt B hat noch nicht die richtige Position./n
               Versuchen Sie es noch einmal."
 comment[3] = "Ihre Antwort ist immer noch nicht richtig. /n
               Der Abstand von B zu M muß, sowohl auf der horizontalen /n
               als auch auf der vertikalen Koordinatenachse, 3 betragen."
</Answer 2>

<Answer 4>
 key = "not(condition[3])"
 comment[1] = "Ihre Antwort ist nicht richtig. /n
               Der Punkt C hat noch nicht die richtige Position./n
               Versuchen Sie es noch einmal."
 comment[2] = "Ihre Antwort ist immer noch nicht richtig. /n
               Der Punkt C hat noch nicht die richtige Position./n
               Versuchen Sie es noch einmal."
 comment[3] = "Ihre Antwort ist immer noch nicht richtig. /n
               Der Abstand von C zu M muß, sowohl auf der horizontalen /n
               als auch auf der vertikalen Koordinatenachse, 3 betragen."
</Answer 4>

<Answer 5>
 key = "not(condition[4])"
 comment[1] = "Ihre Antwort ist nicht richtig. /n
               Der Punkt D hat noch nicht die richtige Position./n
               Versuchen Sie es noch einmal."
 comment[2] = "Ihre Antwort ist immer noch nicht richtig. /n
               Der Punkt D hat noch nicht die richtige Position./n
               Versuchen Sie es noch einmal."
 comment[3] = "Ihre Antwort ist immer noch nicht richtig. /n
               Der Abstand von D zu M muß, sowohl auf der horizontalen /n
               als auch auf der vertikalen Koordinatenachse, 3 betragen."
</Answer 5>
```

```
<Answer 6>
 key = "not(condition[5])"
 comment[1] = "Ihre Antwort ist nicht richtig. /n
               Mindestens zwei Punkte liegen aufeinander./n
               Versuchen Sie es noch einmal."
 comment[2] = "Ihre Antwort ist immer noch nicht richtig. /n
               Wenn zwei Punkte aufeinanderliegen kann kein echtes Viereck
               gebildet werden./n
               Versuchen Sie es noch einmal."
 comment[3] = "Ihre Antwort ist immer noch nicht richtig. /n
               Der Abstand von jedem der Punkte A,B,C,D zu M muß, sowohl /n
               auf der horizontalen als auch auf der vertikalen Koordinatenachse, /n
               3 betragen."
</Answer 6>
```

Eisenbahn-Metrik (Seite 169)

```
//
// Datei: Eisenbahn-Metrik.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
//

// Systemvariablen
// ===============

showLabel         = true
showGrid          = true
gridColor         = lightGray
MEASURE_EXACTNESS = 2

// Figurenbeschreibung
// ===================

e[1] = A;  point;   free;         -6.0, 6.0;
e[2] = B;  point;   free;         -8.0,-2.0;
e[3] = P;  point;   fixed;        -3.0, 3.0;
e[4] = a;  line;    connect;      A,P;       "hideLabel"
e[5] = b;  line;    connect;      B,P;       "hideLabel"
e[6] = c;  line;    straightline; B,A;       "hidden"
e[7] = m0; measure; calculate;    "if (isIncident(P,c)) then (distance(A,B))
           else (distance(A,P)+distance(B,P))",-5.0,10.0,"d(A,B) = ","";

// Textfenster
// ===========

<Textbox>
 Position = 330;20;-1;-1
   Eisenbahn-Metrik:
   d(A,B) := Wenn P auf der Geraden AB liegt,
             dann d=|AB|, sonst d = |AP| + |PB| .
</Textbox>
```

Mengensprache (Seite 170)

```
//
// Datei: Mengensprache.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
//

// Figurenbeschreibung
// ===================

e[1]  = A1; point;   free;   3.0,4.0;  "hidden"
e[2]  = A2; point;   free;   8.0,4.0;  "hidden"
e[3]  = C1; point;   free;   7.0,-2.0; "hidden"
e[4]  = C2; point;   free;   7.0,3.0;  "hidden"
e[5]  = B1; point;   free;   10.0,4.0; "hidden"
e[6]  = B2; point;   free;   5.0,4.0;  "hidden"
e[7]  = A;  circle;  radius; A1,A2;    red;0;red;0
e[8]  = B;  circle;  radius; B1,B2;    gray;0;gray;0
e[9]  = C;  circle;  radius; C1,C2;    blue;0;blue;0
e[10] = P1; point;   free;   2.0,10.0;
e[11] = P2; point;   free;   3.5,10.0;
e[12] = P3; point;   free;   5.0,10.0;
e[13] = P4; point;   free;   6.5,10.0;
e[14] = P5; point;   free;   8.0,10.0;
e[15] = P6; point;   free;   9.5,10.0;
e[16] = P7; point;   free;   11.0,10.0;
e[17] = P8; point;   free;   12.5,10.0;
e[18] = m1; measure; button; "Auswertung","evaluate";

// Bild-Dateien einbinden
// ======================

image[1] = "Mengensprache.gif", 5, 5

// Aufgabenanalyse
// ===============

<Problem>
 MAX_ANSWER = 0
 condition[1] = "isIncluded(P1,A)&NOT(isIncluded(P1,B))&isIncluded(P1,C)"
 condition[2] = "isIncluded(P2,A)&isIncluded(P2,B)&NOT(isIncluded(P2,C))"
 condition[3] = "isIncluded(P3,A)&NOT(isIncluded(P3,B))&isIncluded(P3,C)"
 condition[4] = "isIncluded(P4,A)&isIncluded(P4,B)&NOT(isIncluded(P4,C))"
 condition[5] = "NOT(isIncluded(P5,A))&NOT(isIncluded(P5,B))&isIncluded(P5,C)"
 condition[6] = "NOT(isIncluded(P6,A))&isIncluded(P6,B)&NOT(isIncluded(P6,C))"
 condition[7] = "NOT(isIncluded(P7,A))&NOT(isIncluded(P7,B))&isIncluded(P7,C)"
 condition[8] = "NOT(isIncluded(P8,A))&isIncluded(P8,B)&NOT(isIncluded(P8,C))"
</Problem>

<Answer 1>
 key = "condition[1]&condition[2]&condition[3]&condition[4]&condition[5]&
        condition[6]&condition[7]&condition[8]"
 comment[1] = "Richtig. /nBei dieser Anordnung sind alle fünf Aussagen wahr.
</Answer 1>

<Answer 2>
 key = "NOT(condition[1])"
 comment[1] = "Ihre Antwort ist nicht richtig. /n
               Der Punkt P1 ist falsch zugeordnet./n
               Versuchen Sie es noch einmal."
</Answer 2>

<Answer 3>
 key = "NOT(condition[2])"
 comment[1] = "Ihre Antwort ist nicht richtig. /n
               Der Punkt P2 ist falsch zugeordnet./n
               Versuchen Sie es noch einmal."
</Answer 3>

<Answer 4>
 key = "NOT(condition[3])"
```

```
 comment[1] = "Ihre Antwort ist nicht richtig. /n
              Der Punkt P3 ist falsch zugeordnet./n
              Versuchen Sie es noch einmal."
</Answer 4>

<Answer 5>
 key = "NOT(condition[4])"
 comment[1] = "Ihre Antwort ist nicht richtig. /n
              Der Punkt P4 ist falsch zugeordnet./n
              Versuchen Sie es noch einmal."
</Answer 5>

<Answer 6>
 key = "NOT(condition[5])"
 comment[1] = "Ihre Antwort ist nicht richtig. /n
              Der Punkt P5 ist falsch zugeordnet./n
              Versuchen Sie es noch einmal."
</Answer 6>

<Answer 7>
 key = "NOT(condition[6])"
 comment[1] = "Ihre Antwort ist nicht richtig. /n
              Der Punkt P6 ist falsch zugeordnet./n
              Versuchen Sie es noch einmal."
</Answer 7>

<Answer 8>
 key = "NOT(condition[7])"
 comment[1] = "Ihre Antwort ist nicht richtig. /n
              Der Punkt P7 ist falsch zugeordnet./n
              Versuchen Sie es noch einmal."
</Answer 8>

<Answer 9>
 key = "NOT(condition[8])"
 comment[1] = "Ihre Antwort ist nicht richtig. /n
              Der Punkt P8 ist falsch zugeordnet./n
              Versuchen Sie es noch einmal."
</Answer 9>
```

Labyrinth-Figur (Seite 172)

```
//
// Datei: Labyrinth-Figur.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
//

// Figurenbeschreibung
// ===================

e[1] = B;    point;   free;     11.8,-0.3;
e[2] = Pset; line;    pointSet; "./pics/Maze01a.jpg",125000,0; "hidden"
e[3] = A;    point;   dragable; 10.65,0.05,Pset;
e[4] = b;    measure; button;   "Ortsspur löschen","clearTrace"

// Bild-Dateien einbinden
// ======================

image[1] = ".\pics\Maze01b.gif", 0, 0
```

Labyrinth-Figur (Seite 172)

```
//
// Datei: Zweite_Labyrinth-Figur.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
//

// Figurenbeschreibung
// ===================

e[1] = B;    point;   free;     11.8,-0.3;
e[2] = Pset; line;    pointSet; "./pics/Maze02c.gif",24000,5; "hidden"
e[3] = A;    point;   dragable; 12.0,0.3,Pset;
e[4] = b;    measure; button;   "Ortsspur löschen","clearTrace"

// Bild-Dateien einbinden
// ======================

image[1] = ".\pics\Maze02b.gif", 0, 0
```

Tangram (Seite 173)

```
//
// Datei: Tangram_Vase.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
//

// Systemvariablen
// ===============

allPointsDragable = true
showLabel         = false

// Figurenbeschreibung
// ===================

e[1]  = A;   point;   fixed;          -3.35,-0.03;              black;red;black;smallsquare
e[2]  = A';  point;   translation;    A,1.5,0.0;                "hidden"
e[3]  = kA;  circle;  radius;         A,A';                     "hidden"
e[4]  = A''; point;   circleSlider;   kA,-5.82,2.45;            black;red;black;smallcircle
e[5]  = A1;  point;   translation;    A,-2.35702260,4.71404520; "hidden"
e[6]  = A2;  point;   translation;    A,-2.35702260,-2.3570226; "hidden"
e[7]  = A3;  point;   translation;    A,4.71404520,-2.35702260; "hidden"
e[8]  = A1'; point;   rotation;       A1,A,A',A,A'';            "hidden"
e[9]  = A2'; point;   rotation;       A2,A,A',A,A'';            "hidden"
e[10] = A3'; point;   rotation;       A3,A,A',A,A'';            "hidden"
e[11] = a;   polygon; triangle;       A1',A2',A3';              0;0;blue;green
e[12] = a';  line;    connect;        A,A'';                    0;0;blue;0
e[13] = B;   point;   fixed;          -0.03,3.32;               black;red;black;smallsquare
e[14] = B';  point;   translation;    B,1.5,0.0;                "hidden"
e[15] = kB;  circle;  radius;         B,B';                     "hidden"
e[16] = B''; point;   circleSlider;   kB,2.47,5.81;             black;red;black;smallcircle
e[17] = B1;  point;   translation;    B,-2.35702260,4.71404520; "hidden"
e[18] = B2;  point;   translation;    B,-2.35702260,-2.3570226; "hidden"
e[19] = B3;  point;   translation;    B,4.71404520,-2.35702260; "hidden"
e[20] = B1'; point;   rotation;       B1,B,B',B,B'';            "hidden"
e[21] = B2'; point;   rotation;       B2,B,B',B,B'';            "hidden"
e[22] = B3'; point;   rotation;       B3,B,B',B,B'';            "hidden"
e[23] = b;   polygon; triangle;       B1',B2',B3';              0;0;blue;green
e[24] = b';  line;    connect;        B,B'';                    0;0;blue;0
e[25] = C;   point;   fixed;          -2.50,-4.18;              black;red;black;smallsquare
e[26] = C';  point;   translation;    C,1.0,0.0;                "hidden"
e[27] = kC;  circle;  radius;         C,C';                     "hidden"
e[28] = C''; point;   circleSlider;   kC,-4.61,-6.31;           black;red;black;smallcircle
e[29] = C1;  point;   translation;    C,-1.17851130,2.35702260; "hidden"
e[30] = C2;  point;   translation;    C,-1.17851130,-1.1785113; "hidden"
e[31] = C3;  point;   translation;    C,2.35702260,-1.17851130; "hidden"
e[32] = C1'; point;   rotation;       C1,C,C',C,C'';            "hidden"
e[33] = C2'; point;   rotation;       C2,C,C',C,C'';            "hidden"
e[34] = C3'; point;   rotation;       C3,C,C',C,C'';            "hidden"
e[35] = c;   polygon; triangle;       C1',C2',C3';              0;0;blue;green
e[36] = c';  line;    connect;        C,C'';                    0;0;blue;0
e[37] = D;   point;   fixed;          3.3,-3.32;                black;red;black;smallsquare
e[38] = D';  point;   translation;    D,1.0,0.0;                "hidden"
e[39] = kD;  circle;  radius;         D,D';                     "hidden"
e[40] = D''; point;   circleSlider;   kD,3.3,-0.33;             black;red;black;smallcircle
e[41] = D1;  point;   translation;    D,-1.66666667,3.33333333; "hidden"
e[42] = D2;  point;   translation;    D,-1.66666667,-1.6666667; "hidden"
e[43] = D3;  point;   translation;    D,3.33333333,-1.66666667; "hidden"
e[44] = D1'; point;   rotation;       D1,D,D',D,D'';            "hidden"
e[45] = D2'; point;   rotation;       D2,D,D',D,D'';            "hidden"
e[46] = D3'; point;   rotation;       D3,D,D',D,D'';            "hidden"
e[47] = d;   polygon; triangle;       D1',D2',D3';              0;0;blue;green
e[48] = d';  line;    connect;        D,D'';                    0;0;blue;0
e[49] = E;   point;   fixed;          1.68,0.0;                 black;red;black;smallsquare
e[50] = E';  point;   translation;    E,1.0,0.0;                "hidden"
e[51] = kE;  circle;  radius;         E,E';                     "hidden"
e[52] = E''; point;   circleSlider;   kE,3.8,-2.12;             black;red;black;smallcircle
e[53] = E1;  point;   translation;    E,-1.17851130,2.35702260; "hidden"
e[54] = E2;  point;   translation;    E,-1.17851130,-1.17851130; "hidden"
e[55] = E3;  point;   translation;    E,2.35702260,-1.17851130; "hidden"
```

```
e[56] = E1'; point;   rotation;       E1,E,E',E,E'';             "hidden"
e[57] = E2'; point;   rotation;       E2,E,E',E,E'';             "hidden"
e[58] = E3'; point;   rotation;       E3,E,E',E,E'';             "hidden"
e[59] = e;   polygon; triangle;       E1',E2',E3';               0;0;blue;green
e[60] = e';  line;    connect;        E,E'';                     0;0;blue;0
e[61] = sw;  measure; checkbox;       "Parallelogramm wenden",1;
e[62] = F;   point;   fixed;          0.0,-2.5;                  black;red;black;smallsquare
e[63] = F';  point;   translation;    F,1.0,0.0;                 "hidden"
e[64] = kF;  circle;  radius;         F,F';                      "hidden"
e[65] = F''; point;   circleSlider;   kF,2.13,-0.38;             black;red;black;smallcircle
e[66] = F1;  point;   translation;    F,1.76776695,1.76776695;   "hidden"
e[67] = F2;  point;   translation;    F,-1.76776695,1.76776695;  "hidden"
e[68] = F3;  point;   translation;    F,-1.76776695,-1.76776695; "hidden"
e[69] = F4;  point;   translation;    F, 1.76776695,-1.76776695; "hidden"
e[70] = F1'; point;   rotation;       F1,F,F',F,F'';             "hidden"
e[71] = F2'; point;   rotation;       F2,F,F',F,F'';             "hidden"
e[72] = F3'; point;   rotation;       F3,F,F',F,F'';             "hidden"
e[73] = F4'; point;   rotation;       F4,F,F',F,F'';             "hidden"
e[74] = f;   polygon; quadrilateral;  F1',F2',F3',F4';           0;0;blue;green
e[75] = f';  line;    connect;        F,F'';                     0;0;blue;0
e[76] = G;   point;   fixed;          3.75,1.25;                 black;red;black;smallsquare
e[77] = G';  point;   translation;    G,1.0,0.0;                 "hidden"
e[78] = kG;  circle;  radius;         G,G';                      "hidden"
e[79] = G''; point;   circleSlider;   kG,6.75,1.25;              black;red;black;smallcircle
e[80] = G1;  point;   functionDepend; "if (calculate(sw)) then (coordinateX(G)+1.25)
             else (coordinateX(G)-1.25)","coordinateY(G)+3.75"; "hidden"
e[81] = G2;  point;   functionDepend; "if (calculate(sw)) then (coordinateX(G)+1.25)
             else (coordinateX(G)-1.25)","coordinateY(G)-1.25"; "hidden"
e[82] = G3;  point;   functionDepend; "if (calculate(sw)) then (coordinateX(G)-1.25)
             else (coordinateX(G)+1.25)","coordinateY(G)-3.75"; "hidden"
e[83] = G4;  point;   functionDepend; "if (calculate(sw)) then (coordinateX(G)-1.25)
             else (coordinateX(G)+1.25)","coordinateY(G)+1.25"; "hidden"
e[84] = G1'; point;   rotation;       G1,G,G',G,G'';             "hidden"
e[85] = G2'; point;   rotation;       G2,G,G',G,G'';             "hidden"
e[86] = G3'; point;   rotation;       G3,G,G',G,G'';             "hidden"
e[87] = G4'; point;   rotation;       G4,G,G',G,G'';             "hidden"
e[88] = g;   polygon; quadrilateral;  G1',G2',G3',G4';           0;0;blue;green
e[89] = g';  line;    connect;        G,G'';                     0;0;blue;0

// Textfenster
// ===========

 <Textbox>
  Textbox = 10;10;250;110
  Wie kann man die beiden folgenden
  Figuren legen?
</Textbox>

// Bild-Dateien einbinden
// ======================

image[1] = "Tangram01.gif", 60, 40
```

Problem der acht Damen (Seite 175)

```
//
// Datei: Problem_der_acht_Damen.script
// Autor: Timo Ehmke (ehmke@uni-flensburg.de)
//

// Systemvariablen
// ===============

gridSize          = 20
snapToGrid        = true
showGrid          = false
backgroundColor   = white
controlPanelColor = white

// Figurenbeschreibung
// ===================

e[1]  = O;          point;   fixed;     0.0,0.0;    "hidden"
e[2]  = Qu1;        point;   free;      -13.0,7.0;  "hidden"
e[3]  = Qu2;        point;   free;      -13.0,5.0;  "hidden"
e[4]  = Qu3;        point;   free;      -13.0,3.0;  "hidden"
e[5]  = Qu4;        point;   free;      -13.0,1.0;  "hidden"
e[6]  = Qu5;        point;   free;      -13.0,-1.0; "hidden"
e[7]  = Qu6;        point;   free;      -13.0,-3.0; "hidden"
e[8]  = Qu7;        point;   free;      -13.0,-5.0; "hidden"
e[9]  = Qu8;        point;   free;      -13.0,-7.0; "hidden"
e[10] = f1;         measure; function;  "Functional_EightQueens","Qu1","Qu2","Qu3","Qu4",
                    "Qu5","Qu6","Qu7","Qu8","analyse",13.0,10.0,"f = ",""; "hidden"
e[11] = f2;         measure; function;  "Functional_EightQueens","Qu1","Qu2","Qu3","Qu4",
                    "Qu5","Qu6","Qu7","Qu8","hint_x",13.0,9.0,"x = ",""; "hidden"
e[12]=  f3;         measure; function;  "Functional_EightQueens","Qu1","Qu2","Qu3","Qu4",
                    "Qu5","Qu6","Qu7","Qu8","hint_y",13.7,4.8,"","";
e[13] = Vorschlag:; point;   free;      9.0,5.0;    black;white;white;0
e[14] = A;          point;   free;      13.0,5.0;   black;white;white;0
e[15] = B;          point;   free;      13.0,5.0;   black;white;white;0
e[16] = C;          point;   free;      13.0,5.0;   black;white;white;0
e[17] = D;          point;   free;      13.0,5.0;   black;white;white;0
e[18] = E;          point;   free;      13.0,5.0;   black;white;white;0
e[19] = F;          point;   free;      13.0,5.0;   black;white;white;0
e[20] = G;          point;   free;      13.0,5.0;   black;white;white;0
e[21] = H;          point;   free;      13.0,5.0;   black;white;white;0
e[22] = ---;        point;   free;      13.0,5.0;   black;white;white;0
e[23] = sw1;        measure; checkbox;  "Bewertung zeigen",0;
e[24] = sw2;        measure; checkbox;  "Vorschlag zeigen",0;
e[25] = cond1;      measure; calculate; "if (calculate(f1) != 0.0) then (1.0) else (0.0)";
e[26] = cond2;      measure; calculate; "if (calculate(f1) != 1.0) then (1.0) else (0.0)";
e[27] = cond3;      measure; calculate; "if (calculate(f1) != 2.0) then (1.0) else (0.0)";
e[28] = cond4;      measure; calculate; "if (calculate(f1) != 3.0) then (1.0) else (0.0)";
e[29] = cond5;      measure; calculate; "if (calculate(f1) != 4.0) then (1.0) else (0.0)";
e[30] = cond6;      measure; calculate; "if (calculate(f1) != 5.0) then (1.0) else (0.0)";
e[31] = cond7;      measure; calculate; "if (calculate(f1) != 6.0) then (1.0) else (0.0)";
e[32] = cond8;      measure; calculate; "if (calculate(f2) != 1.0) then (1.0) else (0.0)";
e[33] = cond9;      measure; calculate; "if (calculate(f2) != 2.0) then (1.0) else (0.0)";
e[34] = cond10;     measure; calculate; "if (calculate(f2) != 3.0) then (1.0) else (0.0)";
e[35] = cond11;     measure; calculate; "if (calculate(f2) != 4.0) then (1.0) else (0.0)";
e[36] = cond12;     measure; calculate; "if (calculate(f2) != 5.0) then (1.0) else (0.0)";
e[37] = cond13;     measure; calculate; "if (calculate(f2) != 6.0) then (1.0) else (0.0)";
e[38] = cond14;     measure; calculate; "if (calculate(f2) != 7.0) then (1.0) else (0.0)";
e[39] = cond15;     measure; calculate; "if (calculate(f2) != 8.0) then (1.0) else (0.0)";
e[40] = cond16;     measure; calculate; "if (calculate(f2) = -1.0) then (1.0) else (0.0)";


// Ein- und Ausblenden von Objekten
// ================================

hidden[1]  = "if (not(calculate(sw2))) hide (Vorschlag:)"
hidden[2]  = "if (not(calculate(sw1))|calculate(cond2)) hide (Textbox_2)"
hidden[3]  = "if (not(calculate(sw1))|calculate(cond3)) hide (Textbox_3)"
hidden[4]  = "if (not(calculate(sw1))|calculate(cond4)) hide (Textbox_4)"
```

```
hidden[5]  = "if (not(calculate(sw1))|calculate(cond5)) hide (Textbox_5)"
hidden[6]  = "if (not(calculate(sw1))|calculate(cond6)) hide (Textbox_6)"
hidden[7]  = "if (not(calculate(sw1))|calculate(cond7)) hide (Textbox_7)"
hidden[8]  = "if (not(calculate(sw2))|calculate(cond8)) hide (A)"
hidden[9]  = "if (not(calculate(sw2))|calculate(cond9)) hide (B)"
hidden[10] = "if (not(calculate(sw2))|calculate(cond10)) hide (C)"
hidden[11] = "if (not(calculate(sw2))|calculate(cond11)) hide (D)"
hidden[12] = "if (not(calculate(sw2))|calculate(cond12)) hide (E)"
hidden[13] = "if (not(calculate(sw2))|calculate(cond13)) hide (F)"
hidden[14] = "if (not(calculate(sw2))|calculate(cond14)) hide (G)"
hidden[15] = "if (not(calculate(sw2))|calculate(cond15)) hide (H)"
hidden[16] = "if (not(calculate(sw2))|calculate(cond16)) hide (f3)"
hidden[17] = "if (not(calculate(sw2))|not(calculate(cond16))) hide (---)"

// Textfenster
// ===========

<TextBox>
 Position = 150;10;340;60
   Aufgabe:
   Plazieren Sie die acht Damen so auf dem Schachbrett,
   daß sie sich nicht gegenseitig bedrohen.
</TextBox>

<TextBox>
 Position = 150;10;340;60
   Diese Stellung kann nicht ausgewertet werden.
   Prüfen Sie noch einmal, ob alle Damen
   korrekt auf dem Schachbrett plaziert sind.
</TextBox>

<TextBox>
 Position = 150;10;340;60
   Diese Stellung ist leider falsch.
   Mindestens zwei Damen bedrohen
   einander.
</TextBox>

<TextBox>
 Position = 150;10;340;60
   Gut.
   Setzen Sie die nächste Dame.
</TextBox>

<TextBox>
 Position = 150;10;340;60
   Der letzte Versuch war nicht so gut,
   denn jetzt gibt es keine Lösung mehr.
</TextBox>

<TextBox>
 Position = 150;10;340;60
   Gut.
   Setzen Sie die nächste Dame.
</TextBox>

<TextBox>
 Position = 150;10;340;60
   Richtig.
   In dieser Brettstellung bedrohen
   sich die acht Damen nicht.
</TextBox>

// Bild-Dateien einbinden
// ======================

image[1] = "Chessbord.gif", 0, -1, -1
image[2] = "Chess_White_Queen.gif", Qu1, -1, -1
image[3] = "Chess_White_Queen.gif", Qu2, -1, -1
image[4] = "Chess_White_Queen.gif", Qu3, -1, -1
image[5] = "Chess_White_Queen.gif", Qu4, -1, -1
```

```
image[6] = "Chess_White_Queen.gif", Qu5, -1, -1
image[7] = "Chess_White_Queen.gif", Qu6, -1, -1
image[8] = "Chess_White_Queen.gif", Qu7, -1, -1
image[9] = "Chess_White_Queen.gif", Qu8, -1, -1
```

Anhang D
Quellcode ausgewählter Klassen

Die Klasse `Functional_PickscheFormel` (Seite 109)

```
public class Functional_PickscheFormel extends Functional{

  int type, numBorder, numInside, n, step, choice;
  String term;
  PolygonElement polygon;
  PointElement V[];
  int minX, maxX, minY, maxY;
  PointElement P;

  public void init(int numElem, String[] eList, Slate sl, Measure parent) {
      slate = sl;
      numElement = numElem;
      elementList = eList;

      String c = elementList[1];
      polygon = (PolygonElement) slate.lookupElement(elementList[2]);
      parent.addParent(polygon);
      step = slate.GRID_SIZE;

      V = polygon.V;
      n = polygon.n;
      P = new PointElement();

      if (c.equals("r")) {
          choice = 0;
      } // if
      if (c.equals("i")) {
          choice = 1;
      } // if
  }

  public double getValue() {

      // Bestimme minX, minY, maxX und maxY
      minX = Integer.MAX_VALUE;
      minY = Integer.MAX_VALUE;
      maxX = Integer.MIN_VALUE;
      maxY = Integer.MIN_VALUE;

      for (int i=0;i<n;i++) {
          if (V[i].x<minX) {
              minX = (int) Math.round( V[i].x );
          } // if
          if (V[i].y<minY) {
              minY = (int) Math.round( V[i].y );
          } // if
          if (V[i].x>maxX) {
              maxX = (int) Math.round( V[i].x );
          } // if
          if (V[i].y>maxY) {
              maxY = (int) Math.round( V[i].y );
          } // if
      } // for

      switch (choice) {
        case 0:
            numBorder = 0;
            for (int i=minX;i<=maxX;i+=step) {
                for (int j=minY;j<=maxY;j+=step) {
                    P.x = i;
                    P.y = j;

                    // Prüfe, ob P auf dem Rand des Polygons liegt
                    label:
                    for (int k=0;k<n;k++) {
                        if (P.isCollinear(V[k],V[(k+1)%n])&&
                            Math.abs(V[k].distance(P)+V[(k+1)%n].distance(P)-
                            V[k].distance(V[(k+1)%n]))<0.0001) {
                            numBorder++;
```

```
                        break label;
                    } // if
                } // for
            } // for
        } // for
        return (double) numBorder;
      case 1:
        numInside = 0;
        for (int i=minX;i<=maxX;i+=step) {
            for (int j=minY;j<=maxY;j+=step) {
                P.x = i;
                P.y = j;

                // Prüfe, ob P innerhalb des Polygons liegt
                if (!isPointOnBorder(P) && polygon.contains(P)) {
                    numInside++;
                } // if
            } // for
        } // for
        return (double) numInside;
    } // switch

    // should never reach here
    return -1;
  }

  private boolean isPointOnBorder( PointElement P ) {
    // Prüfe, ob P auf dem Rand des Polygons liegt
    for (int k=0;k<n;k++) {
        if (P.isCollinear(V[k],V[(k+1)%n]) && Math.abs(V[k].distance(P)+
            V[(k+1)%n].distance(P)-V[k].distance(V[(k+1)%n]))<0.0001) {
            return true;
        } // if
    } // for
    return false;
  }
}
```

Die Klasse `Functional_Teilverhaeltnis` (Seite 126)

```
public class Functional_Teilverhaeltnis extends Functional {
  PointElement A, B, T;

  public void init(int numElem, String[] eList, Slate sl, Measure parent) {
      slate = sl;
      numElement = numElem;
      elementList = eList;

      A = (PointElement) slate.lookupElement(elementList[1]);
      B = (PointElement) slate.lookupElement(elementList[2]);
      T = (PointElement) slate.lookupElement(elementList[3]);
      parent.addParent(A,B,T);
  }

  public double getValue() {
      // inzidieren A, B oder T miteinander?
      if (((Math.abs(A.x-B.x)<0.00001) && (Math.abs(A.y-B.y)<0.0001)) ||
          ((Math.abs(T.x-B.x)<0.00001) && (Math.abs(T.y-B.y)<0.0001)) ||
          ((Math.abs(A.x-T.x)<0.00001) && (Math.abs(A.y-T.y)<0.0001))) {
          return 1.0/0.0;
      } // if

      double distanceAB = Math.sqrt( (A.x-B.x)*(A.x-B.x) + (A.y-B.y)*(A.y-B.y) );
      double distanceAT = Math.sqrt( (T.x-A.x)*(T.x-A.x) + (T.y-A.y)*(T.y-A.y) );
      double distanceTB = Math.sqrt( (T.x-B.x)*(T.x-B.x) + (T.y-B.y)*(T.y-B.y) );


      if (distanceTB<distanceAB && distanceAT<distanceAB) {
          return distanceAT/distanceTB;
      } // if
      else {
          return -distanceAT/distanceTB;
      } // if
  }
}
```

Die Klasse `Functional_Wuerfelnetz` (Seite 133)

```
import java.lang.Math;

public class Functional_Wuerfelnetz extends Functional {

  PointElement P1, P2, P3, P4, P5, P6;

  public void init(int numElem, String[] eList, Slate sl, Measure parent) {
      slate = sl;
      numElement = numElem;
      elementList = eList;

      P1 = (PointElement) slate.lookupElement(elementList[1]);
      P2 = (PointElement) slate.lookupElement(elementList[2]);
      P3 = (PointElement) slate.lookupElement(elementList[3]);
      P4 = (PointElement) slate.lookupElement(elementList[4]);
      P5 = (PointElement) slate.lookupElement(elementList[5]);
      P6 = (PointElement) slate.lookupElement(elementList[6]);
      parent.addParent(P1,P2,P3);
      parent.addParent(P4,P5,P6);
  }

   public double getValue() {
      //System.out.println("text" + checkWuerfelnetz());
      return (double) checkWuerfelnetz();
   }

   // Bedeutung der Rückgabewerte:
   //  1 := gültiges Würfelnetz
   // -1 := Die sechs Punkte liegen nicht auf einem (ganzzahligen) Quadratraster
   // -2 := Punkte liegen in einem 2x5 Rechteck, bilden aber kein gültiges Würfelnetz
   // -3 := Punkte liegen in einem 3x4 Rechteck, bilden aber kein gültiges Würfelnetz
   // -4 := Punkte liegen in einem Rechteck, bilden aber kein gültiges Würfelnetz
   private int checkWuerfelnetz() {

       double step = 2.0;
       double[][] P = new double[6][2];

       P[0][0] = P1.X_WindowToWorld(P1.x);
       P[0][1] = P1.Y_WindowToWorld(P1.y);
       P[1][0] = P1.X_WindowToWorld(P2.x);
       P[1][1] = P1.Y_WindowToWorld(P2.y);
       P[2][0] = P1.X_WindowToWorld(P3.x);
       P[2][1] = P1.Y_WindowToWorld(P3.y);
       P[3][0] = P1.X_WindowToWorld(P4.x);
       P[3][1] = P1.Y_WindowToWorld(P4.y);
       P[4][0] = P1.X_WindowToWorld(P5.x);
       P[4][1] = P1.Y_WindowToWorld(P5.y);
       P[5][0] = P1.X_WindowToWorld(P6.x);
       P[5][1] = P1.Y_WindowToWorld(P6.y);

       // Kontrolle:
       for (int i=0;i<6;i++) {
           for (int j=0;j<6;j++) {
               //System.out.println(Math.abs(P[i][0]-P[j][0])%step);
               if ((Math.abs(P[i][0]-P[j][0])%step!=0)||(Math.abs(P[i][1]-P[j][1])%step!=0)) {
                   return -1;
               } // if
           } // for
       } // for

       double xMin = Double.MAX_VALUE;
       double yMin = Double.MAX_VALUE;
       double xMax = -1000;
       double yMax = -1000;

       // Bestimme kleinste und größte x- und y-Koordinate
       for (int i=0;i<6;i++) {
           if (P[i][0] < xMin) {
               xMin = P[i][0];
```

```
    } // if
    if (P[i][1] < yMin) {
        yMin = P[i][1];
    } // if
    if (P[i][0] > xMax) {
        xMax = P[i][0];
    } // if
    if (P[i][1] > yMax) {
        yMax = P[i][1];
    } // if
} // for

int dx = (int) Math.round((xMax-xMin)/step);
int dy = (int) Math.round((yMax-yMin)/step);
//System.out.println("dx = " + dx);
//System.out.println("dy = " + dy);

int c = 0;

if (dx>=dy) {
    if (dx==3 && dy==2) {
        c = 1;
    } // if
    if (dx==4 && dy==1) {
        c = 2;
    } // if
} // if


double[][] matrix = new double[dx+1][dy+1];
for (int i=0;i<6;i++) {
    //System.out.print(" i=" + i );
    int mx = (int) Math.round(Math.abs(P[i][0]-xMin)/step);
    int my = (int) Math.round(Math.abs(P[i][1]-yMax)/step);
    //System.out.print(" i=" + i + " matrix[ " + mx + " ][ " + my + " ] = 1" );
    matrix[ mx ][ my ] = 1;
} // for

if (dy>dx) {

    matrix = rotateMatrix(matrix,dx,dy);
    int d = dx;
    dx = dy;
    dy = d;
} // if


if (dx==4 && dy==1) {
    if (checkMatrix2(matrix)) {
       return 1;
    } // if
    else {
       return -2;
    } // else

} // if

if (dx==3 && dy==2) {

    if (checkMatrix(matrix)) {
       return 1;
    } // if

    matrix = mirrorHorizontal(matrix);

    if (checkMatrix(matrix)) {
       return 1;
    } // if

    matrix = mirrorHorizontal(matrix);
    matrix = mirrorVertical(matrix);
```

```
         if (checkMatrix(matrix)) {
            return 1;
         } // if

         matrix = mirrorHorizontal(matrix);

         if (checkMatrix(matrix)) {
            return 1;
         } // if
         return -3;
      } // if


      return -4;
   }

   private static boolean checkMatrix( double[][] m) {

      // 1. Würfelnetz
      // 1 0 0 0
      // 1 1 1 1
      // 1 0 0 0
      if ( m[0][0]==1 && m[1][0]==0 && m[2][0]==0 && m[3][0]==0 &&
           m[0][1]==1 && m[1][1]==1 && m[2][1]==1 && m[3][1]==1 &&
           m[0][2]==1 && m[1][2]==0 && m[2][2]==0 && m[3][2]==0 ) {
         return true;
      } // if

      // 2. Würfelnetz
      // 1 0 0 0
      // 1 1 1 1
      // 0 1 0 0
      if ( m[0][0]==1 && m[1][0]==0 && m[2][0]==0 && m[3][0]==0 &&
           m[0][1]==1 && m[1][1]==1 && m[2][1]==1 && m[3][1]==1 &&
           m[0][2]==0 && m[1][2]==1 && m[2][2]==0 && m[3][2]==0 ) {
         return true;
      } // if

      // 3. Würfelnetz
      // 1 0 0 0
      // 1 1 1 1
      // 0 0 1 0
      if ( m[0][0]==1 && m[1][0]==0 && m[2][0]==0 && m[3][0]==0 &&
           m[0][1]==1 && m[1][1]==1 && m[2][1]==1 && m[3][1]==1 &&
           m[0][2]==0 && m[1][2]==0 && m[2][2]==1 && m[3][2]==0 ) {
         return true;
      } // if

      // 4. Würfelnetz
      // 1 0 0 0
      // 1 1 1 1
      // 0 0 0 1
      if ( m[0][0]==1 && m[1][0]==0 && m[2][0]==0 && m[3][0]==0 &&
           m[0][1]==1 && m[1][1]==1 && m[2][1]==1 && m[3][1]==1 &&
           m[0][2]==0 && m[1][2]==0 && m[2][2]==0 && m[3][2]==1 ) {
         return true;
      } // if

      // 5. Würfelnetz
      // 0 1 0 0
      // 1 1 1 1
      // 0 1 0 0
      if ( m[0][0]==0 && m[1][0]==1 && m[2][0]==0 && m[3][0]==0 &&
           m[0][1]==1 && m[1][1]==1 && m[2][1]==1 && m[3][1]==1 &&
           m[0][2]==0 && m[1][2]==1 && m[2][2]==0 && m[3][2]==0 ) {
         return true;
      } // if

      // 6. Würfelnetz
      // 0 1 0 0
```

```
    // 1 1 1 1
    // 0 0 1 0
    if ( m[0][0]==0 && m[1][0]==1 && m[2][0]==0 && m[3][0]==0 &&
         m[0][1]==1 && m[1][1]==1 && m[2][1]==1 && m[3][1]==1 &&
         m[0][2]==0 && m[1][2]==0 && m[2][2]==1 && m[3][2]==0 ) {
        return true;
    } // if

    // 7. Würfelnetz
    // 1 1 0 0
    // 0 1 1 1
    // 0 1 0 0
    if ( m[0][0]==1 && m[1][0]==1 && m[2][0]==0 && m[3][0]==0 &&
         m[0][1]==0 && m[1][1]==1 && m[2][1]==1 && m[3][1]==1 &&
         m[0][2]==0 && m[1][2]==1 && m[2][2]==0 && m[3][2]==0 ) {
        return true;
    } // if

    // 8. Würfelnetz
    // 1 1 0 0
    // 0 1 1 1
    // 0 0 1 0
    if ( m[0][0]==1 && m[1][0]==1 && m[2][0]==0 && m[3][0]==0 &&
         m[0][1]==0 && m[1][1]==1 && m[2][1]==1 && m[3][1]==1 &&
         m[0][2]==0 && m[1][2]==0 && m[2][2]==1 && m[3][2]==0 ) {
        return true;
    } // if

    // 9. Würfelnetz
    // 1 1 0 0
    // 0 1 1 1
    // 0 0 0 1
    if ( m[0][0]==1 && m[1][0]==1 && m[2][0]==0 && m[3][0]==0 &&
         m[0][1]==0 && m[1][1]==1 && m[2][1]==1 && m[3][1]==1 &&
         m[0][2]==0 && m[1][2]==0 && m[2][2]==0 && m[3][2]==1 ) {
        return true;
    } // if

    // 10. Würfelnetz
    // 1 1 0 0
    // 0 1 1 0
    // 0 0 1 1
    if ( m[0][0]==1 && m[1][0]==1 && m[2][0]==0 && m[3][0]==0 &&
         m[0][1]==0 && m[1][1]==1 && m[2][1]==1 && m[3][1]==0 &&
         m[0][2]==0 && m[1][2]==0 && m[2][2]==1 && m[3][2]==1 ) {
        return true;
    } // if
    return false;
}

private static boolean checkMatrix2( double[][] m) {

    // 1. Würfelnetz
    // 1 1 1 0 0
    // 0 0 1 1 1
    if ( m[0][0]==1 && m[1][0]==1 && m[2][0]==1 && m[3][0]==0 && m[4][0]==0 &&
         m[0][1]==0 && m[1][1]==0 && m[2][1]==1 && m[3][1]==1 && m[4][1]==1) {
        return true;
    } // if

    // 2. Würfelnetz
    // 0 0 1 1 1
    // 1 1 1 0 0
    if ( m[0][0]==0 && m[1][0]==0 && m[2][0]==1 && m[3][0]==1 && m[4][0]==1 &&
         m[0][1]==1 && m[1][1]==1 && m[2][1]==1 && m[3][1]==0 && m[4][1]==0) {
        return true;
    } // if
 return false;
 }

// Voraussetzung m[4][3]!
```

```
  private static double[][] mirrorHorizontal( double[][] m1) {
      double[][] m2 = new double[4][3];
      m2[0][0] = m1[0][2];
      m2[0][1] = m1[0][1];
      m2[0][2] = m1[0][0];

      m2[1][0] = m1[1][2];
      m2[1][1] = m1[1][1];
      m2[1][2] = m1[1][0];

      m2[2][0] = m1[2][2];
      m2[2][1] = m1[2][1];
      m2[2][2] = m1[2][0];

      m2[3][0] = m1[3][2];
      m2[3][1] = m1[3][1];
      m2[3][2] = m1[3][0];

      return m2;
  }

  // Voraussetzung m[4][3]!
  private static double[][] mirrorVertical( double[][] m1) {
      double[][] m2 = new double[4][3];
      m2[0][0] = m1[3][0];
      m2[0][1] = m1[3][1];
      m2[0][2] = m1[3][2];

      m2[1][0] = m1[2][0];
      m2[1][1] = m1[2][1];
      m2[1][2] = m1[2][2];

      m2[2][0] = m1[1][0];
      m2[2][1] = m1[1][1];
      m2[2][2] = m1[1][2];

      m2[3][0] = m1[0][0];
      m2[3][1] = m1[0][1];
      m2[3][2] = m1[0][2];

      return m2;
  }

  private static double[][] rotateMatrix( double[][] m1, int c, int r ) {

      double[][] m2 = new double[r+1][c+1];

      for (int i=0;i<=r;i++) {
          for (int j=0;j<=c;j++) {
              m2[i][j] = m1[j][i];
          } // for
      } // for
      return m2;
  }

}
```

Die Klasse `Functional_Funktionsparameter` (Seite 150)

```
public class Functional_Funktionsparameter extends Functional {

  PointElement P1,P2,P3;
  double P1x,P1y,P2x,P2y,P3x,P3y;
  int type;

  public void init(int numElem, String[] eList, Slate sl, Measure parent) {
      slate = sl;
      numElement = numElem;
      elementList = eList;
      P1 = (PointElement) slate.lookupElement(elementList[1]);
      P2 = (PointElement) slate.lookupElement(elementList[2]);
      P3 = (PointElement) slate.lookupElement(elementList[3]);
      parent.addParent(P1,P2,P3);
      type = -1;
      if (elementList[4].equals("a")) {
          type = 0;
      } // if
      if (elementList[4].equals("b")) {
          type = 1;
      } // if
      if (elementList[4].equals("c")) {
          type = 2;
      } // if
  }

  public double getValue() {

      P1x = P1.X_WindowToWorld(P1.x);
      P1y = P1.Y_WindowToWorld(P1.y);
      P2x = P1.X_WindowToWorld(P2.x);
      P2y = P1.Y_WindowToWorld(P2.y);
      P3x = P1.X_WindowToWorld(P3.x);
      P3y = P1.Y_WindowToWorld(P3.y);

      switch (type) {
          case 0:
              return (P2y*P2x*P1x-P2y*P2x*P3x+P1y*P1x*P3x-
                      P1x*P3y*P3x+P2x*P3y*P3x-P2x*P1y*P1x)/
                     (P2x*P2x*P1x-P2x*P2x*P3x+P1x*P1x*P3x-
                      P1x*P3x*P3x+P2x*P3x*P3x-P2x*P1x*P1x);
          case 1:
              return (-P2x*P1y+P2x*P3y+P2y*P1x-P1x*P3y-P2y*P3x+P1y*P3x)*P3x*P2x*P1x/
                     (P1x-P3x)/(P1x*P3x-P2x*P1x+P2x*P2x-P2x*P3x);
          case 2:
              return -(P1x*P1x*P2y*P2x-P3y*P3x*P1x*P1x-P1x*P1y*P2x*P2x+P1y*P1x*P3x*P3x-
                       P3x*P3x*P2y*P2x+P3y*P3x*P2x*P2x)/
                      ((P1x-P3x)*(P1x*P3x-P2x*P1x+P2x*P2x-P2x*P3x));
       } // switch

      return Double.NaN;
  }

}
```

Die Klasse `Curve_Funktionsgraph` (Seite 150)

```
public class Curve_Funktionsgraph extends Curve {

  PointElement P1,P2,P3;
  double P1x,P1y,P2x,P2y,P3x,P3y,x0,x1,a,b,c;
  int num;
  double[] param_t;

  public void init(int numElem, String[] eList, Slate sl, CurveElement parent) {
      slate = sl;
      numElement = numElem;
      elementList = eList;
      P1 = (PointElement) slate.lookupElement(elementList[1]);
      P2 = (PointElement) slate.lookupElement(elementList[2]);
      P3 = (PointElement) slate.lookupElement(elementList[3]);
      x0 = MathFunc.grepDouble(elementList[4]);
      x1 = MathFunc.grepDouble(elementList[5]);
      num = Integer.parseInt(elementList[6]);
      parent.addParent(P1,P2,P3);

      double delta = (x1-x0)/(num-1);
      param_t = new double[num];
      for (int i=0;i<num;i++) {
          param_t[i] = x0 + ( i*delta );
      } // for
  }

  public void update() {

      // Gegeben sind drei Punkte P1, P2, P3
      P1x = P1.X_WindowToWorld(P1.x);
      P1y = P1.Y_WindowToWorld(P1.y);
      P2x = P1.X_WindowToWorld(P2.x);
      P2y = P1.Y_WindowToWorld(P2.y);
      P3x = P1.X_WindowToWorld(P3.x);
      P3y = P1.Y_WindowToWorld(P3.y);

      // Berechnen der Kurvenparameter
      a = (P2y*P2x*P1x-P2y*P2x*P3x+P1y*P1x*P3x-P1x*P3y*P3x+P2x*P3y*P3x-P2x*P1y*P1x)/
          (P2x*P2x*P1x-P2x*P2x*P3x+P1x*P1x*P3x-P1x*P3x*P3x+P2x*P3x*P3x-P2x*P1x*P1x);
      b = (-P2x*P1y+P2x*P3y+P2y*P1x-P1x*P3y-P2y*P3x+P1y*P3x)*P3x*P2x*P1x/(P1x-P3x)/
          (P1x*P3x-P2x*P1x+P2x*P2x-P2x*P3x);
      c = -(P1x*P1x*P2y*P2x-P3y*P3x*P1x*P1x-P1x*P1y*P2x*P2x+P1y*P1x*P3x*P3x-P3x*P3x*P2y*P2x+
            P3y*P3x*P2x*P2x)/((P1x-P3x)*(P1x*P3x-P2x*P1x+P2x*P2x-P2x*P3x));

      numInterval = 1;
      numPoints = new int[1];
      numPoints[0] = num;
      xPoint = new int[1][num];
      yPoint = new int[1][num];

      for (int i=0;i<num;i++) {
          xPoint[0][i] = (int) P1.X_WorldToWindow(param_t[i]);
          yPoint[0][i] = (int) P1.Y_WorldToWindow((a*param_t[i])+(b/param_t[i])+c);
      } // for
  }
}
```

Die Klasse `Curve_Algebraic` (Seite 154)

```
public class Curve_Algebraic extends Curve {

  Measure ma,mb;
  double x0, x1, delta, a, b, t, s, x2;
  double[] x;
  int num;

  public void init(int numElem, String[] eList, Slate sl, CurveElement parent) {
      slate = sl;
      numElement = numElem;
      elementList = eList;
      ma = (Measure) slate.lookupElement(elementList[1]);
      mb = (Measure) slate.lookupElement(elementList[2]);
      parent.addParent(ma,mb);
      numInterval = 2;
      numPoints = new int[3];
      numPoints[0] = Integer.parseInt(elementList[3]);
      numPoints[1] = Integer.parseInt(elementList[3]);
      numPoints[2] = 2;
      num = Integer.parseInt(elementList[3]);
      xPoint = new int[3][numPoints[0]];
      yPoint = new int[3][numPoints[0]];

      x0 = MathFunc.grepDouble(elementList[4]);
      x1 = MathFunc.grepDouble(elementList[5]);
      delta = (x1-x0)/(numPoints[0]-1);
      x = new double[numPoints[0]];
      for (int i=0;i<numPoints[0];i++) {
          x[i] = x0 + ( i*delta );
      } // for
  }

  public void update() {
      a = ma.getValue();
      b = mb.getValue();
      numPoints[0] = num;
      numPoints[1] = num;

      // 1. Fall: a>=0 und b>=0
      if (a>=0 && b>=0) {
          numInterval = 2;
          for (int i=0;i<numPoints[0];i++) {
              xPoint[0][i] = (int) slate.X_WorldToWindow(x[i]);
              xPoint[1][i] = (int) slate.X_WorldToWindow(x[i]);
              yPoint[0][i] = (int) slate.Y_WorldToWindow( Math.sqrt(a*x[i]*x[i]*x[i]*x[i] +
                             b*x[i]*x[i]));
              yPoint[1][i] = (int) slate.Y_WorldToWindow(-Math.sqrt(a*x[i]*x[i]*x[i]*x[i] +
                             b*x[i]*x[i]));
          } // for
          return;
      } // if

      // 2. Fall: a>0 und b<0
      if (a>0 && b<0) {
          numInterval = 3;
          x2 = Math.sqrt(-(b/a))+0.000001;
          delta = (x1-x2)/((num/2)-1);

          for (int i=0;i<(num/2);i++) {
              t = x1 - i*delta;
              xPoint[0][i] = (int) slate.X_WorldToWindow(t);
              yPoint[0][i] = (int) slate.Y_WorldToWindow(Math.sqrt(a*t*t*t*t + b*t*t));
              s = -t;
              xPoint[1][i] = (int) slate.X_WorldToWindow(s);
              yPoint[1][i] = (int) slate.Y_WorldToWindow(Math.sqrt(a*s*s*s*s + b*s*s));
          } // for

          for (int i=(num/2);i<num;i++) {
              t = x2 + (i-(num/2))*delta;
```

```
            xPoint[0][i] = (int) slate.X_WorldToWindow(t);
            yPoint[0][i] = (int) slate.Y_WorldToWindow(-Math.sqrt(a*t*t*t*t + b*t*t));
            s = -t;
            xPoint[1][i] = (int) slate.X_WorldToWindow(s);
            yPoint[1][i] = (int) slate.Y_WorldToWindow(-Math.sqrt(a*s*s*s*s + b*s*s));
        } // for
        xPoint[2][0] = (int) slate.X_WorldToWindow(0.0);
        xPoint[2][1] = (int) slate.X_WorldToWindow(0.0);
        yPoint[2][0] = (int) slate.Y_WorldToWindow(0.0);
        yPoint[2][1] = (int) slate.Y_WorldToWindow(0.0);
        return;
    } // if

    // 3. Fall: a<0 und b=0
    if (a<0 && b==0) {
        numInterval = 1;
        numPoints[0] = 2;
        xPoint[0][0] = (int) slate.X_WorldToWindow(0.0);
        xPoint[0][1] = (int) slate.X_WorldToWindow(0.0);
        yPoint[0][0] = (int) slate.Y_WorldToWindow(0.0);
        yPoint[0][1] = (int) slate.Y_WorldToWindow(0.0);
        return;
    } // if

    // 4. Fall: a<0 und b>0
    if (a<0 && b>0) {
        numInterval = 2;
        x2 = Math.sqrt(-(b/a))-0.000001;
        delta = (x2)/((num/2)-1);

        for (int i=0;i<(num/2);i++) {
            t = x2 - i*delta;
            xPoint[0][i] = (int) slate.X_WorldToWindow(t);
            yPoint[0][i] = (int) slate.Y_WorldToWindow(Math.sqrt(a*t*t*t*t + b*t*t));
            s = -t;
            xPoint[1][i] = (int) slate.X_WorldToWindow(s);
            yPoint[1][i] = (int) slate.Y_WorldToWindow(Math.sqrt(a*s*s*s*s + b*s*s));
        } // for

        for (int i=(num/2);i<num;i++) {
            t = (i-(num/2))*delta;
            xPoint[0][i] = (int) slate.X_WorldToWindow(t);
            yPoint[0][i] = (int) slate.Y_WorldToWindow(-Math.sqrt(a*t*t*t*t + b*t*t));
            s = -t;
            xPoint[1][i] = (int) slate.X_WorldToWindow(s);
            yPoint[1][i] = (int) slate.Y_WorldToWindow(-Math.sqrt(a*s*s*s*s + b*s*s));
        } // for
        return;
    } // if
    numInterval = 0;
  }

}
```

Die Klasse `Functional_Integral` (Seite 154)

```
import JSci.maths.*;
import JSci.io.*;
import java.io.*;

public class Functional_Integral extends Functional implements Mapping {

  private int N, choice;
  private Measure a2, a1, a0, a, b;
  private CurveElement function;

  public void init(int numElem, String[] eList, Slate sl, Measure parent) {
      slate = sl;
      numElement = numElem;
      elementList = eList;

      function = (CurveElement) slate.lookupElement(elementList[1]);
      a  = (Measure) slate.lookupElement(elementList[2]);
      b  = (Measure) slate.lookupElement(elementList[3]);
      N  = MathFunc.grepInt(elementList[4]);
      String method = elementList[5];

      parent.addParent(function, a, b);

      if (method.toLowerCase().equals("trapezium")) {
          choice = 0;
      } // if
      if (method.toLowerCase().equals("simpson")) {
          choice = 1;
      } // if
      if (method.toLowerCase().equals("richardson")) {
          choice = 2;
      } // if
      if (method.toLowerCase().equals("gaussian4")) {
          choice = 3;
      } // if
      if (method.toLowerCase().equals("gaussian8")) {
          choice = 4;
      } // if
  }

 public double getValue() {
    switch (choice) {
        case 0:
            return NumericalMath.trapezium(N,this,a.getValue(),b.getValue());
        case 1:
            return NumericalMath.simpson(N,this,a.getValue(),b.getValue());
        case 2:
            return NumericalMath.richardson(N,this,a.getValue(),b.getValue());
        case 3:
            return NumericalMath.gaussian4(N,this,a.getValue(),b.getValue());
        case 4:
            return NumericalMath.gaussian8(N,this,a.getValue(),b.getValue());
    } // switch

  return -1;
 }

 public double map(double x) {
     return function.getFunctionValue(x);
 }

 public Complex map(Complex z) {
     return null;
 }
}
```

Die Klasse `Curve_Kochkurve` (Seite 163)

```
public class Curve_Kochkurve extends Curve {

  PointElement A,B;
  Measure m;
  int stufe;
  double r;
  double[] xlinks;
  double[] ylinks;
  double[] xrechts;
  double[] yrechts;


  public void init(int numElem, String[] eList, Slate sl, CurveElement parent) {
      slate = sl;
      numElement = numElem;
      elementList = eList;
      A = (PointElement) slate.lookupElement(elementList[1]);
      B = (PointElement) slate.lookupElement(elementList[2]);
      m = (Measure) slate.lookupElement(elementList[3]);
      parent.addParent(A,B,m);
  }

  public void update() {
      stufe = (int)Math.round(m.getValue());
      if (stufe < 1) {
          stufe = 1;
      } // if
      if (stufe > 7) {
          stufe = 7;
      } // if
      int p = (int)Math.round(Math.pow(4,stufe));
      numInterval = 0;
      numPoints = new int[p];
      numPoints[0] = 0;

      xPoint = new int[p][2];
      yPoint = new int[p][2];
      xlinks = new double[10];
      ylinks = new double[10];
      xrechts = new double[10];
      yrechts = new double[10];

      r = 0.29;
      xlinks[stufe] = A.x;
      ylinks[stufe] = A.y;
      xrechts[stufe] = B.x;
      yrechts[stufe] = B.y;

      subPrg1();
      return;
  }

  private void subPrg1() {

      // Zeichne eine Linie auf der niedrigsten Stufe der Rekursion
      if (stufe > 1) {
          subPrg2();
      } // if
      else {
          xPoint[numInterval][0] = (int) Math.round(xlinks[1]);
          yPoint[numInterval][0] = (int) Math.round(ylinks[1]);
          xPoint[numInterval][1] = (int) Math.round(xrechts[1]);
          yPoint[numInterval][1] = (int) Math.round(yrechts[1]);
          numPoints[numInterval] = 2;
          numInterval++;
      } // else
  }

  private void subPrg2() {
```

```
      // Verzweige in niedrigere Stufe
      stufe--;

      // Linker Zweig
      xlinks[stufe] = xlinks[stufe+1];
      ylinks[stufe] = ylinks[stufe+1];
      xrechts[stufe] = 0.333*xrechts[stufe+1] + 0.667*xlinks[stufe+1];
      yrechts[stufe] = 0.333*yrechts[stufe+1] + 0.667*ylinks[stufe+1];
      subPrg1();

      // Mittlerer linker Zweig
      xlinks[stufe] = xrechts[stufe];
      ylinks[stufe] = yrechts[stufe];
      xrechts[stufe] = 0.5*xrechts[stufe+1] + 0.5*xlinks[stufe+1] -
                       r*(ylinks[stufe+1]-yrechts[stufe+1]);
      yrechts[stufe] = 0.5*yrechts[stufe+1] + 0.5*ylinks[stufe+1] +
                       r*(xlinks[stufe+1]-xrechts[stufe+1]);
      subPrg1();

      // Mittlerer rechter Zweig
      xlinks[stufe] = xrechts[stufe];
      ylinks[stufe] = yrechts[stufe];
      xrechts[stufe] = 0.667*xrechts[stufe+1] + 0.333*xlinks[stufe+1];
      yrechts[stufe] = 0.667*yrechts[stufe+1] + 0.333*ylinks[stufe+1];
      subPrg1();

      // Rechter Zweig
      xlinks[stufe] = xrechts[stufe];
      ylinks[stufe] = yrechts[stufe];
      xrechts[stufe] = xrechts[stufe+1];
      yrechts[stufe] = yrechts[stufe+1];
      subPrg1();

      stufe++;
  }
}
```

Die Klasse `Curve_Sierpinski` (Seite 163)

```
public class Curve_Sierpinski extends Curve {

  PointElement A,B,C;
  Measure m;
  int stufe;
  double r, step;
  double[] xlinks;
  double[] ylinks;
  double[] xrechts;
  double[] yrechts;
  double[] xoben;
  double[] yoben;


  public void init(int numElem, String[] eList, Slate sl, CurveElement parent) {
      slate = sl;
      numElement = numElem;
      elementList = eList;
      A = (PointElement) slate.lookupElement(elementList[1]);
      B = (PointElement) slate.lookupElement(elementList[2]);
      C = (PointElement) slate.lookupElement(elementList[3]);
      m = (Measure) slate.lookupElement(elementList[4]);
      parent.addParent(A,B,C,m);
  }

  public void update() {
      stufe = (int)Math.round(m.getValue());
      if (stufe < 1) {
          stufe = 1;
      } // if
      if (stufe > 7) {
          stufe = 7;
      } // if
      int p = (int)Math.round(Math.pow(4,stufe));
      numInterval = 0;
      numPoints = new int[p];
      numPoints[0] = 0;

      xPoint = new int[p][2];
      yPoint = new int[p][2];
      xlinks = new double[10];
      ylinks = new double[10];
      xrechts = new double[10];
      yrechts = new double[10];
      xoben = new double[10];
      yoben = new double[10];

      r = 0.29;
      xlinks[stufe] = A.x;
      ylinks[stufe] = A.y;
      xrechts[stufe] = B.x;
      yrechts[stufe] = B.y;
      xoben[stufe] = C.x;
      yoben[stufe] = C.y;
      subPrg1();
      return;
  }

  private void subPrg1() {

      // Zeichne eine Linie auf der niedrigsten Stufe der Rekursion
      if (stufe > 1) {
          subPrg2();
      } // if
      else {
          xPoint[numInterval][0] = (int) Math.round(xlinks[1]);
          yPoint[numInterval][0] = (int) Math.round(ylinks[1]);
          xPoint[numInterval][1] = (int) Math.round(xrechts[1]);
          yPoint[numInterval][1] = (int) Math.round(yrechts[1]);
```

```
            numPoints[numInterval] = 2;
            numInterval++;

            xPoint[numInterval][0] = (int) Math.round(xrechts[1]);
            yPoint[numInterval][0] = (int) Math.round(yrechts[1]);
            xPoint[numInterval][1] = (int) Math.round(xoben[1]);
            yPoint[numInterval][1] = (int) Math.round(yoben[1]);
            numPoints[numInterval] = 2;
            numInterval++;

            xPoint[numInterval][0] = (int) Math.round(xoben[1]);
            yPoint[numInterval][0] = (int) Math.round(yoben[1]);
            xPoint[numInterval][1] = (int) Math.round(xlinks[1]);
            yPoint[numInterval][1] = (int) Math.round(ylinks[1]);
            numPoints[numInterval] = 2;
            numInterval++;

        } // else
    }

    private void subPrg2() {
        // Verzweige in niedrigere Stufe
        stufe--;

        // Linkes unteres Dreieck
        xlinks[stufe] = xlinks[stufe+1];
        ylinks[stufe] = ylinks[stufe+1];
        xrechts[stufe] = xlinks[stufe+1] + 0.5*(xrechts[stufe+1]-xlinks[stufe+1]);
        yrechts[stufe] = ylinks[stufe+1] + 0.5*(yrechts[stufe+1]-ylinks[stufe+1]);
        xoben[stufe] = xlinks[stufe+1] + 0.5*(xoben[stufe+1]-xlinks[stufe+1]);
        yoben[stufe] = ylinks[stufe+1] + 0.5*(yoben[stufe+1]-ylinks[stufe+1]);
        subPrg1();

        // Rechtes unteres Dreieck
        xlinks[stufe] = xlinks[stufe+1] + 0.5*(xrechts[stufe+1]-xlinks[stufe+1]);
        ylinks[stufe] = ylinks[stufe+1] + 0.5*(yrechts[stufe+1]-ylinks[stufe+1]);
        xrechts[stufe] = xrechts[stufe+1];
        yrechts[stufe] = yrechts[stufe+1];
        xoben[stufe] = xrechts[stufe+1] + 0.5*(xoben[stufe+1]-xrechts[stufe+1]);
        yoben[stufe] = yrechts[stufe+1] + 0.5*(yoben[stufe+1]-yrechts[stufe+1]);
        subPrg1();

        // Oberes Dreieck
        xlinks[stufe] = xlinks[stufe+1] + 0.5*(xoben[stufe+1]-xlinks[stufe+1]);
        ylinks[stufe] = ylinks[stufe+1] + 0.5*(yoben[stufe+1]-ylinks[stufe+1]);
        xrechts[stufe] = xrechts[stufe+1] + 0.5*(xoben[stufe+1]-xrechts[stufe+1]);
        yrechts[stufe] = yrechts[stufe+1] + 0.5*(yoben[stufe+1]-yrechts[stufe+1]);
        xoben[stufe] = xoben[stufe+1];
        yoben[stufe] = yoben[stufe+1];
        subPrg1();

        stufe++;
    }
}
```

Die Klasse `Curve_Baeume` (Seite 163)

```
public class Curve_Baeume extends Curve {

  PointElement A,B,C,D,E;
  Measure m;
  int stufe;
  double alpha, step, beta, gamma, k1, k2, k3;
  double[] Ax,Ay;
  double[] Bx,By;
  double[] Cx,Cy;
  double[] Dx,Dy;
  double[] Ex,Ey;

  public void init(int numElem, String[] eList, Slate sl, CurveElement parent) {
      slate = sl;
      numElement = numElem;
      elementList = eList;
      A = (PointElement) slate.lookupElement(elementList[1]);
      B = (PointElement) slate.lookupElement(elementList[2]);
      C = (PointElement) slate.lookupElement(elementList[3]);
      D = (PointElement) slate.lookupElement(elementList[4]);
      E = (PointElement) slate.lookupElement(elementList[5]);
      m = (Measure) slate.lookupElement(elementList[6]);
      parent.addParent(A,B,C);
      parent.addParent(D,E,m);
  }

  public void update() {
      stufe = (int)Math.round(m.getValue());
      if (stufe < 1) {
          stufe = 1;
      } // if
      if (stufe > 7) {
          stufe = 7;
      } // if
      int p = (int)Math.round(Math.pow(5,stufe));
      numInterval = 0;
      numPoints = new int[p];
      numPoints[0] = 0;

      xPoint = new int[p][2];
      yPoint = new int[p][2];
      Ax = new double[10];
      Ay = new double[10];
      Bx = new double[10];
      By = new double[10];
      Cx = new double[10];
      Cy = new double[10];
      Dx = new double[10];
      Dy = new double[10];
      Ex = new double[10];
      Ey = new double[10];

      alpha = E.angle2D(A,B);
      beta = E.angle2D(A,C);
      gamma = E.angle2D(A,D);

      k1 = B.distance(E) / A.distance(E);
      k2 = C.distance(E) / A.distance(E);
      k3 = D.distance(E) / A.distance(E);

      Ax[stufe] = A.x;
      Ay[stufe] = A.y;
      Bx[stufe] = B.x;
      By[stufe] = B.y;
      Cx[stufe] = C.x;
      Cy[stufe] = C.y;
      Dx[stufe] = D.x;
      Dy[stufe] = D.y;
      Ex[stufe] = E.x;
```

```
    Ey[stufe] = E.y;
    subPrg1();
    return;
}

private void subPrg1() {

    // Zeichne eine Linie auf der niedrigsten Stufe der Rekursion
    if (stufe > 1) {
        subPrg2();
    } // if
    else {
        xPoint[numInterval][0] = (int) Math.round(Ax[1]);
        yPoint[numInterval][0] = (int) Math.round(Ay[1]);
        xPoint[numInterval][1] = (int) Math.round(Ex[1]);
        yPoint[numInterval][1] = (int) Math.round(Ey[1]);
        numPoints[numInterval] = 2;
        numInterval++;

        xPoint[numInterval][0] = (int) Math.round(Bx[1]);
        yPoint[numInterval][0] = (int) Math.round(By[1]);
        xPoint[numInterval][1] = (int) Math.round(Ex[1]);
        yPoint[numInterval][1] = (int) Math.round(Ey[1]);
        numPoints[numInterval] = 2;
        numInterval++;

        xPoint[numInterval][0] = (int) Math.round(Cx[1]);
        yPoint[numInterval][0] = (int) Math.round(Cy[1]);
        xPoint[numInterval][1] = (int) Math.round(Ex[1]);
        yPoint[numInterval][1] = (int) Math.round(Ey[1]);
        numPoints[numInterval] = 2;
        numInterval++;

        xPoint[numInterval][0] = (int) Math.round(Dx[1]);
        yPoint[numInterval][0] = (int) Math.round(Dy[1]);
        xPoint[numInterval][1] = (int) Math.round(Ex[1]);
        yPoint[numInterval][1] = (int) Math.round(Ey[1]);
        numPoints[numInterval] = 2;
        numInterval++;

    } // else
}

private void subPrg2() {
    // Verzweige in niedrigere Stufe
    stufe--;

     // Linker Ast
    Ax[stufe] = Ex[stufe+1];
    Ay[stufe] = Ey[stufe+1];
    Bx[stufe] = Dx[stufe+1] + k1*Math.cos(alpha)*(Ex[stufe+1]-Dx[stufe+1]) -
                k1*Math.sin(alpha)*(Ey[stufe+1]-Dy[stufe+1]);
    By[stufe] = Dy[stufe+1] + k1*Math.sin(alpha)*(Ex[stufe+1]-Dx[stufe+1]) +
                k1*Math.cos(alpha)*(Ey[stufe+1]-Dy[stufe+1]);
    Cx[stufe] = Dx[stufe+1] + k2*Math.cos(beta)*(Ex[stufe+1]-Dx[stufe+1]) -
                k2*Math.sin(beta)*(Ey[stufe+1]-Dy[stufe+1]);
    Cy[stufe] = Dy[stufe+1] + k2*Math.sin(beta)*(Ex[stufe+1]-Dx[stufe+1]) +
                k2*Math.cos(beta)*(Ey[stufe+1]-Dy[stufe+1]);
    Dx[stufe] = Dx[stufe+1] + k3*Math.cos(gamma)*(Ex[stufe+1]-Dx[stufe+1]) -
                k3*Math.sin(gamma)*(Ey[stufe+1]-Dy[stufe+1]);
    Dy[stufe] = Dy[stufe+1] + k3*Math.sin(gamma)*(Ex[stufe+1]-Dx[stufe+1]) +
                k3*Math.cos(gamma)*(Ey[stufe+1]-Dy[stufe+1]);
    Ex[stufe] = Dx[stufe+1];
    Ey[stufe] = Dy[stufe+1];
    subPrg1();

    // Mittlerer Ast
    Ax[stufe] = Ex[stufe+1];
    Ay[stufe] = Ey[stufe+1];
    Bx[stufe] = Cx[stufe+1] + k1*Math.cos(alpha)*(Ex[stufe+1]-Cx[stufe+1]) -
                k1*Math.sin(alpha)*(Ey[stufe+1]-Cy[stufe+1]);
```

```
      By[stufe] = Cy[stufe+1] + k1*Math.sin(alpha)*(Ex[stufe+1]-Cx[stufe+1]) +
                  k1*Math.cos(alpha)*(Ey[stufe+1]-Cy[stufe+1]);
      Cx[stufe] = Cx[stufe+1] + k2*Math.cos(beta)*(Ex[stufe+1]-Cx[stufe+1]) -
                  k2*Math.sin(beta)*(Ey[stufe+1]-Cy[stufe+1]);
      Cy[stufe] = Cy[stufe+1] + k2*Math.sin(beta)*(Ex[stufe+1]-Cx[stufe+1]) +
                  k2*Math.cos(beta)*(Ey[stufe+1]-Cy[stufe+1]);
      Dx[stufe] = Cx[stufe+1] + k3*Math.cos(gamma)*(Ex[stufe+1]-Cx[stufe+1]) -
                  k3*Math.sin(gamma)*(Ey[stufe+1]-Cy[stufe+1]);
      Dy[stufe] = Cy[stufe+1] + k3*Math.sin(gamma)*(Ex[stufe+1]-Cx[stufe+1]) +
                  k3*Math.cos(gamma)*(Ey[stufe+1]-Cy[stufe+1]);
      Ex[stufe] = Cx[stufe+1];
      Ey[stufe] = Cy[stufe+1];
      subPrg1();

      // Rechter Ast
      Ax[stufe] = Ex[stufe+1];
      Ay[stufe] = Ey[stufe+1];
      Bx[stufe] = Bx[stufe+1] + k1*Math.cos(alpha)*(Ex[stufe+1]-Bx[stufe+1]) -
                  k1*Math.sin(alpha)*(Ey[stufe+1]-By[stufe+1]);
      By[stufe] = By[stufe+1] + k1*Math.sin(alpha)*(Ex[stufe+1]-Bx[stufe+1]) +
                  k1*Math.cos(alpha)*(Ey[stufe+1]-By[stufe+1]);
      Cx[stufe] = Bx[stufe+1] + k2*Math.cos(beta)*(Ex[stufe+1]-Bx[stufe+1]) -
                  k2*Math.sin(beta)*(Ey[stufe+1]-By[stufe+1]);
      Cy[stufe] = By[stufe+1] + k2*Math.sin(beta)*(Ex[stufe+1]-Bx[stufe+1]) +
                  k2*Math.cos(beta)*(Ey[stufe+1]-By[stufe+1]);
      Dx[stufe] = Bx[stufe+1] + k3*Math.cos(gamma)*(Ex[stufe+1]-Bx[stufe+1]) -
                  k3*Math.sin(gamma)*(Ey[stufe+1]-By[stufe+1]);
      Dy[stufe] = By[stufe+1] + k3*Math.sin(gamma)*(Ex[stufe+1]-Bx[stufe+1]) +
                  k3*Math.cos(gamma)*(Ey[stufe+1]-By[stufe+1]);
      Ex[stufe] = Bx[stufe+1];
      Ey[stufe] = By[stufe+1];
      subPrg1();

      xPoint[numInterval][0] = (int) Math.round(Ax[stufe+1]);
      yPoint[numInterval][0] = (int) Math.round(Ay[stufe+1]);
      xPoint[numInterval][1] = (int) Math.round(Ex[stufe+1]);
      yPoint[numInterval][1] = (int) Math.round(Ey[stufe+1]);
      numPoints[numInterval] = 2;
      numInterval++;

      stufe++;
    }
}
```

Die Klasse `Functional_EightQueens` (Seite 174)

```
public class Functional_EightQueens extends Functional {

  PointElement[] P;
  boolean[] a;
  boolean[] b;
  boolean[] diaR;
  boolean[] diaL;
  int[] zugx;
  int[] zugy;
  int zugnr;
  int xCoord, yCoord;
  int[] loesx;
  boolean[] Lb;
  boolean[] Lc;
  boolean[] Ld;
  boolean eine_loesung, working = false;
  String choice;

  public void init(int numElem, String[] eList, Slate sl, Measure parent) {
      slate = sl;
      numElement = numElem;
      elementList = eList;

      P = new PointElement[8];
      P[0] = (PointElement) slate.lookupElement(elementList[1]);
      P[1] = (PointElement) slate.lookupElement(elementList[2]);
      P[2] = (PointElement) slate.lookupElement(elementList[3]);
      P[3] = (PointElement) slate.lookupElement(elementList[4]);
      P[4] = (PointElement) slate.lookupElement(elementList[5]);
      P[5] = (PointElement) slate.lookupElement(elementList[6]);
      P[6] = (PointElement) slate.lookupElement(elementList[7]);
      P[7] = (PointElement) slate.lookupElement(elementList[8]);
      parent.addParent(P[0],P[1],P[2],P[3]);
      parent.addParent(P[4],P[5],P[6],P[7]);

      choice = elementList[9];
      a = new boolean[8];
      b = new boolean[8];
      diaR = new boolean[16];
      diaL = new boolean[16];
      zugx = new int[8];
      zugy = new int[8];

  }

  public double getValue() {

      /*
         Bedeutungen der Rückgabewerte:

         0 - Es ist noch keine Dame auf dem Spielfeld plaziert.

         1 - Ungültiger Zug!
             Mind. eine Dame ist nicht eindeutig auf dem Spielfeld plaziert.

         2 - Ungültiger Zug!
             Mind. zwei Damen bedrohen sich gegenseitig.

         3 - Gültige Brettkonstellation

         4 - Gültiger Zug!
             Es gibt aber keine Lösungsmöglichkeit mehr.

         5 - Gültiger Zug und
             es gibt noch mind. eine Lösungsmöglichkeit.

         6 - Gültiger Zug und
             Lösung wurde gefunden.
```

```
    */
    if (working) {
        return -1;
    } // if

    int v = checkPositionOnBoard();

    //print();
    if (choice.toLowerCase().equals("analyse")) {
       if (v==3 && zugnr==8) {
           return 6.0;
       } // if
       if (v<=2) {
           return (double) v;
       } // if
    } // if

    if (!working && v>=2) {
        v = forceMove();
    } // if

    if (choice.toLowerCase().equals("analyse")) {
       return (double) v;
    } // if

    if (choice.toLowerCase().equals("hint_x") && v==5) {
       return (double) xCoord + 1;
    } // if

    if (choice.toLowerCase().equals("hint_y") && v==5) {
       return (double) yCoord + 1;
    } // if

    return -1.0;
}

private int checkPositionOnBoard() {
    zugnr = 0;
    int x, y, val = -1;
    double px, py;
    zugx = new int[8];
    zugy = new int[8];
    a = new boolean[8];
    b = new boolean[8];
    diaR = new boolean[16];
    diaL = new boolean[16];

    label:
    for (int i=0;i<8;i++) {
        px = slate.X_WindowToWorld(Math.round(P[i].x));
        py = slate.Y_WindowToWorld(Math.round(P[i].y));

        // Liegt P[i] im Spielfeld?
        if ((Math.abs(px) < 8) && (Math.abs(py) < 8)) {
            x = -1;
            y = -1;

            if (Math.abs(px+7) < 0.000001) {
                x = 0;
            } // if
            else {
                if (Math.abs(px+5) < 0.000001) {
                    x = 1;
                } // if
                else {
                    if (Math.abs(px+3) < 0.000001) {
                        x = 2;
                    } // if
                    else {
                        if (Math.abs(px+1) < 0.000001) {
                            x = 3;
```

```
            } // if
            else  {
                if (Math.abs(px-1) < 0.000001) {
                    x = 4;
                } // if
                else {
                    if (Math.abs(px-3) < 0.000001) {
                        x = 5;
                    } // if
                    else {
                        if (Math.abs(px-5) < 0.000001) {
                            x = 6;
                        } // if
                        else {
                            if (Math.abs(px-7) < 0.000001) {
                                x = 7;
                            } // if
                        } // else
                    } // else
                } // else
            } // else
        } // else
    } // else
} // else

if (Math.abs(py+7) < 0.000001) {
    y = 0;
} // if
else {
    if (Math.abs(py+5) < 0.000001) {
        y = 1;
    } // if
    else {
        if (Math.abs(py+3) < 0.000001) {
            y = 2;
        } // if
        else {
            if (Math.abs(py+1) < 0.000001) {
                y = 3;
            } // if
            else  {
                if (Math.abs(py-1) < 0.000001) {
                    y = 4;
                } // if
                else {
                    if (Math.abs(py-3) < 0.000001) {
                        y = 5;
                    } // if
                    else {
                        if (Math.abs(py-5) < 0.000001) {
                            y = 6;
                        } // if
                        else {
                            if (Math.abs(py-7) < 0.000001) {
                                y = 7;
                            } // if
                        } // else
                    } // else
                } // else
            } // else
        } // else
    } // else
} // else

if (x!=-1 && y!=-1) {

    if (!(b[y] || a[x] || diaL[x+y] || diaR[y-x+7])) {
        zugx[zugnr] = x;
        zugy[zugnr] = y;
        zugnr++;
        a[x] = true;
```

```
                    b[y] = true;
                    diaL[x+y] = true;
                    diaR[y-x+7] = true;
                    // 3 - Gültige Brettkonstellation
                    val = 3;
                } // if
                else {
                    zugx[zugnr] = x;
                    zugy[zugnr] = y;
                    zugnr++;
                    a[x] = true;
                    b[y] = true;
                    diaL[x+y] = true;
                    diaR[y-x+7] = true;
                    // 2 - Ungültiger Zug!
                    // Mind. zwei Damen bedrohen sich gegenseitig.
                    val = 2;
                    break label;
                } // else

            } // if
            else {
                // 1 - Ungültiger Zug!
                // Die i-te Dame ist nicht korrekt auf dem Feld plaziert
                val = 1;
                return val;
            } // if
        } // if
    } // for

    if (zugnr==0) {
        // 0 - Es ist noch keine Dame auf dem Spielfeld plaziert.
        val = 0;
    } // if

    return val;
}

private synchronized int forceMove(){
  working = true;
  int ret = -1;
  xCoord = -1;
  yCoord = -1;
  loesx = new int[8];
  Lb = new boolean[8];
  Lc = new boolean[16];
  Ld = new boolean[16];

  eine_loesung = false;
  if (zugnr<8) {
      tryMove(0);
  } // if
  if (!eine_loesung) {
      // Es ist keine Lösung mehr möglich!
      ret = 4;
  } // if
  else {
      // Es kann noch eine Dame auf das Feld (x,y) gesetzt werden.
      ret = 5;
  } /// if
  working = false;
  return ret;
}

int zaehler;

private void tryMove(int i){

  for (int j=0;j<8;j++) {
      if (!(Lb[j] || Ld[i+j] || Lc[i-j+7])) {
          loesx[i] = j;
```

```
            Lb[j] = true;
            Ld[i+j] = true;
            Lc[i-j+7] = true;
            if (i<7) {
                tryMove(i+1);
            } // if
            else {
                if (!eine_loesung) {
                    zaehler = 0;
                    for (int k=0;k<zugnr;k++) {
                        if (loesx[zugx[k]] == zugy[k]) {
                            zaehler++;
                        } // if
                    } // for
                    if (zaehler == zugnr) {
                        eine_loesung = true;
                        for (int l=0;l<8;l++) {
                            if (!a[l]) {
                                xCoord = l;
                                yCoord = loesx[l];
                                l = 9;
                            } // if
                        } // for
                    } // if
                } // if
            } // else
            Lb[j] = false;
            Ld[i+j] = false;
            Lc[i-j+7] = false;
        } // if
    } // for
  }
}
```

Die Klasse `Functional_GrashofSumme`

```
public class Functional_GrashofSumme extends Functional {

  Measure d1, d2, d3, d4;
  int choice;

  public void init(int numElem, String[] eList, Slate sl, Measure parent) {
      slate = sl;
      numElement = numElem;
      elementList = eList;

      d1 = (Measure) slate.lookupElement(elementList[1]);
      d2 = (Measure) slate.lookupElement(elementList[2]);
      d3 = (Measure) slate.lookupElement(elementList[3]);
      d4 = (Measure) slate.lookupElement(elementList[4]);
      choice = MathFunc.grepInt(elementList[5]);
      parent.addParent(d1,d2,d3,d4);
  }

  /*
     choice=1: liefert die Summe des größten und kleinsten Werts von d1-d4
     choice=0: liefert die Summe der beiden mittleren Werte von d1-d4
     choice=sonst: liefert 0.0
  */
  public double getValue() {
      double tmp;
      double[] d = new double[4];
      d[0] = d1.getValue();
      d[1] = d2.getValue();
      d[2] = d3.getValue();
      d[3] = d4.getValue();
      // Array sortieren
      for (int i=1;i<=3;i++) {
          for (int j=3;j>=i;j--) {
              if (d[j-1]<d[j]) {
              } // if
              else {
                  tmp = d[j-1];
                  d[j-1] = d[j];
                  d[j] = tmp;
              } // else
          } // for
      } // for

      if (choice==1) {
          // Summe vom größten und kleinsten Wert
          return d[0]+d[3];
      } // if

      if (choice==0) {
          // Summe der beiden mittleren Werte
          return d[1]+d[2];
      } // if

      return 0.0;
  }
}
```

Anhang E
Fragebögen der Evaluation

Evaluation des Online-Skripts zur Elementargeometrie

Das Online-Skript zur Elementargeometrie wurde im Rahmen eines Promotionsvorhabens an der Bildungswissenschaftlichen Universität Flensburg entwickelt. Helfen Sie mit, es durch Ihre Rückmeldung zu verbessern.
Für das Ausfüllen des Fragebogens bedanke ich mich recht herzlich!

Timo Ehmke

E-Mail: ehmke@uni-flensburg.de
Online-Skript: http://www.uni-flensburg.de/mathe/diss_ehmke/elementargeometrie/inhalt.htm

Allgemeine Evaluation

Im folgenden werden einige allgemeine Aussagen zum Online-Skript gemacht. Geben Sie bitte zu jeder Aussage an, inwieweit Sie dieser zustimmen können. Drücken Sie den Grad Ihrer Zustimmung durch einen Wert auf der Skala 0-5 aus.

Die Handhabung des Online-Skriptes fiel mir leicht.
Grad meiner Zustimmung:

überhaupt nicht 0 ❑	mit starken Einschränkungen 1 ❑	mit Einschränkungen 2 ❑	im wesentlichen 3 ❑	fast vollständig 4 ❑	vollständig 5 ❑

Die Gliederung des Online-Skriptes erscheint mir sinnvoll.
Grad meiner Zustimmung:

überhaupt nicht 0 ❑	mit starken Einschränkungen 1 ❑	mit Einschränkungen 2 ❑	im wesentlichen 3 ❑	fast vollständig 4 ❑	vollständig 5 ❑

Die sprachliche Darstellung erscheint mir prägnant.
Grad meiner Zustimmung:

überhaupt nicht 0 ❑	mit starken Einschränkungen 1 ❑	mit Einschränkungen 2 ❑	im wesentlichen 3 ❑	fast vollständig 4 ❑	Vollständig 5 ❑

Die Figuren waren übersichtlich.
Grad meiner Zustimmung:

überhaupt nicht 0 ❑	mit starken Einschränkungen 1 ❑	mit Einschränkungen 2 ❑	im wesentlichen 3 ❑	fast vollständig 4 ❑	Vollständig 5 ❑

Die Figuren funktionierten, wie ich es erwartete.
Grad meiner Zustimmung:

überhaupt nicht 0 ❑	mit starken Einschränkungen 1 ❑	mit Einschränkungen 2 ❑	im wesentlichen 3 ❑	fast vollständig 4 ❑	Vollständig 5 ❑

Die Wartezeit beim Laden und Initialisieren war akzeptabel.
Grad meiner Zustimmung:

überhaupt nicht 0 ❑	mit starken Einschränkungen 1 ❑	mit Einschränkungen 2 ❑	im wesentlichen 3 ❑	fast vollständig 4 ❑	vollständig 5 ❑

Kenndaten

Die Angaben zu Ihrer Person werden wir unter Berücksichtigung der Datenschutzbestimmungen anonym und vertraulich behandeln.

Geschlecht:

weiblich ❑	männlich ❑

Alter:

0-20 ❑	21-30 ❑	31-40 ❑	41-50 ❑	51-60 ❑	61- ❑

Beruf:

Schüler/in ❑	Student/in (Lehramt) ❑	Student/in (andere Fachrichtung) ❑	Lehrberuf ❑	nicht aufgeführt ❑

Themenspezifische Evaluation

Ich habe vorwiegend mit dem folgenden Thema gearbeitet:

Schwerpunkte	Satz von Varignon	Satz von Ceva	Winkelhalbierenden-Vierecke	Gelenkvierecke
❑	❑	❑	❑	❑

Ich habe ca. _____ Minuten mit dem Online-Skript gearbeitet.

Ich habe gerne mit dem Online-Skript gearbeitet.
Grad meiner Zustimmung:

überhaupt nicht	mit starken Einschränkungen	mit Einschränkungen	im wesentlichen	fast vollständig	vollständig
0	1	2	3	4	5
❑	❑	❑	❑	❑	❑

Die interaktiven Figuren haben mir das Verständnis erleichtert.
Grad meiner Zustimmung:

überhaupt nicht	mit starken Einschränkungen	mit Einschränkungen	im wesentlichen	fast vollständig	vollständig
0	1	2	3	4	5
❑	❑	❑	❑	❑	❑

Die Interaktion mit den Figuren fiel mir leicht.
Grad meiner Zustimmung:

überhaupt nicht	mit starken Einschränkungen	mit Einschränkungen	im wesentlichen	fast vollständig	vollständig
0	1	2	3	4	5
❑	❑	❑	❑	❑	❑

Ich konnte die Aufgaben leicht lösen.
Grad meiner Zustimmung:

überhaupt nicht	mit starken Einschränkungen	mit Einschränkungen	im wesentlichen	fast vollständig	vollständig
0	1	2	3	4	5
❑	❑	❑	❑	❑	❑

Was hat Ihnen an dem Online-Skript gut gefallen?

_____

_____

Was hat Ihnen insgesamt nicht so gut gefallen? Haben Sie Verbesserungsvorschläge? Haben Sie Programmfehler festgestellt?

_____

_____

Anhang F
Korrelationstabelle

Tabelle 19: Korrelation aller Variablen

	Handhabung	Gliederung	Sprachprägnanz	Übersichtlichkeit	Erwartungskonformität	Wartezeit	Bearbeitungsdauer	Bearbeitungsfreude	Verständniserleichterung	Figureninteraktion	Aufgabenschwierigkeitsgrad
Handhabung		0.69 (ss)	0.51 (ss)	0.64 (ss)	0.57 (ss)	0.28	0.06	0.49 (ss)	0.62 (ss)	0.42 (ss)	0.43 (ss)
Gliederung			0.47 (ss)	0.55 (ss)	0.48 (ss)	0.18	0.18	0.51 (ss)	0.56 (ss)	0.51 (ss)	0.49 (ss)
Sprachprägnanz				0.51 (ss)	0.43 (ss)	0.19	0.14	0.38 (s)	0.45 (ss)	0.51 (ss)	0.35 (s)
Übersichtlichkeit					0.71 (ss)	0.22	0.06	0.35 (s)	0.67 (ss)	0.59 (ss)	0.45 (ss)
Erwartungskonformität						0.19	0.08	0.45 (ss)	0.59 (ss)	0.53 (ss)	0.44 (ss)
Wartezeit							0.14	0.17	0.14	-0.09	-0.05
Bearbeitungsdauer								0.20	0.09	0.09	0.10
Bearbeitungsfreude									0.51 (ss)	0.44 (ss)	0.45 (ss)
Verständniserleichterung										0.51 (ss)	0.48 (ss)
Figureninteraktion											0.62 (ss)
Aufgabenschwierigkeitsgrad											
$r(38; 0.05) = 0,313$ signifikant (s) $r(38; 0.01) = 0,404$ sehr signifikant (ss)											

Programmverzeichnis

Cabri-Géomètre II von Laborde, J. M. & Bellemain, F.
Texas Instruments, 1988-96
Version 1.1 MS-DOS
http://www.cabri.net/

Cinderella von Richter-Gebert, J. & Kortenkamp, U. H.
Berlin (u. a.): Springer-Verlag, 1996-1999
http://www.cinderella.de

Dr. Geo von Fernandes, H.
Taiwan, 1997-98
Version 0.7.3b
http://members.xoom.com/FeYiLai/dr_geo/doctor_geo.html

Elly für Windows von Ingold, J.
Hannover: Cornelsen Software 1994

Euklid von Mechling, R.
Offenburg, 1994-1999
Version 2.0
http://www.mechling.de

Geolog von Holland, G.
Bonn: Ferdinand Dümmlers Verlag, 1996
Version 4.0b
http://www.uni-giessen.de/~ gcp3/geologde.htm

Geometry-Applet von Joyce, D.
Clark University: 1996-1997
Version 2.0.0, 3d
http://aleph0.clarku.edu/~ djoyce/java/Geometry/Geometry.html

Geonet Lehrstuhl für Mathematik und ihre Didaktik der Universität Bayreuth
1997-1999
http://did.mat.uni-bayreuth.de/geonet

Java Development Kit von Sun Microsystems, Inc.
Version 1.1.5
Palo Alto, USA: 1994-1997
http://java.sun.com/products/jdk/1.1/

KMSS von Kleiter, E. F.
Kleiter-Microcomputer-Statistik-Software
Kiel 1988-1996
http://www.uni-flensburg.de/psychologie/statisti.htm

The Geometer's Sketchpad von Jackiw, N.
Key Curriculum Press 1992-1997
Version 3.10d
http://www.keypress.com/product_info/sketch-demo.html

Thales von Kadunz, G. & Kautschitsch, H.
Stuttgart: Klett-Verlag, 1993
Version 1.02a

The Geometric Supposer von Schwarz, J. L. & Yerushalmy, M.
Sunburst Communications Inc., New York
1985-1993

Zirkel und Lineal von Grothmann, R.
KU-Eichstätt 1999
Version 2.0
http://mathsrv.ku-eichstaett.de/MGF/homes/grothmann/zul.html

Literaturverzeichnis

[Baptist 1992] Baptist, P.: Die Entwicklung der neueren Dreiecksgeometrie. Mannheim, Leipzig, Wien, Zürich: BI-Wiss.-Verl. 1992

[Bender & Schreiber 1985] Bender, P., Schreiber, A.: Operative Genese der Geometrie. Wien: Hölder-Pichler-Tempsky, Stuttgart: Teubner 1985

[Biehler 1992] Biehler, R.: Entwicklungen bei didaktisch-orientierten Softwarewerkzeugen zur Geometrie. Vom interaktiven Programmieren zur direkten Interaktion. In: Zentralblatt für Didaktik der Mathematik 24 (4), S. 121-127

[Blaschke & Müller 1956] Blaschke, W., Müller, H. R.: Ebene Kinematik. München: Oldenbourg 1956

[Bortz 1977] Bortz, J.: Lehrbuch der Statistik. Berlin, Heidelberg, New York: Springer 1977

[Brieskorn 1981] Brieskorn, E., Knörrer, H.: Ebene algebraische Kurven. Basel u. a.: Birkhäuser 1981

[Claus 1982] Claus, H. J.: Extremwertaufgaben in metrischen Räumen. Der Mathematik-Unterricht, Heft 5, 1982, S. 27-58

[Claus 1992] Claus, H. J.: Extremwertaufgaben. Probleme, ihre Geschichte, Lösungen, Methoden. Darmstadt: Wissenschaftliche Buchgesellschaft 1992

[Danckwerts & Vogel 1998] Danckwerts, R., Vogel, D.: Extremwertaufgaben ja, aber wie? MNU 51, Heft 2, Bonn: Dümmler 1998, S. 74-79

[Dudeney 1958] Dudeney, H. E.: The Canterbury puzzles and other curious problems. 4. Auflage, New York: Dover Publications 1958

[Dudeney 1917] Dudeney, H. E.: Amusements in mathematics. New York: Dover Publications 1970 (Nachdruck der Ausgabe von 1917)

[Eckel 1989] Eckel, K.: Didaktiksprache. Grundlagen einer strengen Unterrichtswissenschaft. Köln, Wien: Böhlau 1989

[Eigenmann 1981] Eigenmann, P.: Geometrische Denkaufgaben. Stuttgart: Klett 1981

[Elschenbroich 1996] Elschenbroich, H.-J.: Geometrie beweglich mit Euklid. Bonn: Dümmler 1996

[Elschenbroich 1997] Elschenbroich; H. J.: Tod des Beweisens oder Wiederauferstehung? - Zu Auswirkungen des Computereinsatzes auf die Stellung des Beweisens im Unterricht. In: Hischer, H. (Hrsg.): Computer und Geometrie. Neue Chancen für den Geometrieunterricht? Bericht über die 14. Arbeitstagung des Arbeitskreises "Mathematikunterricht und Informatik" in der Gesellschaft für Didaktik der Mathematik e. V., Hildesheim: Franzbecker 1997, S. 58-62

[Elschenbroich 1999] Elschenbroich; H. J.: Anschaulich(er) Beweisen mit dem Computer? Neue Möglichkeiten für visuelle Beweise. In: Kadunz, G., Ossimitz, G., Peschek, W., Schneider, E., Winkelmann, B.: Mathematische Bildung und neue Technologien. Vorträge beim 8. Internationalen Symposium zur Didaktik der Mathematik, Universität Klagenfurt, 28.9.-2.10.1998, Stuttgart, Leipzig: Teubner 1999, S. 61-68

[Embacher 1998] Embacher, F.: Multimedia-Didaktik und spontanes Verstehen. In: Kadunz, G., Ossimitz, G., Peschek, W., Schneider, E., Winkelmann, B.: Mathematische Bildung und neue Technologien. Vorträge beim 8. Internationalen Symposium zur Didaktik der Mathematik, Universität Klagenfurt, 28.9.-2.10.1998, Stuttgart, Leipzig: Teubner 1999, S. 69-76

[Fittkau & Maaß 1999] Fittkau, Maaß: W3B-Umfrage. Hamburg 1999, http://www.w3b.de

[Fukagawa & Pedoe 1989] Fukagawa, H., Pedoe, D.: Japanese Temple Geometry Problems. San Gaku. Winnipeg, Canada 1989

[Gardner 1971] Gardner, M.: Logik unterm Galgen. Braunschweig: Vieweg 1971

[Graumann et al. 1996] Graumann, G., Hölzl, R., Krainer, K., Neubrand, M., Struve, H.: Tendenzen der Geometriedidaktik der letzten 20 Jahre. Journal für Mathematik-Didaktik 17, Heft 3/4, 1996, S. 163-237

[Hale 1997] Hale, M., Cannings, R., Cross, D., van Kooten, J., Lemire, D., Smith, T.: JavaSci - A science API for Java. 1997-2000, http://fourier.dur.ac.uk:8000/~ dma3mjh/jsci/index.html

[Heigl 1998] Heigl, A.: Fraktale im Mathematikunterricht. Köln: Aulis Deubner 1998

[Henn 1995] Henn, H.-W.: Ebene und runde Spiegel. In: Der Mathematik-Unterricht, Heft 1, 1995, S. 25-33

[Henn 1996] Henn, H.-W.: Entdeckendes Lernen im Umkreis von zentrischer Streckung und Strahlensätzen. In: Mathematik lehren, Heft 82, 1996, S. 48-51

[Henn & Jock 1992] Henn & Jock: Arbeitsbuch Cabri-Géomètre. Bonn: Dümmler 1992

[Hischer 1997] Hischer, H. (Hrsg.): Computer und Geometrie. Neue Chancen für den Geometrieunterricht? Bericht über die 14. Arbeitstagung des Arbeitskreises "Mathematikunterricht und Informatik" in der Gesellschaft für Didaktik der Mathematik e. V., Hildesheim: Franzbecker 1997

[Hischer 1998] Hischer, H. (Hrsg.): Geometrie und Computer. Suchen, Entdecken und Anwenden. Bericht über die 15. Arbeitstagung des Arbeitskreises "Mathematikunterricht und Informatik" in der Gesellschaft für Didaktik der Mathematik e. V., Hildesheim: Franzbecker 1998

[Hole 1998] Hole, V.: Erfolgreicher Mathematikunterricht mit dem Computer. Methodische und didaktische Grundfragen in der Sekundarstufe I. Donauwörth: Auer 1998

[Holland 1996] Holland, G.: Geometrie in der Sekundarstufe. Didaktische und methodische Fragen. 2. Auflage. Heidelberg, Berlin, Oxford: Spektrum Akademischer Verlag 1996

[Hölzl 1994] Hölzl, R.: Im Zugmodus der Cabri-Geometrie. Interaktionsstudien und Analysen zum Mathematiklernen mit dem Computer. Weinheim: Deutscher Studien Verlag 1994

[Hölzl 1995] Hölzl, R.: Eine empirische Untersuchung zum Schülerhandeln mit Cabri-Géomètre. In: Journal für Mathematik-Didaktik 16, Heft 1/2, 1995

[Hölzl 1999] Hölzl, R.: Aspekte des heuristischen Einsatzes von Dynamischer Geometriesoftware. In: Der Mathematik-Unterricht, Heft 1, 1999, S. 52-60

[Hölzl & Schneider 1996] Hölzl, R., Schneider, W.: Die Inversion am Kreis. In: Mathematik lehren, Heft 82, 1996, S. 53-56

[Kadunz 1996] Kadunz, G.: Experimentelle Geometrie. Entwicklung und Bewertung von Software für den Geometrieunterricht. Dissertation, Klagenfurt 1996

[Kadunz 1999] Kadunz, G., Ossimitz, G., Peschek, W., Schneider, E., Winkelmann, B.: Mathematische Bildung und neue Technologien. Vorträge beim 8. Internationalen Symposium zur Didaktik der Mathematik, Universität Klagenfurt, 28.9.-2.10.1998, Stuttgart, Leipzig: Teubner 1999

[Kaput 1988] Kaput, J. J.: Looking Back From the Future. A History of Computers in Mathematics Education. Unveröffentlichtes Manuskript, Cambridge 1978-1998

[King & Schattschneider 1997] King, J. R., Schattschneider, D. (Hrsg.): Geometry Turned On! Dynamic Software in Learning, Teaching and Research. Washington, D.C.: The Mathematical Association of America 1997

[Kleiter 1988] Kleiter, E. F.: Lehrbuch der Statistik in KMSS. Band 1/1: Überblick und niedrig-komplexe Verfahren. Weinheim: Deutscher Studien Verlag 1988

[Kortenkamp 1999] Kortenkamp, U.: Foundations of Dynamic Geometry. Dissertation, Zürich 1999

[Lugon, Chastellain & Atzbach 1991] Lugon, S., Chastellain, M., Atzbach, R.: Cabricolages. Expeditionen in die Welt der ebenen Geometrie. Duisburg: CoMet, Cornelsen 1991

[Mason 1992] Mason, J., Burton, L., Stacey, K.: Hexeneinmaleins - kreativ mathematisch denken. 3. Auflage. München, Wien: Oldenbourg 1992

[Mechling 1997] Mechling, R.: Euklid von innen. In: Hischer, H. (Hrsg.): Computer und Geometrie. Neue Chancen für den Geometrieunterricht?, Hildesheim: Franzbecker 1997

[Meyer 1997] Meyer, J.: Bahnkurven als geometrische Objekte - Vernetzung durch Variation. In: Hischer, H. (Hrsg.): Computer und Geometrie. Neue Chancen für den Geometrieunterricht? Bericht über die 14. Arbeitstagung des Arbeitskreises "Mathematikunterricht und Informatik" in der Gesellschaft für Didaktik der Mathematik e. V., Hildesheim: Franzbecker 1997, S. 90-95

[Neubrand 1981] Neubrand, M.: Das Haus der Vierecke. Aspekte beim Finden mathematischer Begriffe. Journal für Mathematikdidaktik, 1981, Heft 2, S. 37-50

[Nie u. a. 1975] Nie, N. N., Hull, C. H., Jenkins, J. G., Steinbrenner, K., Bent, D. H.: SPSS. Statistical Package for the Social Sciences. New York: McGraw-Hill 1975

[Peitgen 1992] Peitgen, H. O., Jürgens, H., Saupe, D.: Bausteine des Chaos. Fraktale, Berlin: Springer/Klett-Cotta 1992

[Polya 1949] Polya, G.: Schule des Denkens. Vom Lösen mathematischer Probleme. 4. Auflage, Tübingen, Basel: Francke 1995

[Richter-Gebert 1999] Richter-Gebert, J., Kortenkamp, U.: The interactive geometry software Cinderella. Berlin: Springer 1999

[Rothman 1998] Rothmann, T.: Japanese Temple Geometry. Scientific American, Heft 5, 1998, http://www.sciam.com/1998/0598issue/-0598rothman.html

[Schreiber 1996] Schreiber, A.: Mathematische Lernbausteine in Software-Umgebungen – ihre Entwicklung, Integration und flexible Nutzung. Preprint. Arbeiten aus dem Institut für Mathematik und ihre Didaktik. Bildungswissenschaftliche Hochschule Flensburg, Universität, Heft 6, 1996

[Schreiber 1998] Schreiber, A.: CBT-Anwendungen professionell entwickeln. Berlin, Heidelberg, New York: Springer 1998

[Schreiber 1999] Schreiber, A.: Heuristik / Problemlösen. Materialien zu einem Seminar an der Universität Flensburg (seit 1998), http://www.uni-flensburg.de/mathe/zero/veranst/heuristik/heuristik.html

[Schumann 1991] Schumann, H.: Schulgeometrisches Konstruieren mit dem Computer. Beiträge zur Didaktik des interaktiven Konstruierens. Stuttgart: Metzler/Teubner 1991

[Schumann 1994] Schumann, H.: Cabri-Géomètre. Eine Evaluation durch Lehrer der Bundesrepublik Deutschland. Hannover: Cornelsen 1994

[Schumann 1997] Schumann, H.: Interaktive Arbeitsblätter für das Geometrielernen. Preprint. PH Weingarten 1997

[Schumann 1998] Schumann, H.: Dynamische Behandlung elementarer Funktionen. In: Mathematik in der Schule 36 (1998) 3, S. 172-189

[Schumann 1999] Schumann, H.: Computerunterstützte Behandlung geometrischer Extremwertaufgaben. Materialien Gymnasium Mathematik. Landesinstitut für Erziehung und Unterricht: Stuttgart 1999

[Schumann & Green 1997] Schumann, H., Green, D.: Producing and using Loci with Dynamic Geometry Software. In: King, J. R., Schattschneider, D. (Hrsg.): Geometry Turned On! Dynamic Software in Learning, Teaching and Research. Washington, D.C.: The Mathematical Association of America 1997, S. 79-87

[Schupp 1992] Schupp, H.: Optimieren. Lehrbücher und Monographien zur Didaktik der Mathematik. Band 20. Mannheim: BI-Wiss.-Verlag 1992

[Schupp & Dabrock 1995] Schupp, H., Dabrock, H.: Höhere Kurven - situative, mathematische, historische und didaktische Aspekte. Mannheim: BI-Wiss.-Verlag 1995

[Shaffer 1995] Shaffer, D.: Exploring Trigonometry with the Geometers' SketchPad. Key Curriculum Press 1995

[Schwarze 1996] Schwarze, M.: Von beweglichen Vierecken und Scheibenwischern. In: Mathematik lehren, Heft 82, 1996, S. 9-12

[vom Hofe 1999] vom Hofe, R.: Explorativer Umgang mit Funktionen - Interaktion und Kommunikation in selbstorganisierten Arbeitsphasen. Journal für Mathematik-Didaktik 20, Heft 2/3, 1999, S. 186-221

[Weigand 1996] Weigand, H.-G.: Mechanisches und computerunterstütztes Zeichnen von Kegelschnitten. In: Mathematik lehren, Heft 82, 1996, S. 14-18

[Wellstein 1999] Wellstein, H.: Elementargeometrie. Unveröffentlichtes Manuskript, Universität Flensburg 1999

[Weth 1991] Weth, T.: Ein abbildungsgeometrischer Zugang zu algebraischen Kurven dritter und höherer Ordnung. In: Didaktik der Mathematik, Heft 2, 1991, S. 145-164

[Winter 1989] Winter, H.: Entdeckendes Lernen im Mathematikunterricht. Einblicke in die Ideengeschichte und ihre Bedeutung für die Pädagogik. Braunschweig, Wiesbaden: Vieweg 1989

[Wirth 1975] Wirth, N.: Algorithmen und Datenstrukturen. Stuttgart: Teubner 1975

[Wurm 1996] Wurm, C.: Erzeugung von Rollkurven mit einem Geometrieprogramm. In: Mathematik lehren, Heft 82, 1996, S. 57-60

Versicherung

Ich versichere, die vorliegende Arbeit selbständig angefertigt, nur die von mir angegebenen Hilfsmittel benutzt und sämtliche dem Wortlaut oder dem Inhalt nach aus anderen Schriften übernommenen Stellen unter genauer Quellenangabe als solche gekennzeichnet zu haben.

Rotenburg (Wümme), den 4. Juli 2000

Timo Ehmke

Lebenslauf

Persönliche Angaben:

Name:	Timo Ehmke
Geburtsort / -tag:	Lübeck, den 29.11.1971
Familienstand:	verheiratet mit Katja Ehmke-Janell
Kinder:	Tochter Sophie Ehmke

Schulausbildung:

1978 – 1981	Grundschule am Koggenweg in Lübeck
1981 – 1982	Grundschule am Masurenweg in Bad Oldesloe
1982 – 1988	Dietrich-Buxtehude-Realschule in Bad Oldesloe
1988 – 1991	Technisches Gymnasiums in Bad Oldesloe
06/1991	Abschluß: Allgemeine Hochschulreife

Ersatzdienst:

08/1991 – 10/1992	Pflegedienst im Franziskus-Hospital in Flensburg

Universitätsstudium:

10/1992 – 03/1993	Studium an der Universität Kiel Lehramt Mathematik / Ev. Religion
04/1993 – 09/1996	Studium an der Universität Flensburg Lehramt Mathematik (Schwerpunkt Informatik) / Technik Staatsexamensarbeit: "Zuordnungen als lokale Interaktionsform in Lernprogrammen" Abschluß: 1. Staatsexamen
05/1997 – 03/2001	Doktorand am Institut für Mathematik und ihre Didaktik der Universität Flensburg Abschluß: Promotion

Praktische Tätigkeiten:

10/1995 – 09/1998	Studentische Hilfskraft am Rechenzentrum der Universität Flensburg
10/1996 – 04/1997	Mitarbeit am Projekt "Multimediales Lehr-/Lernsystem für die Ausbildung von Mathematiklehrer/innen" am Institut für Mathematik und ihre Didaktik der Universität Flensburg
09/2000 – 11/2000	Mitarbeit am Projekt "ZERO – Mathematik online" am Institut für Mathematik und ihre Didaktik der Universität Flensburg