FernUniversität
Gesamthochschule in Hagen

Begleitmaterial zum Kurs 1580/1582/1584

Aufgabenbeschreibung
für das
Java-Programmierpraktikum
im Wintersemester 2002/03

Praktische Informatik IV

Inhalt

1 Das Programmierpraktikum

Das Programmierpraktikum dient der Übung im praktischen Umgang mit einer gängigen Programmiersprache. Zur Zeit findet dazu die Programmiersprache *Java* Anwendung, die aufgrund ihrer Konzepte und insbesondere auch aufgrund ihrer Eignung für Internet-Anwendungen weithin Akzeptanz gefunden hat. Gute Java-Kenntnisse sind deshalb Voraussetzung für eine erfolgreiche Teilnahme an diesem Praktikum. Sie sollen lernen, die Lösung zu einem überschaubaren Problem zu finden und mit Hilfe der Programmiersprache Java zu implementieren.

Wie bereits im letzten vom Lehrgebiet Praktische Informatik IV durchgeführten Programmierpraktikum können Sie je nach Interesse und Fähigkeiten zwischen sechs Themen wählen. Diese sind im einzelnen:

- Morsealphabet
- Lottoschein-Lesegerät
- Puzzle
- Rollende Kugel
- Räumliche Darstellung mit Stereobildern
- Räumliche Darstellung durch Überlagerung repetitiver Muster

Unsere Absicht ist es, mit diesen Aufgaben ein so breites Spektrum von Themen abzudecken, daß jeder etwas darunter finden kann, das seinen persönlichen Neigungen entspricht. Wir haben an manchen Stellen bewußt auf eine detaillierte Spezifikation des von Ihnen zu erstellenden Programms verzichtet, insbesondere in Bezug auf die graphische Benutzeroberfläche und die Interaktion von Programm und Benutzer, um Ihnen die Möglichkeit zu geben, selbst kreativ zu werden und eigene Konzepte zu realisieren. Wir hoffen, daß Ihnen dadurch die Arbeit an Ihrer Aufgabe noch mehr Spaß macht.

Die Aufgaben weisen einen unterschiedlichen Schwierigkeitsgrad auf, der auch von Ihrer individuellen Vorbildung abhängig ist. Wählen Sie Ihre Aufgabe sorgfältig aus; berücksichtigen Sie dabei auch den Aspekt der Implementierbarkeit im Zeitrahmen des Programmierpraktikums.

Hinweis: Manche Aufgabenbeschreibungen nehmen Bezug auf farbige Darstellungen in Graphiken. Da der Druck der Aufgabenbeschreibungen aber in schwarz-weiß erfolgt, ist es notwendig, sich bezüglich der Farbinformation die zugehörige PDF-Datei auf der Home Page des Lehrgebiets anzusehen.

2 Morsealphabet

Um diese Aufgabe bearbeiten zu können, benötigen Sie einen PC mit Soundkarte, Mikrofon und Lautsprechern.

Das Morsealphabet wurde von Samuel Finley Breese Morse erfunden, 1843 errichtete er die erste Telegrafenlinie von Washington nach Baltimore. Im Morsealphabet zu kommunizieren wird auch als „morsen“ bezeichnet. Das Morsealphabet ist eine simples aber universelles Kommunikationsprotokoll, kleine Sequenzen von mehreren kurzen (·) oder langen Signalen (–) stellen eine Codierung für ein Zeichen des Alphabets dar. Als Signale dienen meistens Schall– oder Lichtimpulse. Der internationale Hilferuf sos (save our souls) sieht beispielsweise folgendermaßen aus: · · · – – – · · ·, also dreimal kurz, dreimal lang und dreimal kurz. Das komplette Morsealphabet benutzt die unten aufgeführten Codierungen:

Buchstaben (maximal 4 Zeichen)

A · –	G – – ·	M – –	S · · ·	Y – · – –
B – · · ·	H · · · ·	N – ·	T –	Z – – · ·
C – · – ·	I · ·	O – – –	U · · –	
D – · ·	J · – – –	P · – – ·	V · · · –	
E ·	K – · –	Q – – · –	W · – –	
F · · – ·	L · – · ·	R · – ·	X – · · –	

Ziffern (5 Zeichen)

1 · – – – –	3 · · · – –	5 · · · · ·	7 – – · · · ·	9 – – – – ·
2 · · – – –	4 · · · · –	6 – · · · ·	8 – – – · ·	0 – – – – –

Interpunktion (6 Zeichen)

. · – · – · – (AAA)	, – – · · – – (MIM)	? · · – – · · (IMI)

Das Ende einer Zeichensequenz wird durch sehr kurze Pausen, das Ende ganzer Wörter durch längere Pausen signalisiert. Das Leerzeichen entspricht also einer bestimmten Ruhezeit. Innerhalb einer Zeichensequenz werden die langen und kurzen Signale durch

extrem kurze Ruhepausen voneinander separiert. Im wesentlichen soll ein Programm entwickelt werden, welches einen Text in Morsezeichen umwandelt und umgekehrt, als Ein– und Ausgabe dienen Mikrofon und Lautsprecher. Hauptaufgabe dabei ist die Erzeugung und Analyse von Audio–Datenströmen.

2.1 Grundlagen der Audioverarbeitung

Schall ist ein Wellenphänomen, kugelförmig um eine Schallquelle breitet sich eine Druckschwankung der Luft aus. Ein Schallsignal besteht aus der Überlagerung von Schallwellen mit verschiedenen Frequenzen. Misst man an einer bestimmten Stelle den Schalldruck über einen bestimmten Zeitraum, so erhält man einen Schwingungsförmigen Verlauf. Über ein Mikrofon wird diese Schalldruckschwankung zunächst in die mechanische Schwingung einer Membran und dann in elektrische Schwingungen umgesetzt. Dieses analoge Signal wird nun durch einen Analog/Digital–Konverter, einen spezialisierten Chip auf der Soundkarte, in ein digitales Format überführt.

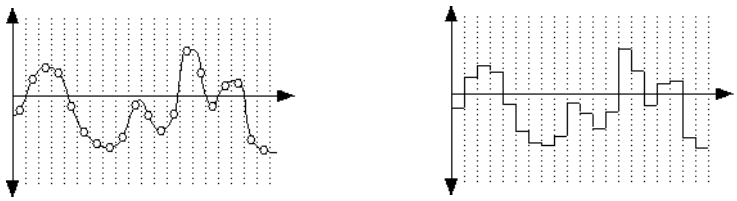


Abbildung: Sampling[1]

Diesen Vorgang nennt man Sampling. Umgekehrt läßt sich über einen Digital/Analog–Wandler aus einem gesampelten Signal wieder ein analoges Signal herstellen, welches dann eine Lautsprechermembran in Schwingung versetzt, um Schallwellen zu erzeugen. Folgende Begriffe werden in diesem Zusammenhang benutzt:

Encoding: Wie wird das analoge Signal in Zahlenwerte umgewandelt? Das einfachste Verfahren ist PCM (pulse code modulation), bei dem, wie in der obigen Abbildung gezeigt, die Amplitudenwerte auf einen durch wenige Bits darstellbaren Bereich umgerechnet (quantisiert) werden.

Sample–Size: Auflösung in k Bits, in der das Signal auf einer Skala zwischen 0 bis 2^k (unsigned) bzw. $-2^{(k-1)}$ bis $2^{(k-1)}$ (signed) dargestellt wird.

1. Quelle: Guido Krüger, Handbuch der Java-Programmierung, 3. Auflage, Addison-Wesley. HTML Version zum freien Download unter http://www.javabuch.de

Sample–Rate: Anzahl der Samples, die pro Sekunde pro Aufnahmekanal (Stereo = 2, Mono = 1) erzeugt werden.

Frame: Ein Frame umfasst die Anzahl aller gleichzeitig erstellten Samples aller Aufnahmekanäle. Im PCM–Format ist die Frame–Rate gleich der Sample–Rate. Dies muss nicht immer so sein, kompliziertere Formate können auch ganze Serien von Samples und zusätzliche Informationen enthalten.

Beispiele

Ein PCM–Datenstrom mit einer Sample–Rate von 8000, Auflösung 8 Bit und einem Aufnahmekanal (Mono) enthält 8000 Byte pro Sekunde. Wird dagegen in CD–Qualität (Sample– Rate 44100, Auflösung 16 Bit, Stereo) aufgenommen, so fallen 176400 Byte je Sekunde an.

2.2 Java Sound–API

Seit der Version 1.3 enthält das JDK die Pakete javax.sound.sampled und javax.sound.midi. Ersteres dient der Behandlung von Digital Audio, und letzteres ermöglicht es MIDI[1]–Daten an Musikgeräte zu senden oder zu empfangen. Das MIDI–Datenformat orientiert sich am Notensatz, hier werden Daten über die Tonhöhe, die Länge, die Anschlagsstärke und die Klangfarbe zwischen Programm und Musikgerät ausgetauscht.

Einen guten Überblick erhält man, wenn man die Java Sound–Demo[2] installiert, und anhand des Beispielcodes, der API–Dokumentationen und dem Java Sound–Programmer–Guide[3] die Konzepte der Sound–Programmierung näher studiert.

Mindestanforderungen

Mit den oben beschriebenen Hilfsmitteln ist nun ein Programm unter Berücksichtigung folgender Aspekte zu realisieren:

- Aus verschiedenen Eingabequellen soll ein Strom von Morsecodes erzeugt werden, der graphisch als Punkt–Strich–Code und als Text visualisiert werden soll.

1. Musical Instruments Digital Interface
2. http://java.sun.com/products/java-media/sound/index.html
3. http://java.sun.com/j2se/1.4/docs/guide/sound

Eingabequellen sind die weiter unten beschriebenen Dateiformate, die Tastatur und das Mikrofon.

- Im Aufnahmemodus können über Mikrofon oder Tastatur Morsesequenzen eingegeben werden. Eine Überarbeitung des erkannten Textes nach der Aufnahme sollte möglich sein.
- Im Abspielmodus kann die aktuell im Speicher enthaltene Nachricht über die Soundkarte abgespielt werden. Dabei können die Abspielgeschwindigkeit in Buchstaben pro Zeiteinheit, sowie Musikinstrument und Tonhöhe variiert werden.
- Mittels einer Importfunktion können Morsesequenzen aus einer Audiodatei, einer Morsecodedatei, die Punkt–Strich–Codes enthält, oder einer Textdatei eingelesen werden. Über eine Exportfunktion kann die Nachricht in einem der o.g. Formate unter Beachtung der aktuell eingestellten Abspielparameter gespeichert werden.
- Um das Erlernen des Morsecodes zu unterstützen, soll ein Diktatmodus existieren, in dem entweder zu einer Tonfolge der Text eingegeben, oder zu einem Text die Morsecodes eingetastet werden müssen. Hierzu kann eine Datei geladen werden, und nach Auswahl einer Abspielgeschwindigkeit und der Diktatform werden Töne abgespielt bzw. Buchstaben angezeigt. Nach Ablauf der Nachricht werden die Eingaben des Benutzers ausgewertet, und eventuell vorhandene Fehler angezeigt.

Datei–Formate

Eine Audiodatei sollte in allen vom Java Sound–API unterstützten Formaten (.wav, .au, etc ..) verarbeitet werden können. Beim Export sollte der Benutzer ein Format auswählen können.

Ein Textfile darf nur große und kleine lateinische Buchstaben ohne Umlaute enthalten. Leerraum kann beliebig vorhanden sein.

Eine Morsecodedatei enthält die Zeichen Punkt (.) und Minus (-), Buchstabensequenzen werden durch ein oder mehrere Leerzeichen oder Zeilenumbrüche getrennt.

Analyse–Algorithmus

Die anspruchvollste Teilaufgabe ist die Implementierung eines geeigneten Algorithmus zur Analyse eines Audio–Datenstromes. Einige Probleme, die dabei zu bewältigen sind, sollen hier kurz diskutiert werden.

Erkennung der Zeiträume von Pause und Signal: Dazu muss die Abweichung der Samples vom Null–Pegel betrachtet werden, ist in einem gewissen Zeitraum der Durchschnittswert der Samples innerhalb eines Toleranzbereiches, so wird dieses Zeitintervall als Pause gewertet, andernfalls als Signal.

Unterscheidung verschiedener Pausenarten: Einzelne Buchstaben werden beim Morsen durch kurze Pausen voneinander getrennt, ohne diese Pause wäre nicht entscheidbar ob die Folge · · – nun EA oder U bedeutet. Diese sind zu unterscheiden von den kürzeren Pausen, die dadurch entstehen, dass ein Buchstabe mit bis zu 4 Signalen – und 3 kurzen Pausen dazwischen – codiert wird. Problematisch kann dies werden, wenn das Morsesignal von einem Instrument erzeugt wird, welches nur eine sehr kurze Klangphase hat (z.B. eine Trommel) in diesem Fall müssen drei Arten von Pausen erkannt werden: Pausen nach einem langen Signal (–), Pausen nach Ende eines Buchstabens und Pausen zwischen den Signalen innerhalb eines Buchstabens. Die Wörter einer Nachricht werden ebenfalls durch eine Pause, welche deutlich länger sein sollte als die vorgenannten Pausenarten, voneinander getrennt.

Geschwindigkeit der Morsesequenzen: Die Pausen und Signalzeiten können innerhalb einer von Menschenhand erzeugten Nachricht leicht schwanken, daher muss auch hier für Pausenzeiten, sowie langen und kurzen Signalzeiten an einen Toleranzbereich gedacht werden.

Geschwindigkeit von Nachrichten: Im Idealfall sollte ein größerer Bereich von Übertragungsraten möglich sein, als untere Grenze sollten Nachrichten mit etwa 30 Zeichen pro Minute und als obere Grenze Nachrichten mit ca. 200 Zeichen pro Minute dekodierbar sein. Die Geschwindigkeit einer Nachricht sollte vom Algorithmus selbständig erkannt werden, damit er in der Lage ist, die Verhältnisse der Zeiten für kurze und lange Signal– bzw. Pausenzeiten selbständig zu erkennen und Toleranzgrenzen anzupassen.

Um die genannten Probleme zu lösen, sollten Sie zunächst von maschinell erzeugten Morsesequenzen ausgehen, die feste Verhältnisse von Pausen und Signalzeiten besitzen. Durch Experimentieren sollte danach die Erkennung robuster gestaltet werden, damit auch von Menschenhand erzeugte Morsesequenzen erkannt werden.

3 Lottoschein-Lesegerät

3.1 Aufgabenbeschreibung

Die Aufgabe bei diesem Thema besteht darin, die Informationen von ausgefüllten und eingescannten Lottoscheinen auszulesen und abzuspeichern. Als Vorlage dient ein handelsüblicher Lottoschein einer Lottogesellschaft (siehe Abbildung 3.1).

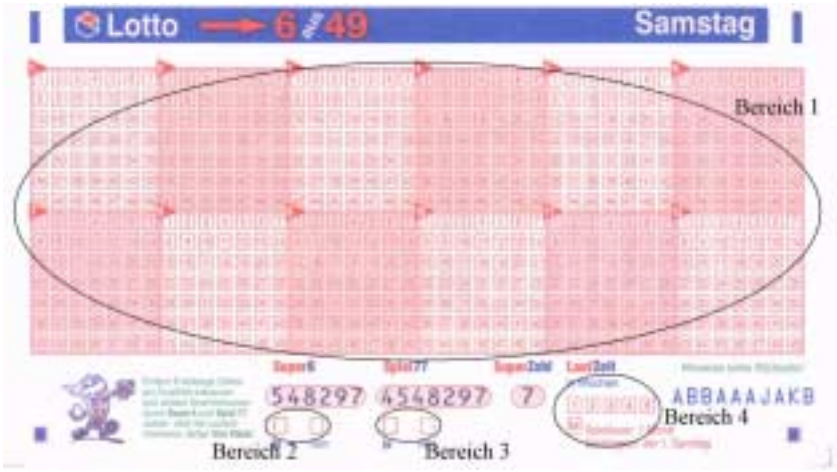


Abbildung 3.1: Die markierten Bereiche enthalten die relevanten Daten.

Auszulesen sind folgende Daten:

- Anzahl der getippten Felder (Bereich 1)
- getippte Zahlen pro Feld (Bereich 1)
- ja/nein bei Spiel „Super 6“ (Bereich 2)
- ja/nein bei „Spiel 77“ (Bereich 3)
- Laufzeit (Bereich 4)

Desweiteren ist zu prüfen, ob der Lottoschein gültig ausgefüllt ist, d.h. ob pro getipptem Feld genau sechs Zahlen angekreuzt sind, ob nur aufeinanderfolgende Felder beginnend bei Feld 1 ausgefüllt worden sind und ob und welche Kreuze bei den Zusatzspielen und der Laufzeit gemacht worden sind. Falls der Schein als gültig bewertet wird, sollen die Daten des Lottoscheins auf dem Bildschirm dargestellt werden. Die Möglichkeit zum Speichern der Daten soll ebenfalls vorhanden sein.

Zu implementieren ist unter anderem eine graphische Benutzeroberfläche, die die folgenden Funktionen hat:

- Auswählen eines Lottoscheins (hierbei soll der Scan des Lottoscheins als Bitmap-File ausgewählt werden)
- Prüfen des Lottoscheins (Auslesen und Prüfen der o.a. Informationen)
- Darstellen der Lottoschein-Daten
- Speichern der Daten
- Laden von Lottoschein-Daten (Laden und Darstellen von abgespeicherten Lottoschein-Daten)

Optional sind folgende, weitere Funktionen:

- Spiel durchführen (per Zufallsgenerator werden Lottozahlen gezogen)
- Gewinnsumme berechnen (für alle abgespeicherten Lottoscheine die Gewinne berechnen und darstellen)

3.2 Zur Umsetzung

Wir gehen davon aus, daß die Lottoscheine so eingescannt worden sind, daß die Ausrichtung, Skalierung, Auflösung und Farbtiefe aller Bilder gleich sind. Im Speziellen liegen hier Bitmap-Files mit 1 Bit Farbtiefe vor, die also nur schwarze und weiße Bildpunkte enthalten.

Das Verfahren zur Erkennung von Kreuzen auf dem Schein beruht nun darauf, Unterschiede zwischen einem ausgefüllten und einem nicht ausgefüllten Lottoschein zu finden. Darüber hinaus muß noch festgestellt werden, wo genau auf dem Schein Unterschiede bzw. Kreuze gefunden wurden. Dazu beginnt man damit, die interessanten Bereiche auf dem Schein zu identifizieren. Zu diesem Zweck wird der Lottoschein in verschiedene Abschnitte unterteilt. Die Aufteilung muß derart vorgenommen werden, daß für alle auf dem Schein vorhandenen Kreuze gilt, daß sie nicht im selben Abschnitt liegen. D.h. die Abschnitte müssen so klein sein, daß sie maximal ein Kästchen, in das ein Kreuz gemacht werden kann, enthalten.

Eine Aufteilung in derartige Abschnitte wird durch die Suche nach Zeilen und Spalten mit ausschließlich weißen Bildpunkten vorgenommen. In Abbildung 3.2 sehen wir einen ersten Aufteilungsschritt in drei Abschnitte. Hier wurde nach Zeilen mit ausschließlich weißen Bildpunkten gesucht. Als nächstes wird Abschnitt 3 nach „leeren“ Spalten untersucht. Das Ergebnis ist in Abbildung 3.3 zu sehen. Die Suche nach weiteren „leeren“ Zeilen und Spalten in den interessanten Unterabschnitten von Abschnitt 3 läßt sich in Abbildung 3.4 betrachten. Die geforderte Bedingung für die Aufteilung von Abschnitt 3 ist offensichtlich erfüllt.

Abbildung 3.2: Die erste Aufteilung in drei Abschnitte ist an den roten Linien zu erkennen.

Abbildung 3.3: Abschnitt 3 wurde durch Suchen nach „leeren“ Spalten in mehrere Unterabschnitte unterteilt. Die Aufteilung ist durch die orangefarbigen Linien gekennzeichnet.

Etwas schwieriger gestaltet sich die Aufteilung von Abschnitt 2, da es hier keine völlig „leeren“ Zeilen und Spalten gibt. Bei genauerer Betrachtung stellt man aber leicht fest, daß eine Aufteilung vorgenommen werden kann, wenn die in Abbildung 3.5 mit roten Pfeilen gekennzeichneten Stellen gefunden werden. Von diesen Stellen ausgehend können Aufteilungen vorgenommen werden. Die markierten Stellen zeichnen sich dadurch aus, daß auf wenige schwarze Bildpunkte (Rahmen) zunächst viele weiße Bildpunkte folgen. Wenn nun also zeilen- und spaltenweise nach dem Auftreten solcher Sequenzen gesucht wird, läßt sich eine vollständige Aufteilung des Zahlenfeldes in Unterabschnitte erreichen. Zu beachten ist, daß die Suche nach den beschriebenen Sequenzen nur in den weiß unterlegten Zahlenfeldern funktioniert. Daher muß die Suche nach den Sequenzen von allen Seiten des Zahlenfeldes erfolgen, um eine vollständige Aufteilung zu erreichen.

Danach ist die Identifizierung der Abschnitte vollendet.

Abbildung 3.4: Die Aufteilung des gesamten Scheins (bis auf die Unterteilung von Abschnitt 2) in Abschnitte wurde in sechs Schritten durchgeführt. Die einzelnen Aufteilungen sind durch verschiedene Farben gekennzeichnet.

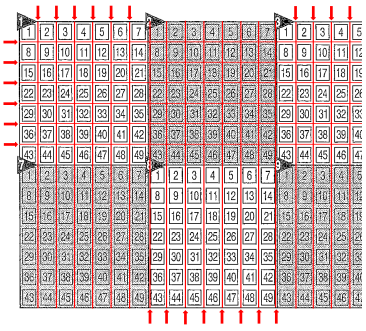


Abbildung 3.5: Die mit roten Pfeilen gekennzeichneten Stellen im Zahlenfeld sind aufzuspüren. Dort muß eine Unterteilungslinie eingezogen werden.

Nun werden die markierten Bereiche bestimmt. Dazu wird aus den Scans eines unausgefüllten und eines ausgefüllten Lottoscheins ein Differenzbild berechnet, in dem nur die Bildpunkte schwarz sind, an denen sich die ursprünglichen Bilder unterscheiden. Schließlich wird im Differenzbild nach zusammenhängenden schwarzen Bildpunkten gesucht und die Lage dieser Punkte bestimmt. Der Abschnitt, in den diese Punkte fallen, wird als markiert betrachtet.

Es wird erwartet, daß die Darstellung der ausgelesenen Daten in einem Fenster erfolgt. Die oben beschriebenen Funktionen sollen über eine Menüleiste abrufbar sein.

Abzuspeichern sind die ausgelesenen Daten in folgendem Format:

```
feld1=7,8,10,23,27,40
feld2=1,2,10,36,37,38
super6=ja
spiel77=nein
laufzeit=m
```

Jede Zeile wird mit einem CR abgeschlossen. Die obigen Daten stehen für einen Schein, in dem nur die ersten beiden Felder Kreuze enthalten und in dem die Spiele „Super 6“ mit „Ja“ und „Spiel77 mit „Nein“ angekreuzt wurden. Die Laufzeit des Scheins beträgt einen Monat.

3.3 Testen

Bitte testen Sie Ihr Programm ausgiebig. Neben den in der Aufgabenstellung beschriebenen Anforderungen sollten Sie das von Ihnen erstellte Programm noch auf folgende, weitere Punkte überprüfen:

- Erkennt das Programm die zur Verfügung gestellten Lottoscheine 1 bis 5?
- Werden die Spielregeln befolgt (bzgl. der Anordnung der Felder und dem Setzen der Kreuze)?
- Wird das vorgegebene Format beim Speichern eingehalten?

3.4 Material

Auf der Webseite des LS PI4 steht Ihnen folgendes Arbeitsmaterial zur Verfügung:

- Scans von leeren und ausgefüllten Lottoscheinen (im BMP- und RAS-Format)
- Spezifikation von BMP- und RAS-Files

4 Puzzle

4.1 Einführung

Die Puzzle-Aufgabe ist die Computer-Umsetzung eines Spiels, das einige vielleicht noch aus ihrer Kinderzeit kennen. Es geht dabei darum, eine vorgegebene Fläche mit Spielsteinen unterschiedlicher Form so überlappungsfrei zu füllen, daß die Fläche vollständig bedeckt wird. Die Spielsteine selbst sind durch Geraden begrenzt, haben nur rechte Winkel und dürfen keine Löcher haben. Im Originalspiel ist die Fläche quadratisch, hier sollen aber auch andere Formen zugelassen werden. Ein Eindruck vom Spiel wird in Abbildung 4.1 vermittelt. Das Spielfeld und die Steine sind hier sehr an das Originalspiel angelehnt.

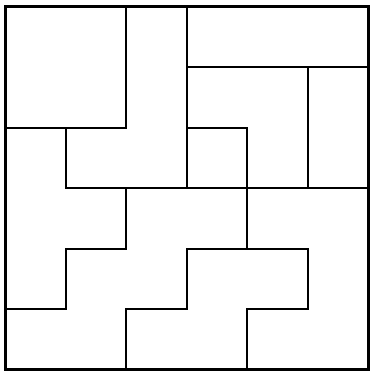


Abbildung 4.1: Dies ist ein bereits zusammengelegtes Puzzle.

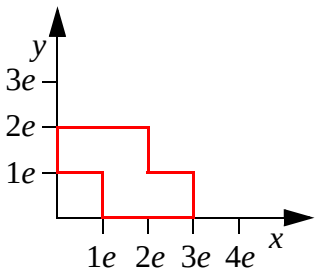
4.2 Aufgabenbeschreibung

Die Aufgabe besteht nun darin, sowohl einen Editor für Spielfeld und Spielsteine, als auch das eigentliche Spiel zu implementieren.

Mit dem Editor muß es möglich sein, zunächst das Spielfeld zu definieren. Dabei gelten hierfür dieselben Regeln wie für Spielsteine auch: Feld und Steine sind durch Geradenabschnitte („Segmente“) begrenzt, haben nur rechtwinklige Ecken und dürfen keine Löcher haben. Zudem sind für die Länge sämtliche Begrenzungen nur Vielfache einer Grundeinheit *e* erlaubt. D.h. die kleinstmögliche Form ist *e* * *e*. Das in Abbildung 4.1 dargestellte Spielfeld hat folglich die Größe 6*e* * 6*e*, wenn für die Höhe des Spielsteins

oben rechts der Wert *e* angenommen wird (wenn die Höhe dieses Steins 2*e* ist, ist die Spielfeldgröße natürlich 12*e* * 12*e*). Nach der Definition des Spielfeldes sollte die Spielfläche „zerschnitten“ werden können. Dadurch entsteht eine Menge von Spielsteinen, die die Fläche ausfüllt. Vom Editor ist dabei zu prüfen, ob irreguläre Steine entstehen, was er dann verhindern muß. Gesetzte Schnitte sollten auch wieder rückgängig gemacht werden können, d.h. es muß eine unbegrenzte Undo-Funktion geben. Ein Wert für die Spielzeit in Sekunden, die für das editierte Szenario empfohlen wird, muß außerdem eingegeben werden können. Letztlich soll das Szenario im XML-Format abgespeichert werden.

Das XML-File besteht dabei im wesentlichen aus zwei Teilen, nämlich dem Spielfeld und der Menge der Spielsteine. Feld und Steine werden in die Menge der Kanten zerlegt, die sie formen. Um diese Menge für ein Objekt zu berechnen, wird es zunächst derart in ein Koordinatensystem gelegt, daß es mit seiner linken Seite von rechts an die y-Achse und mit der unteren Seite von oben an die x-Achse stößt. Der Ursprung des Koordinatensystems liegt hier unten links. So wird verhindert, daß für die Koordinaten der darstellenden Segmente negative Werte auftreten. Für jedes dieser Segmente werden nun Anfangs- und Endpunkt bestimmt und die Menge der Segmente für das Objekt abgespeichert. In Abbildung 4.2 ist zu sehen, wie eine solche Menge für *e* = 1 aussieht. Die Reihenfolge



line ((1, 0), (3, 0)),
line ((3, 0), (3, 1)),
line ((2, 1), (3, 1)),
line ((2, 1), (2, 2)),
line ((0, 2), (2, 2)),
line ((0, 1), (0, 2)),
line ((0, 1), (1, 1)),
line ((1, 0), (1, 1))

Abbildung 4.2: Die rechts notierten *line*-Objekte repräsentieren die Menge der das dargestellte Objekt begrenzenden Segmente.

der Segmente ist hierbei beliebig. Eine DTD, die das Format festlegt, ist auf den Seiten des Lehrgebiets zu finden. Für Informationen zu XML sei hier auf die XML-Seiten des Word-Wide-Web-Consortiums (W3C) verwiesen (http://www.w3.org/xml). Beispielhaft für XML-Literatur sei hier außerdem auch das online-Buch von Behme und Mintert erwähnt (http://www.mintert.com/xml/buch/). Zum Erzeugen von XML-Daten und dem Abspeichern und Laden derselben stehen Java-Bibliotheken zur Verfügung, die benutzt werden dürfen (http://java.sun.com/xml/).

Im Spiel selbst hat der Benutzer nun die Möglichkeit, *Szenarien*, die mit dem Editor erstellt wurden, zu laden. Als Szenario wird dabei die Gesamtheit von Spielfeld, Spielsteinen und Spielzeit bezeichnet. Die Daten werden dann analysiert und dargestellt. Dabei muß die im File angegebene Feldgröße dem tatsächlich auf dem Bildschirm verfügbaren Platz angepaßt werden. Die Spielsteine werden dann neben dem Feld ungeordnet angezeigt. Per Drag&Drop kann der Spieler nun die einzelnen Teile anwählen und in der Spielfläche ablegen. Die Teile sollen dabei auf dem Raster, das durch *e* festgelegt ist, ein“rasten“. Um den Spielcharakter ein wenig zu fördern, gibt es ein Zeitlimit für das Lösen des Puzzles, das ebenfalls im XML-File abgespeichert wird. Nach dem Laden der Daten wird das Spiel über einen Knopf gestartet. Die Zeit wird deutlich sichtbar heruntergezählt, das Spiel endet, wenn die Zeit abgelaufen ist. Der Spieler hat dann die Möglichkeit, sich vom Programm eine Lösung des Puzzles anzeigen zu lassen. Da die Lösung nicht im XML-File gespeichert wird, muß sie vom Programm berechnet werden.

4.3 Optionale Funktionen

Folgende weitere Funktionen erweitern die Möglichkeiten des Programms und können auf freiwilliger Basis implementiert werden:

- Die Darstellung der Spielsteine kann, auch mit einfachen Mitteln, verbessert werden. Neben der Darstellung in verschiedenen Farben ist es möglich, den Eindruck von dreidimensionalen Steinen zu erzeugen. Bitmaps, die auf die Steine gelegt werden, können ebenfalls zur Verschönerung beitragen.
- Verschiedene Szenarien können zusammengefaßt und nacheinander durchgespielt werden. Informationen über die gruppierten Szenarien werden ebenfalls im XML-Format abgespeichert. Ein Spieler kann dann über eine Zusatzoption die Gruppe laden und spielen.
- Eine Highscore-Liste verwaltet die Wertungen der besten Spieler. Punkte können z.B. für die Restzeit vergeben werden. Die Liste sollte ebenfalls im XML-Format gespeichert werden. Eine Kombination dieser Option mit der vorigen Option fördert den Spielspaß deutlich.

5 Rollende Kugel

5.1 Überblick

Im Rahmen dieser Aufgabe soll ein System implementiert werden, das eine Kugel simuliert, die über eine Fläche im dreidimensionalen Raum rollt. Diese Fläche soll durch eine Funktion $f : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ beschrieben werden, die in einem kartesischen Koordinatensystem von (x, y) nach z abbildet.

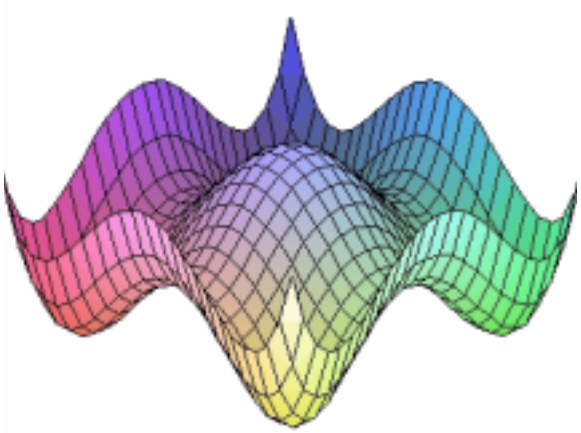


Abbildung 5.1: Durch ein Polynom beschriebene Fläche

Ein Beispiel sehen Sie in Abbildung 5.1. Diese Fläche wurde durch ein Polynom der Ordnung 4 in x und y beschrieben:

$$f(z) = \frac{1}{24}x^4 - \frac{2}{3}x^3 + \frac{10}{3}x^2 - \frac{16}{3}x + 1 + \frac{1}{24}y^4 - \frac{2}{3}y^3 + \frac{10}{3}y^2 - \frac{16}{3}y + 1, 0 < x < 8, 0 < y < 8$$

Polynome haben den großen Vorteil, daß Ihre Ableitungen nach x und y sehr leicht zu berechnen sind. Diese Ableitungen braucht man, um den Beschleunigungsvektor zu ermitteln, der ausschlaggebend für Richtung und Geschwindigkeit der rollenden Kugel ist.

Allerdings kann auch die Verwendung anderer Funktionen als nur Polynome sinnvoll sein. Beispielsweise können schwingende Formen gut mit Hilfe von Winkelfunktionen beschrieben werden. So sind die Wellen der Fläche in Abbildung 5.2 durch den Sinus-Anteil der Definition entstanden:

Es sind natürlich auch Möglichkeiten der Flächenbeschreibung denkbar, die über die Angabe einer einzigen Funktion hinausgehen. Eine sinnvolle Erweiterung wäre es, verschiedene Funktionen für verschiedene Bereiche der x-y-Ebene anzugeben. Allerdings

$$f(z) = 0{,}4\,x^4 - 50x^2 + 1000\left(\frac{\sin(2y)}{y}\right),\ -12 < x < 12,\ 0 < y < 12$$

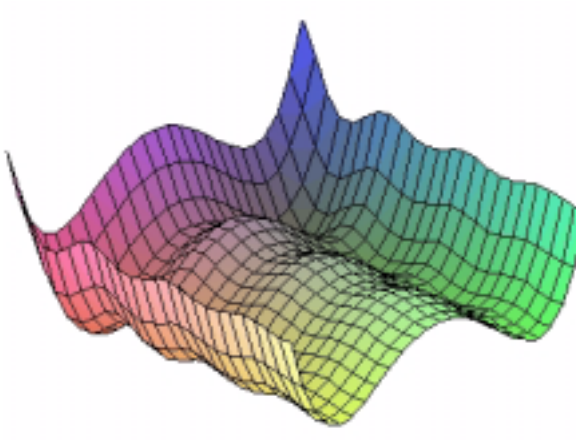


Abbildung 5.2: Unter Verwendung der Sinusfunktion beschriebene Fläche

müßte der Benutzer darauf achten, daß in der resultierenden Gesamtfläche keine Unstetigkeiten auftauchen und darüber hinaus die gesamten Fläche, insbesondere auch an den Übergangsstellen zwischen verschiedenen Funktionen, differenzierbar bleibt.[1] — Eine alternative, sehr benutzerfreundliche Möglichkeit der Flächendefinition besteht darin, den Benutzer beliebig viele Stützpunkte im dreidimensionalen Raum eingeben zu lassen, aus denen dann das System die Gesamtfläche interpoliert (z.B. mit Hilfe der Spline-Interpolation).

Auf eine solche Fläche soll der Benutzer nun eine Kugel positionieren, ihre Masse angeben und sie mit einer beliebigen Anfangsgeschwindigkeit und -richtung versehen können. Daraufhin simuliert das System, wie die Kugel sich entsprechend ihrer Position auf der Fläche bewegt. Im Rahmen des Programmierpraktikums muß die Simulation nicht allzu korrekt sein; eine einfache Näherungslösung, bei der z.B. die Kugel nicht von der Fläche abhebt, ist durchaus akzeptabel.

1. Natürlich wäre es auch denkbar, Flächen zu simulieren, die *Kanten* enthalten. Dies würde die Simulation jedoch zusätzlich erschweren und ist im Rahmen des Programmierpraktikums nicht erforderlich.

5.2 Mindestfunktionalität

Das zu erstellende Programm muß mindestens folgende Funktionalität aufweisen:

- Die Fläche im dreidimensionalen Raum kann entweder
  - durch ein Polynom definiert werden, dann ermittelt das System die erforderlichen Ableitungen, oder
  - durch eine beliebige Funktion beschrieben werden. In diesem Fall wird vom Benutzer verlangt, daß er auch die Ableitungsfunktionen nach x und y angibt. Hier ist es akzeptabel, daß die Funktionen als Java-Methoden codiert werden und nach einer Neueingabe der Funktionen neu kompiliert wird.
- Die Fläche wird graphisch dargestellt.
- Der Benutzer kann eine Kugel auf der Fläche positionieren, ihre Masse angeben und sie mit einer beliebigen Anfangsgeschwindigkeit und -richtung versehen.
- Das System zeigt, wie die Kugel über die Fläche rollt. Die Simulation kann relativ einfach sein. Insbesondere ist zulässig, daß die Kugel an der Fläche „klebt“ und bei der Berechnung nur ihr Schwerpunkt berücksichtigt wird. Sie darf also so in der Fläche versinken, daß ihr Mittelpunkt genau auf der Fläche liegt.
- Der Benutzer kann die Geschwindigkeit der Simulation steuern.

Diese Aufgabe geht über den üblichen Anspruch des Programmierpraktikums hinaus. Sie sollte nur bei Interesse an der zugrundeliegenden Physik und an der Entwicklung bewegter Graphiken gewählt werden. Dann bietet sie jedoch besonders viele Möglichkeiten, Kreativität zu entwickeln und eigene Ideen zu verwirklichen.

6 Räumliche Darstellung mit Stereobildern

6.1 Räumliches Sehen

Die folgenden beiden Aufgaben haben mit *räumlichem Sehen* zu tun. In beiden Aufgaben sollen Sie ein Programm schreiben, das auf dem Bildschirm oder einem Blatt Papier Darstellungen erzeugt, die Sie mit dem bloßen Auge betrachten und von denen Sie dabei einen räumlichen Eindruck gewinnen können.

Vor einigen Jahren sind Bücher populär geworden, u.a. mit Titeln „Das magische Auge“ und „Das magische Auge II“,[1] in denen große farbige Bilder abgedruckt sind, die zunächst wie verworrene Muster aussehen. In der Einleitung dieser Bücher wird erklärt, wie man diese Muster ansehen soll. Wenn man auf eine bestimmte Art durch das Blatt Papier hindurchsieht, kann man plötzlich eine räumliche Szene erkennen. Es bedarf allerdings einer gewissen Übung bzw. eines Lernprozesses. Eine Möglichkeit besteht darin, das Blatt zunächst dicht vor die Nase zu halten, dabei in die Ferne zu schauen und dann langsam den Abstand zwischen Augen und Blatt zu vergrößern.

Diese beiden Praktikumsaufgaben wenden sich an Leute, die von solchen Bildern fasziniert waren, oder sich jetzt damit beschäftigen wollen. Wenn es Ihnen allerdings nicht gelingt, in solchen Büchern oder auch in unseren Beispielen weiter unten irgend etwas Dreidimensionales zu erkennen, dann sollten Sie wohl besser eine andere Aufgabe wählen.

Wie ist es möglich, auf ein Blatt Papier zu sehen und eine 3D-Szene zu erkennen? Wir wollen uns kurz erinnern, wie räumliches Sehen funktioniert. Man braucht bekanntlich zwei Augen dazu. Es ist uns vielleicht nicht so bewußt, aber wir sehen tatsächlich in jedem Moment *zwei* Bilder, nämlich eins mit jedem Auge. Im Gehirn werden die beiden Bilder überlagert und es entsteht ein räumlicher Eindruck.

Die Entfernung, die wir wahrnehmen, entsteht dadurch, daß wir beide Augen auf einen Gegenstand fokussieren. Der Winkel zwischen den beiden „Sehstrahlen“ bestimmt diese Entfernung. Bei einem nahen Gegenstand ist der Winkel relativ groß. Bei einem weiter entfernten Gegenstand wird der Winkel kleiner (Abbildung 6.1). Bei einem sehr weit entfernten Gegenstand ist der Winkel nahezu 0; die Sehstrahlen sind fast parallel.

Wenn wir ein Blatt Papier oder einen Bildschirm betrachten, ist offensichtlich alles dort Dargestellte gleich weit entfernt. Der Trick, um in einer Papierebene ein räumliches Bild

1. Verlag Ars Edition, München 1994.

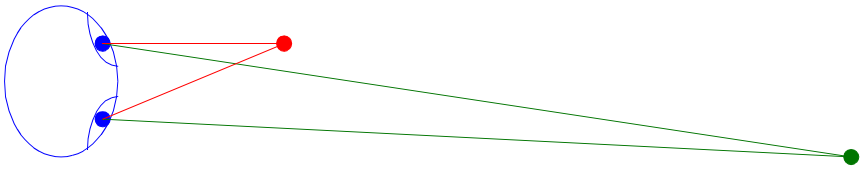


Abbildung 6.1: Betrachten eines nahen und eines weiter entfernten Gegenstandes

sehen zu können,[1] besteht darin, die Augen nicht auf die Papierebene, sondern auf einen Punkt in einiger Entfernung *dahinter* zu fokussieren (Abbildung 6.2). Natürlich muß die

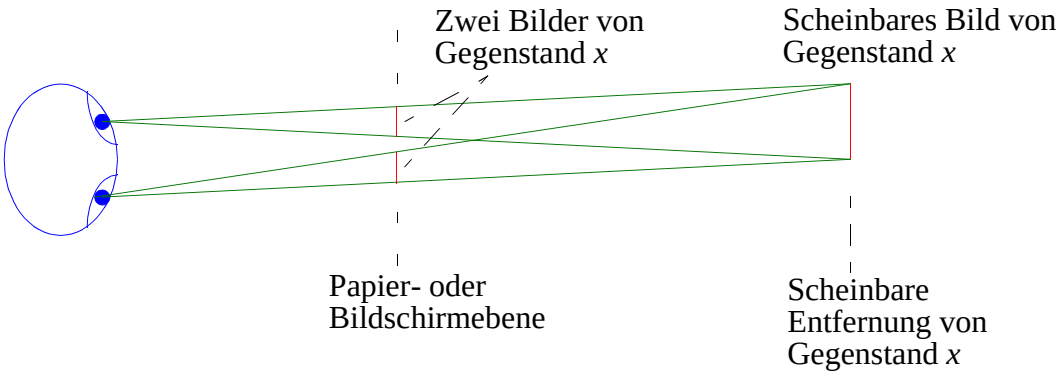


Abbildung 6.2: Räumliches Sehen mittels „Hindurchschauen“ durch die Papierebene

Darstellung in der Papierebene auch dafür gemacht sein. Sie muß zwei Bilder des gleichen Gegenstandes, aus leicht unterschiedlichen Blickwinkeln, in geringem Abstand enthalten. Abbildung 6.2 zeigt, von oben gesehen, zwei Darstellungen eines Gegenstandes *x*. Da die Augen in die Ferne gerichtet sind, überlagern sich die beiden Darstellungen und erzeugen ein scheinbares Bild von *x* in einiger Entfernung. Bei „normaler“ Betrachtung der Papierebene würde man jeweils nur auf eine Darstellung von *x* fokussieren und auch kein räumliches Bild sehen, wie in Abbildung 6.3 dargestellt.

Die scheinbare Entfernung des Gegenstandes *x* in Abbildung 6.2 hängt nun offensichtlich vom Abstand der beiden Darstellungen in der Papierebene ab. Geringfügiges Variieren dieses Abstandes wird die scheinbare Entfernung ändern. Das kann man sich zunutze machen, um räumliche Bilder zu erzeugen (Abbildung 6.4).

Abbildung 6.4 zeigt links zwei Bilder von drei übereinander angeordneten Rechtecken. Wenn Sie versuchen, durch die Rechtecke hindurch in die Ferne zu sehen, werden Sie (hoffentlich) nach einer Weile *drei* Gruppen von Rechtecken sehen (links, Mitte, rechts),

1. Zeichner und Maler benutzen auch noch andere Techniken wie Perspektive, blassere Farben im Hintergrund, usw.

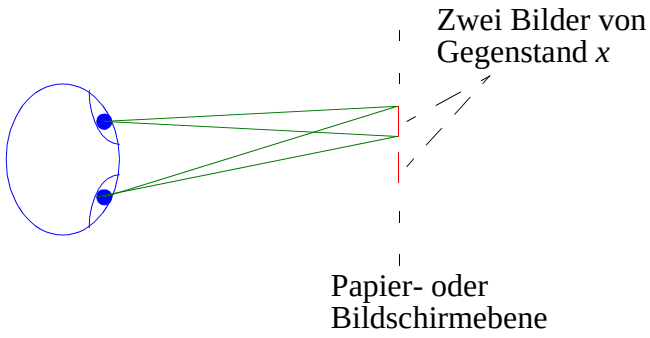


Abbildung 6.3: Normales Betrachten der Papierebene

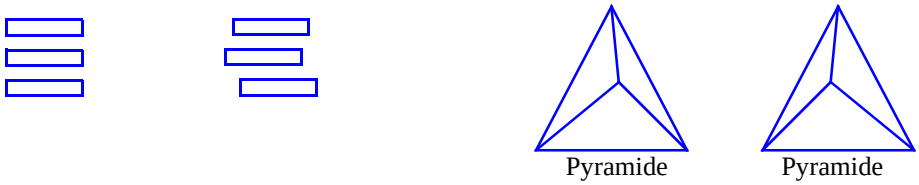


Abbildung 6.4: Stereobilder

die tatsächlich aus den *vier* Gruppen von Rechtecken entstehen, die Ihre Augen wahrnehmen (jedes Auge sieht zwei Gruppen). Die linke Gruppe wird nur von Ihrem linken Auge gesehen, die rechte nur vom rechten Auge. Die mittlere Gruppe aber entsteht durch Überlagerung des rechten Bildes vom linken Auge mit dem linken Bild vom rechten Auge und sie wirkt räumlich. Sie werden erkennen, daß die drei Rechtecke im mittleren Bild unterschiedliche Abstände von Ihnen als Betrachter zu haben scheinen.

Den gleichen Effekt kann man in der Abbildung 6.4 rechts bei der Überlagerung der beiden (von oben gesehenen) Pyramiden erzielen. Die Spitze der Pyramide liegt deutlich weiter vorn als die Grundfläche. Ebenso hat die Schrift „Pyramide“ eine andere Tiefe.

6.2 Aufgabenbeschreibung

In dieser Aufgabe soll ein Programm geschrieben werden, das zu einer gegebenen dreidimensionalen Szene die in Abschnitt 6.1 beschriebenen Stereobilder berechnet.

Die Szene wird dabei in dem weiter unten beschriebenen Format angegeben. Mittels perspektivischer Projektion sind dazu jeweils zwei Bilder zu berechnen, eines aus der Position des linken und eines aus der Position des rechten Auges. Für die Darstellung der Bilder sollen zwei Varianten implementiert werden, nämlich die in Abschnitt 6.1 beschriebene Variante mit zwei schwarz gezeichneten Bildern, als auch eine Variante mit zwei

Bildern in rot und grün (oder rot und blau), die mit einer speziellen Rotgrün-Brille (bzw. Rotblau-Brille) zu betrachten sind (siehe Abbildung 6.5). Es soll die Möglichkeit bestehen, die berechneten Stereobilder sowohl auf dem Monitor als auch auf dem Drucker auszugeben.

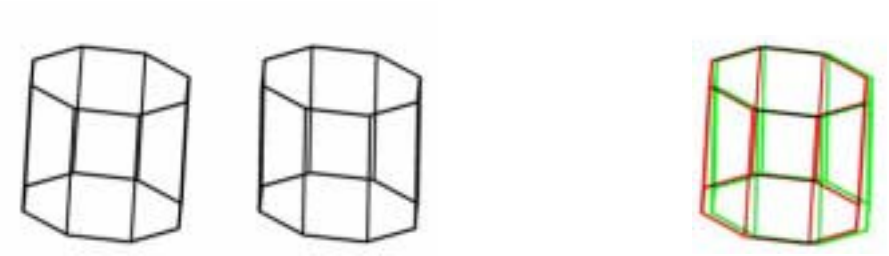


Abbildung 6.5: links die Darstellung als normales Stereobild und rechts die Darstellung als Rotgrün-Stereobild

6.3 Das Objektformat

Die Szenen, aus denen die Stereobilder zu berechnen sind, werden in einem bestimmten Format in Dateien gespeichert. Die geometrischen Objekte werden textuell als Mengen von Linienzügen in der Form

obj(line((x_0, y_0, z_0), (x_1, y_1, z_1)), ... , line((x_{m-1}, y_{m-1}, z_{m-1})(x_m, y_m, z_m))),
...
obj(line((x_0, y_0, z_0), (x_1, y_1, z_1)), ... , line((x_{n-1}, y_{n-1}, z_{n-1})(x_n, y_n, z_n)))

angegeben.

Ein einfacher Würfel wird also wie folgt dargestellt (siehe auch Abbildung 6.6):

```
obj(
   line((-1, -1, 1), (1, -1, 1)), line((1, -1, 1), (1, -1, 3)),
   line((1, -1, 3), (-1, -1, 3)), line((-1, -1, 3), (-1, -1, 1)),
   line((-1, 1, 1), (1, 1, 1)), line((1, 1, 1), (1, 1, 3)),
   line((1, 1, 3), (-1, 1, 3)), line((-1, 1, 3), (-1, 1, 1)),
   line((-1, -1, 1), (-1, 1, 1)), line((1, -1, 1), (1, 1, 1)),
   line((1, -1, 3), (1, 1, 3)), line ((-1, -1, 3), (-1, 1, 3))
)
```

Es ist die Aufgabe des Szene-Architekten, eine Szene zu erzeugen, die ausschließlich „sinnvolle“ geometrische Objekte enthält. Das zu erstellende Programm braucht in dieser Hinsicht keine Überprüfung vorzunehmen. Es ist allerdings zu prüfen, ob die Syntax

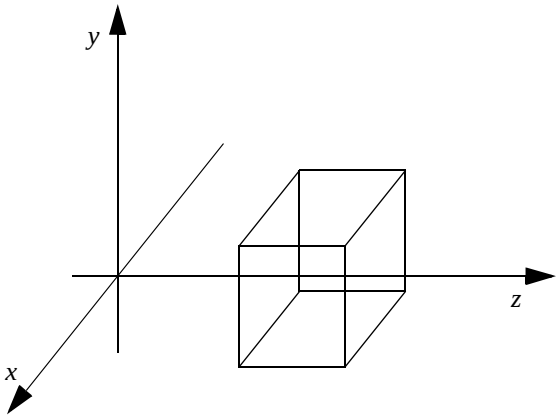


Abbildung 6.6: Der modellierte Würfel

des Files mit der Spezifikation übereinstimmt. Für fehlerhafte Beschreibungen sollen keine Berechnungen durchgeführt und stattdessen eine entsprechende Fehlermeldung ausgegeben werden.

6.4 Die perspektivische Projektion

Zum Verständnis der perspektivischen Projektion beschränken wir uns auf den zweidimensionalen Fall. Wie in Abbildung 6.7 dargestellt, haben wir einen Augenpunkt $A =$

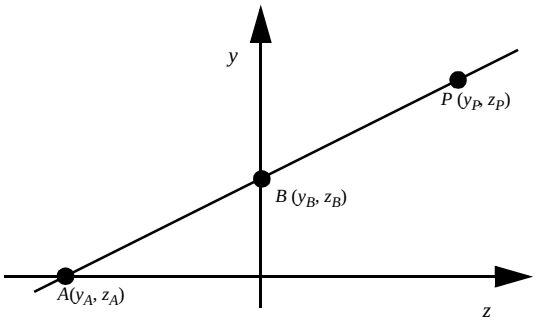


Abbildung 6.7: Die Gerade, die durch den Augenpunkt A und den zu projizierenden Punkt P festgelegt ist, schneidet die Projektionsgerade $z = 0$ im Punkt B.

(y_A, z_A) und einen zu projizierenden Punkt $P = (y_P, z_P)$. Die Projektionsgerade liegt bei $z = 0$. Diese Projektion läßt sich nun folgendermaßen berechnen: Sind der zu projizierende

Punkt $P = (y_P, z_P)$ und der Augenpunkt $A = (y_A, z_A)$ gegeben, so gilt nach dem Strahlensatz für die Koordinaten des Punktes $B = (y_B, z_B)$

$$\frac{y_B}{y_P} = \frac{z_A}{z_P + z_A}$$

Dieses Prinzip muß nun nur noch auf den dreidimensionalen Fall übertragen werden.

Beachten Sie für diese Aufgabenstellung folgende Besonderheiten (siehe Abbildung 6.8):

- Für die Augenpunkte A_l und A_r gilt $A_l = (x_A , 0, -z_A)$ und $A_r = (-x_A, 0, -z_A)$. Die Werte z_A (Abstand des Augenpaares von der Projektionsfläche) und $abst = 2 \cdot x_A$ (Abstand der Augen zueinander) müssen einstellbar sein.
- Die Projektionsebene ist die *xy*-Ebene.

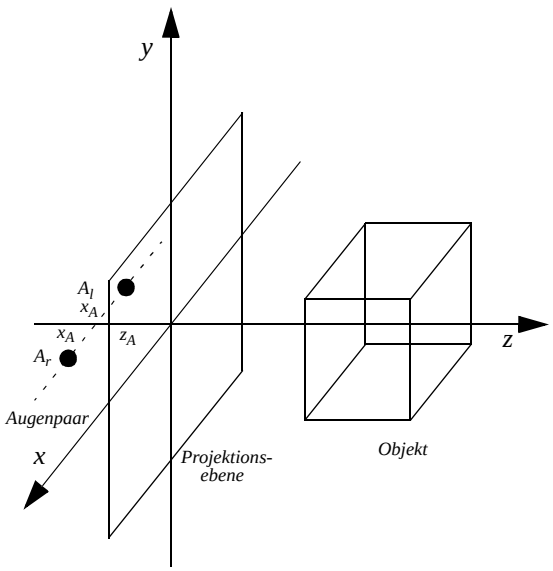


Abbildung 6.8: Die Anordnung von Augenpaar, Projektionsebene und zu projizierendem Objekt

Um ein Stereobild zu erhalten, muß also von jedem der beiden Augenpunkte aus eine Projektion berechnet werden. Die beiden Projektionen sind dann darzustellen. Der Abstand zwischen den dargestellten Bildern muß ebenfalls einstellbar sein. Der Unterschied zwischen den normalen Stereobildern und dem Rotgrün-Stereobild besteht im Wesentlichen in der unterschiedlichen Farbe und dem Abstand der beiden Teilbilder. Beachten Sie außerdem, daß bei den farbigen Bildern auch Mischfarben entstehen können.

6.5 Anforderungen

Neben den bereits oben aufgeführten Anforderungen bezüglich Objektformat und Einstellungsmöglichkeiten, soll eine graphische Benutzeroberfläche implementiert werden, die die folgenden Funktionen unterstützt:

- Auswählen und Prüfen einer Szene (als Datei von der Festplatte)
- Darstellen der ausgewählten Szene
- Umschalten zwischen normaler und rotgrüner Darstellung
- Modifikation der Werte für Augenabstand, Entfernung der Projektionsfläche und Abstand der dargestellten Bilder
- Druck der dargestellten Bilder
- Modifikation der Farbwerte (um Variationen in der Tönung unterschiedlicher Brillen auszugleichen)

Optional ist die Möglichkeit, das Augenpunktpaar beliebig hinter der Projektionsebene zu verschieben.

6.6 Material

Auf der Webseite des LS PI4 stehen Ihnen als Arbeitsmaterial einige bereits modellierte Szenen zur Verfügung.

7 Räumliche Darstellung durch Überlagerung repetitiver Muster

7.1 Einführung

Bitte lesen Sie, falls Sie dies bisher noch nicht getan haben, zunächst die einleitenden Bemerkungen zum *räumlichen Sehen* in Abschnitt 6.1. Die bisher beschriebene Methode verwendet zwei nebeneinander liegende Bilder, die durch Sehen in die Ferne übereinander geschoben werden. Ein Nachteil dabei ist, daß jedes der Bilder nur relativ klein sein kann, da man die Bilder nicht weiter als ca. 5 bis 6 cm verschieben kann.

Vor einigen Jahren ist die in Abschnitt 6.1 schon erwähnte Variante populär geworden, die diesen Nachteil vermeidet und es erlaubt, große räumliche Bilder zu sehen. Die Grundidee ist, ein Muster zu verwenden, das sich in der Ebene von links nach rechts wiederholt. Man schiebt dann mit den Augen jeweils zwei Instanzen des Musters übereinander. Kleine Variationen in den Abständen von Elementen des Musters erzeugen unterschiedliche Tiefenwirkungen und damit wiederum dreidimensionale Formen.

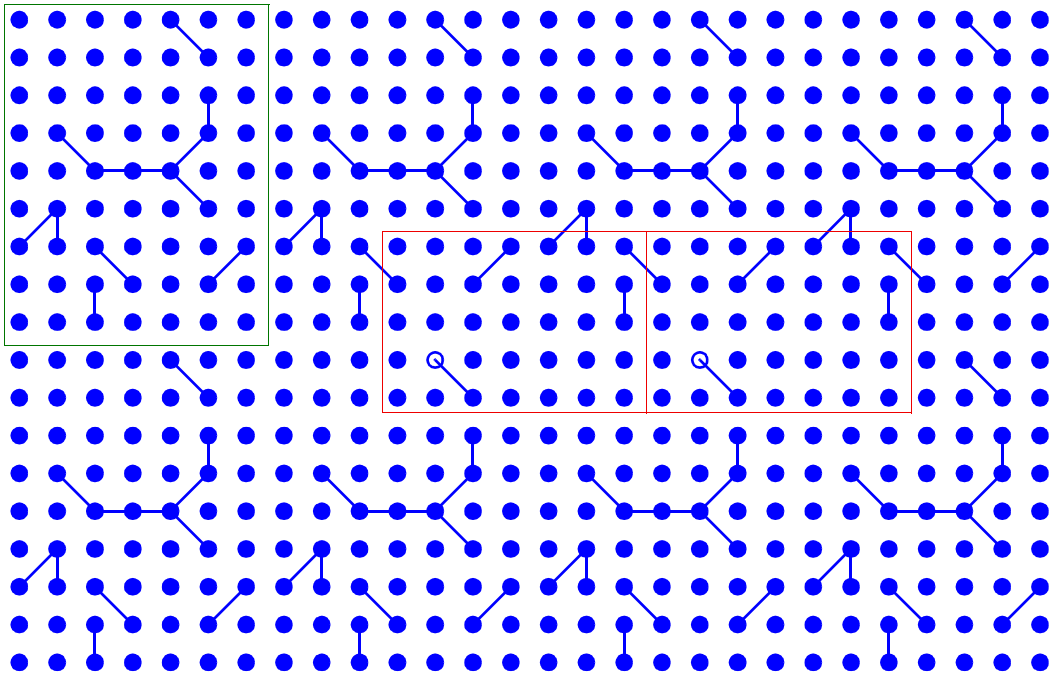


Abbildung 7.1: Ein Wiederholungsmuster

Ein Beispiel eines sich wiederholenden Musters zeigt Abbildung 7.1. Das Grundmuster ist links oben eingerahmt gezeigt; es wird dreimal nach rechts und einmal nach unten wiederholt. Es besteht an sich aus einer Matrix von Punkten; die Verbindungen zwischen den Punkten sind willkürlich hinzugefügt worden, um dem Auge beim „Übereinanderschieben“ Orientierung zu bieten. Man kann nun einen beliebigen Teil des Musters von der Breite des Grundmusters (hier: 7 Reihen Punkte) herausgreifen und die Wiederholung rechts davon (wie mit den beiden roten Rechtecken gezeigt); das Auge wird diese beiden Teile übereinanderschieben. Zwei sich entsprechende Punkte x und y solcher aufeinanderfolgender Teile, z.B. die beiden nicht gefüllten Punkte in Abbildung 7.1, nennen wir *Partner*. Sie bilden gemeinsam einen Punkt $P(x, y)$ in der 3D Sicht. Die Tiefe dieses Punktes $P(x, y)$, also die scheinbare Entfernung, wird durch den Abstand zwischen den Partnern x und y bestimmt.

Die Elemente des Musters (Punkte) haben zunächst alle den gleichen Abstand voneinander. Man kann nun räumliche Information in das Muster hineinkodieren, indem man die Abstände zwischen den Partnern geringfügig variiert. In Abbildung 7.2 haben wir einige

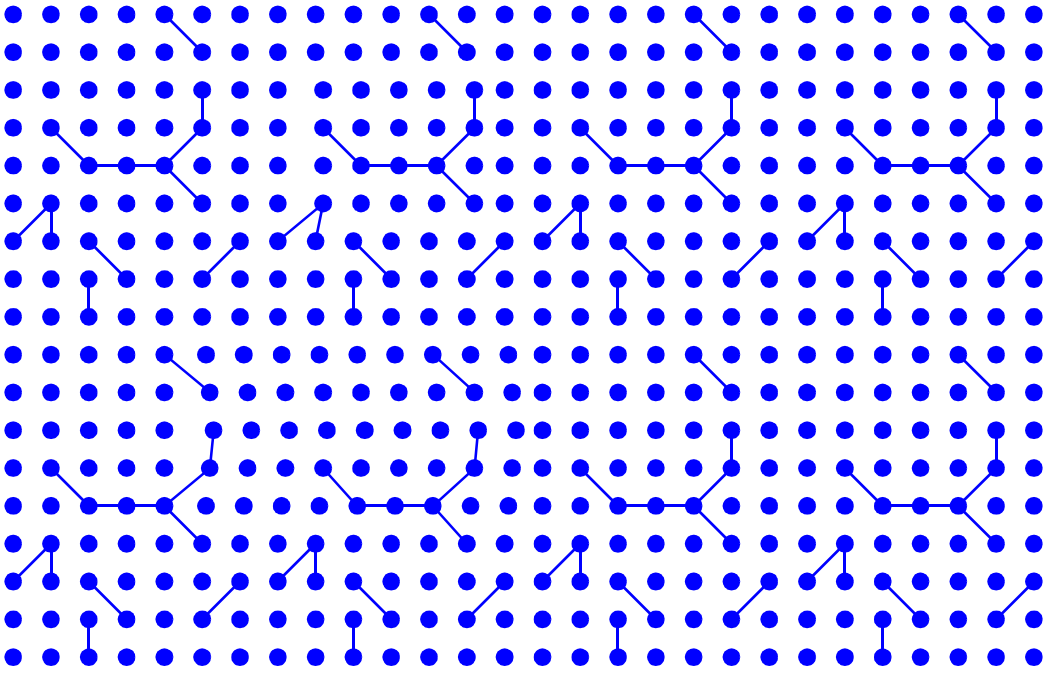


Abbildung 7.2: Wiederholungsmuster mit einkodierter räumlicher Information

Punktgruppen, unter Anpassung der inzidenten Kanten, etwas verschoben. Wenn Sie dieses Bild „räumlich“ betrachten, können Sie jeweils eine rechteckige Punktgruppe erkennen, die vor bzw. hinter der Ebene der anderen Punkte liegt, sowie jeweils ein nach vorne und ein nach hinten herausstehendes kleines „Dach“.

7.2 Aufgabenbeschreibung

Ihre Praktikumsaufgabe besteht nun darin, ein Programm zu schreiben, das solche sich wiederholende Muster mit einkodierter räumlicher Information berechnet. Die Auflösung sollte dabei wesentlich feiner sein als in den Abbildungen 7.1 und 7.2. Wir schlagen vor, daß Sie als Muster ein Punktraster mit einigen zusätzlichen Kanten verwenden, wie oben gezeigt; wenn Sie aber andere Ideen haben, wie man ein Muster wiederholen und seine „Verzerrungen“ systematisch berechnen kann, und die mindestens ebenso gute räumliche Darstellungen erzeugen, so ist das auch in Ordnung.

Die räumliche Darstellung, die in das Muster kodiert wird, kann man abstrakt auffassen als eine Funktion $z = f(x, y)$, wobei z die Tiefe eine Punktes (x, y) in einem Koordinatensystem wie in Abbildung 7.3 bezeichnet.

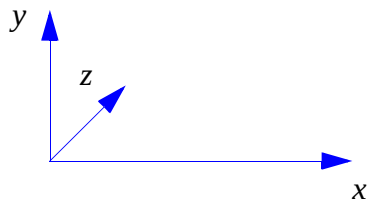


Abbildung 7.3: Koordinatensystem

Das ist so etwas wie eine Hügellandschaft über der xy-Ebene (ähnlich der Aufgabe in Abschnitt 5). Ihr Programm sollte zunächst in der Lage sein, einige solcher Funktionen darzustellen, z.B.

- $z = ax^2 + by^2$
- $z = \dfrac{a}{b|x| + c|y| + 1}$
- $z = a\sin(bx)$

wobei der Definitionsbereich der Funktion geeignet zu wählen ist, z.B. $-5 \le x \le 5$, $-5 \le y \le 5$, und der gesamte darzustellende Bereich am besten etwas größer ist, z.B. $-8 \le x \le 8$, $-8 \le y \le 8$.

Ihr Programm muß im wesentlichen für jeden Punkt Ihres Musters die Tiefe anhand dieser Funktion berechnen und anhand dessen den Partner-Punkt plazieren. Dabei kann man z.B. von links nach rechts das Muster durchlaufen und die Punkte setzen.

Ihr Programm sollte dem Benutzer eine Auswahl derartiger Funktionen anbieten, für die das Programm dann die Ansicht berechnet. Funktionsparameter wie a, b, c oben, könnte

der Benutzer noch interaktiv setzen. Zusätzlich könnte man auch weitere Funktionen im Programmtext einfügen und das Programm neu kompilieren.

Weiterhin sollte der Benutzer interaktiv eine Szene definieren können, indem er eine Reihe von einfachen Elementen in der 3D-Szene plaziert. Die anzubietenden Elemente sind:

- ein achsenparalleles Rechteck, das auch parallel zur *xy*-Ebene liegt, für das man also die Koordinaten (x_1, x_2, y_1, y_2, z) angibt.
- eine Dreiecksfläche in allgemeiner Lage, mit Eckpunkten (p, q, r), wobei jeder Eckpunkt im Raum beschrieben ist, also z.B. $p = (p_x, p_y, p_z)$

Abbildung 7.4 zeigt links eine aus Rechtecken aufgebaute Szene, rechts eine, die auch Dreiecke verwendet. Beachten Sie, daß man die Flächen in der räumlichen Darstellung sowieso nicht anhand ihrer Randlinien wahrnimmt; insofern schadet eine Zerlegung einer Fläche in mehrere Dreiecke nicht. Falls mehrere Flächen hintereinander liegen, sollte Ihr Programm jeweils den Punkt des zu erzeugenden Musters entsprechend der am weitesten vorn liegenden (also sichtbaren) Fläche plazieren; damit werden automatisch weiter hinten liegende Flächen verdeckt.

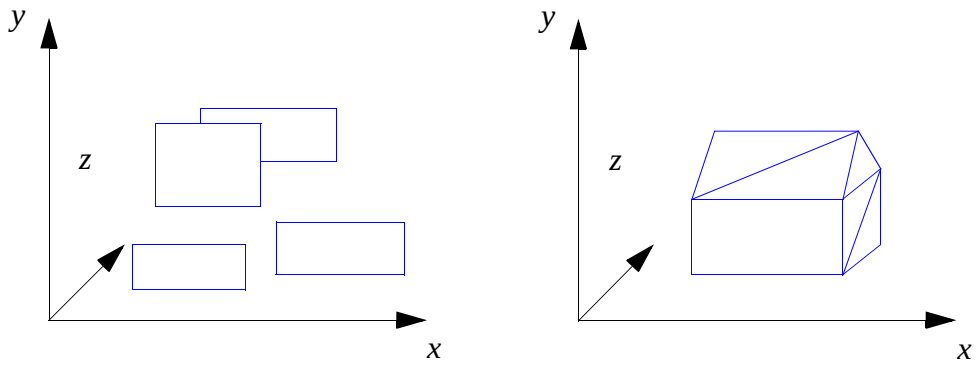


Abbildung 7.4: Aus Rechtecken und Dreiecksflächen aufgebaute Szenen

7.3 Mindestanforderungen

Das zu erstellende Programm muß mindestens folgende Funktionalität besitzen:

- Eine in der oben beschrieben Weise in ein Muster kodierte dreidimensionale Szene kann am Bildschirm dargestellt sowie auf einem Tintenstrahl- oder Laserdrucker ausgedruckt werden.

- Für die Beschreibung der Szene kann der Benutzer aus einigen „festverdrahteten“ Funktionen wählen. Insbesondere sollten die 3 oben genannten Funktionen wählbar sein. Funktionsparameter wie a, b, c sind interaktiv einstellbar.
- Der Benutzer kann eine Szene interaktiv durch Eingabe von Rechteck- und Dreiecksflächen aufbauen. Numerische Eingabe der Koordinaten ist ausreichend.
- Eine solche Szene kann in einem geeigneten Textformat in einer Datei abgespeichert und daraus gelesen werden. Man kann die Szene auch direkt in der Datei definieren. Mindestens zwei Szenen ähnlich denen aus Abbildung 7.4 sollten ladbar sein.

8 Programmierrichtlinien

Beim Programmentwurf sind folgende Punkte zu beachten:

- **Vollständigkeit**. Die in den Aufgabenstellungen beschriebenen Mindestanforderungen müssen implementiert sein.
- **Korrektheit**. Das Programm muß sich immer den Vorgaben entsprechend verhalten. Ein Programmabsturz muß ausgeschlossen sein, so unsinnig die Benutzereingaben auch sein mögen.
- **Verständlichkeit**. Das Programm muß so einfach zu verstehen sein, daß es später von einem anderen Programmierer ohne großen Aufwand gepflegt werden kann. Ein wichtiges Konzept, um dies zu erreichen, ist die
- **Modularisierung**. Unterteilen Sie ein großes Problem solange in mehrere kleinere, bis jedes Teilproblem klein genug ist, um auf einfache Weise gelöst zu werden.
- **Effizienz**. Ihr Programm sollte effizient bezüglich Ausführungszeit und Speicherplatzbedarf sein. Dies darf jedoch keinesfalls zu Lasten der anderen Forderungen gehen.
- **Portierbarkeit**. Das Programm muß auf anderen PCs kompilierbar und lauffähig sein. Verwenden Sie deshalb bitte nur JAVA™ SDK, Standard Edition, Version 1.4.1. Verweisen Sie nur auf solche Dateien, die von Ihnen selbst erzeugt wurden, und benutzen Sie dabei niemals absolute Pfadnamen.

Insbesondere die Forderung nach Verständlichkeit hat nicht nur Auswirkungen auf den Entwurf, sondern auch auf den Programmcode:

- Namen von Klassen, Methoden und Variablen müssen immer eine sinnvolle Bedeutung haben. Außerdem erhöht es die Lesbarkeit von Programmen, Regeln bezüglich der Form von Namen zu definieren:
  - Variablen- und Methodennamen beginnen immer mit einem Kleinbuchstaben.
  - Klassennamen beginnen immer mit einem Großbuchstaben.
  - Diese Regeln sind recht willkürlich gewählt, haben sich jedoch bewährt.
- Achten Sie auf die Länge der Methoden. Ab einer gewissen Länge (z.B. 40 Zeilen) sollten Sie die Methode gemäß der Forderung nach Modularisierung unterteilen.

Beschränken Sie sich auf die mit dem JAVA™ SDK, Standard Edition, Version 1.4.1 mitgelieferten Pakete. Alle über diese Pakete hinausgehenden Funktionalitäten sind eigenständig zu implementieren. Es ist allerdings dringend anzuraten, in den Paketen

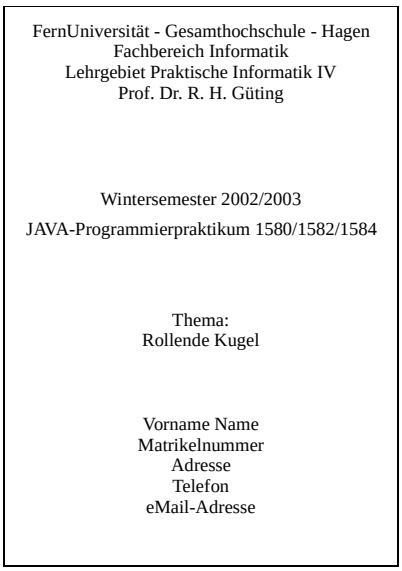
zunächst nach Methoden mit bestimmter Funktionalität zu suchen, bevor sie eigenständig implementiert werden.

Erstellen Sie einen einfachen objektorientierten Entwurf *bevor* Sie mit der Implementierung beginnen. Strukturieren Sie dazu das gestellte Problem und teilen Sie es in Einheiten auf, die einzelne Teilprobleme behandeln. Außerdem soll ein Klassendiagramm erstellt werden, in dem die Abhängigkeit der Klassen untereinander und die Zugehörigkeit zu verschiedenen Einheiten deutlich wird.

9 Dokumentationsrichtlinien

Eine vollständige Dokumentation besteht aus folgenden Teilen:

1. Deckblatt (Abbildung 9.1 zeigt ein Beispiellayout)
2. Eine max. zweiseitige Anleitung zu Installation und Aufruf Ihres Programmes
3. Ein kurze Bedienungsanleitung zur Benutzerschnittstelle
4. Überblick über die in Ihrem Programm zur Lösung der Aufgabe verwendeten Konzepte (objektorientierter Entwurf, Datentypen, Algorithmen) im Umfang von zwei bis drei DINA4-Seiten
5. Kommentierter Quelltext


FernUniversität - Gesamthochschule - Hagen
Fachbereich Informatik
Lehrgebiet Praktische Informatik IV
Prof. Dr. R. H. Güting

Wintersemester 2002/2003

JAVA-Programmierpraktikum 1580/1582/1584

Thema:
Rollende Kugel

Vorname Name
Matrikelnummer
Adresse
Telefon
eMail-Adresse

Abbildung 9.1: Ein Beispiel für ein Deckblatt

Das gesamte Dokument ist mit fortlaufenden Seitenzahlen zu versehen.

Zusätzlich sollen Sie eine Dokumentation Ihrer Implementierung mit *javadoc* (im SDK enthalten) erstellen. Die von *javadoc* erzeugten Dateien liefern Sie uns bitte auf Diskette oder CD und auch ausgedruckt.

Um den Anforderungen des Programmierpraktikums an eine gute Dokumentation zu genügen, muß der Quelltext einige Voraussetzungen erfüllen:

- Der Kopf jeder Methode wird um einen Kommentar ergänzt, der

  - die Bedeutung jedes Parameters erklärt, sofern sie nicht eindeutig aus dem Namen des Parameters hervorgeht.
  - die Aufgabe und Funktionsweise der Methode gut verständlich erläutert.
  - die Voraussetzungen an den Systemzustand beschreibt, unter denen diese Methode aufgerufen werden darf. Dies betrifft im wesentlichen den zulässigen Wertebereich von Eingabeparametern oder globalen Variablen und Annahmen über den vorherigen Aufruf anderer Methoden. So setzen beispielsweise Dateioperationen voraus, daß die betroffene Datei zuvor geöffnet worden ist.
- Jede Klasse ist mit einer Beschreibung zu versehen, aus der hervorgeht, welche Bedeutung diese Klasse hat und welche Methoden implementiert sind.
- Zusätzlich wird jede erklärungsbedürftige Anweisung um einen leicht verständlichen Kommentar ergänzt.
- Jede Programmzeile enthält immer nur eine Anweisung.
- Schachtelungstiefen von Anweisungsblöcken sind durch entsprechend tiefe Einrückungen zu visualisieren.
- Logisch zusammengehörende Anweisungsteile müssen als solche erkennbar sein. Dies kann durch Kommentare oder sinnvolles Einfügen von Leerzeilen erreicht werden.