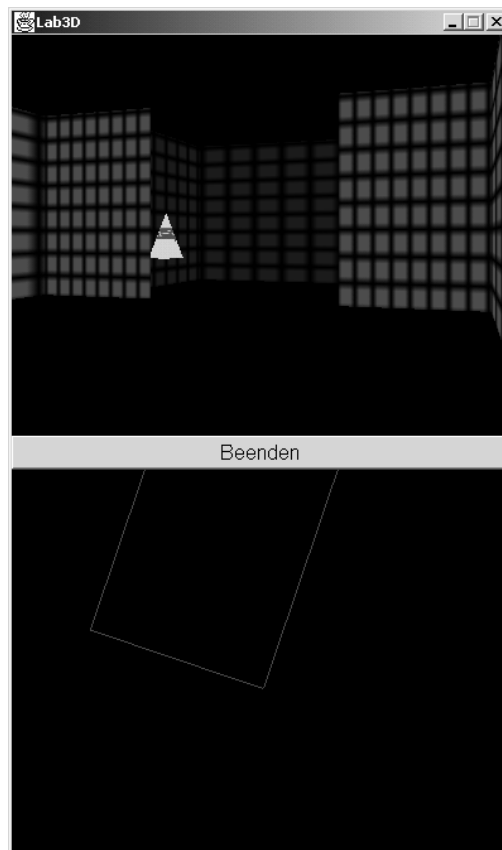


„Capture the flag“

Entwurf und Realisierung eines Spiels mit dreidimensionaler Graphik

Aufgabenstellung zum Software-Praktikum
im Sommersemester 2002



FernUniversität Hagen
Lehrgebiet Praktische Informatik II
58084 Hagen

Inhaltsverzeichnis

1. Einleitung	4
1.1. Ziel des Software-Praktikums	4
1.2. Das Projekt	4
1.3. Zur Implementierung	4
2. Was fordern wir?	4
3. Durchführung und Zeitplan des Software-Praktikums.....	5
4. Anforderungen zur Teilnahmeberechtigung an den Präsenzphasen	7
5. Anforderungen zum Erlangen des Leistungsnachweises	7
6. Organisatorisches	8
7. Kurzbeschreibung der Beispielprogramme.....	9
7.1. Java 3D	9
7.2. Ausführen des Beispielprogramms.....	9
7.3. Bedienung des Beispielprogramms	11
8. Spielbeschreibung.....	13
8.1. Bewegung auf einem horizontalen Raster	13
8.2. Labyrinth und Wände.....	13
8.3. Spielfeld und Felder.....	14
8.4. Arten von Feldern.....	14
8.5. Die Bewegung der Spieler durch das Labyrinth	14
8.6. Die Auswahl und Steuerung der Computergegner	14
8.7. Drei Ansichten auf das Labyrinth.....	15
8.8. Der Labyrinth-Editor	15
8.9. Das Dateiformat	16
8.10. Füllen der MyLabData-Klasse	16
9. Teilnahmevoraussetzung: Erstellung eines Prototypen	17
10. Dokumentation.....	17
10.1. Drei Arten der Dokumentation.....	17
10.2. Format der Dokumentation	18
10.3. Zur Dokumentierung des Sourcecodes.....	18
11. Anhang A: Java 3D	19

11.1.	Was ist Java 3D?	19
11.2.	Literatur	19
11.3.	Installation.....	19
11.4.	Körper, Oberflächen, Dreiecke und Punkte.....	19
11.5.	Transformationen	20
11.6.	Transformationen als Matrizen	21
11.7.	Anwendung einer Transformation auf einen Punkt.....	22
11.8.	Verknüpfung von Transformationen	23
11.9.	Transformationen in Java 3D	25
11.10.	Szenegraph	26
12.	Anhang B: Das Beispielprogramm	28
12.1.	Wiederholung: Aussehen und Bedienung des Beispielprogramms	28
12.2.	Die wichtigsten 5 Klassen.....	29
12.2.1.	Lab3D.java	29
12.2.2.	MySceneGraph_EgoEye, MySceneGraph_BirdsEye.....	30
12.2.3.	MyLab_EgoEye, MyLab_BirdsEye.....	32
12.3.	Kurze Beschreibung aller 18 Klassen	33
13.	Anhang C: javadoc.....	35
14.	Anhang D: Kurzeinführung in UML-Klassendiagramme	37

1. Einleitung

1.1. Ziel des Software-Praktikums

In dem Softwarepraktikum sollen Sie lernen, ein größeres Projekt nach Software Engineering-Prinzipien in einer Gruppe umzusetzen.

1.2. Das Projekt

In diesem Software-Praktikum sollen Sie (mit Ihrer Gruppe) das Spiel „Capture the Flag“ programmieren. Gegeben ist ein Labyrinth mit Flaggen. Ein menschlicher Spieler¹ tritt dabei gegen einen oder mehrere vom Computer gesteuerte Gegner an.

Ziel des Spiels ist es, die im Labyrinth verteilten Flaggen zu erobern. Die zu Beginn des Spiels neutralen Flaggen werden durch die Berührung eines Spielers in einer Farbe eingefärbt, die ihn repräsentiert. Einmal eroberte Flaggen können jederzeit von einer anderen Partei durch Berührung zurückerobert werden. Dabei wird die Flagge entsprechend eingefärbt.

Bei n Spielern (ein menschlicher Spieler und $n-1$ Computergegner) und f Flaggen, hat derjenige Spieler gewonnen, der als erster mehr als f/n Flaggen erobert hat. Damit ist das Spiel beendet.

Das Spiel läuft weitestgehend in einer dreidimensionalen Ansicht.

1.3. Zur Implementierung

Die zu verwendende Programmiersprache ist Java. Zur Darstellung des Labyrinths wird die Software-Bibliothek Java 3D verwendet. (Zu Java 3D siehe Abschnitt 7.1 und Anhang A.)

2. Was fordern wir?

Als am Software-Praktikum teilnehmender Studierender müssen Sie folgende Voraussetzungen erfüllen:

- Sie müssen mit Java vertraut sein. Beispielsweise sollten Begriffe in Java wie Canvas, Frame, Vector, Exception und Thread bekannt sein.
- Sie sollten die Bereitschaft mitbringen, sich in die Software-Bibliothek Java 3D einzuarbeiten. Um Ihnen über die ersten Probleme hinwegzuhelfen, stellen wir ein Beispielprogramm zur Verfügung, das einen Teil der Funktionalität des zu implementierenden Spiels bietet. Um das Spiel zu implementieren, bietet es sich an, das Beispielprogramm zu erweitern. Kapitel 7.2 zeigt, wie das Beispielprogramm installiert und gestartet wird; Anhang B zeigt, aus welchen Klassen das Programm besteht und wie diese zusammenhängen.

Zur Unterstützung Ihrer Einarbeitung in Java 3D führt Anhang A in Java 3D ein. Um jedoch mit Java 3D Programme zu erstellen, ist eine wesentlich intensivere Auseinandersetzung mit Java 3D notwendig. Hierzu liegt der vorliegenden Aufgabenstel-

¹ Auf die Nennung der weiblichen Form wird der Einfachheit halber verzichtet.

lung die Kurseinheit „Grafik-Bibliotheken“ aus dem Kurs 1692 „Graphische Datenverarbeitung I“ bei.

Wir setzen kein Wissen aus dem Bereich „graphische Datenverarbeitung“ voraus; Sie sollten sich jedoch damit vertraut machen, wie Transformationen (Verschiebungen, Drehungen und Skalierungen) von visuellen Objekten in 3D mit Java 3D realisiert werden. Dies wird in den Abschnitten 11.5 bis 11.9 erläutert.

3. Durchführung und Zeitplan des Software-Praktikums

Das Software-Praktikum gliedert sich in folgende Phasen:

1. Wir schicken Ihnen mit dieser Aufgabenstellung folgendes zu:
 - die Aufgabenstellung zum Software-Praktikum, also das vorliegende Dokument,
 - eine CD-ROM mit
 - i. dem Java SDK 2, Java 3D (für Windows) inklusive HTML-Dokumentation zu Java und zu Java 3D
 - ii. dem Beispielprogramm,
 - iii. der Kurseinheit „Grafik-Bibliotheken“ aus dem Kurs Nr. 1692 „Graphische Datenverarbeitung I“ und
 - iv. einem Java 3D Tutorial.

2. Sie erstellen zu Hause einen Prototyp und einen Entwurf des Spiels in Einzelarbeit. Weitere Informationen finden Sie dazu im Kapitel 4.

Spätestens bei der Erstellung dieses Prototyps beschäftigen Sie sich intensiv mit der vorliegenden Aufgabenstellung und dem Beispielprogramm. Sie arbeiten sich soweit in Java 3D ein, dass Sie den Prototypen erstellen können.

Zusätzlich analysieren Sie die Anforderungen des Spiels so weit, dass Sie einen Entwurf des endgültigen Spiels erstellen können.

3. Sie übergeben den Prototypen und einen Entwurf des Gesamtsystems Ihrem Betreuer. Wir empfehlen, dem Betreuer die Ergebnisse per E-Mail zukommen zu lassen. Als Formate für die Texte akzeptieren wir Postscript und PDF. Den Quellcode Ihres Prototypen schicken Sie uns bitte in einer ZIP-Datei als Email oder auf Diskette.

Unsere E-Mail-Adressen finden Sie in Kapitel 6. Möchten Sie uns Ihre Unterlagen auf dem Postweg zukommen lassen, senden Sie diese bitte an :

FernUniversität Hagen

Praktische Informatik II

Kennwort: SOPRA

58084 Hagen

Einsendeschluss: **Bis zum 24.05.2002** (Datum der E-Mail oder des Poststempels). Bitte nehmen Sie diesen Termin sehr ernst. Verspätet eingereichte Einsendungen werden nicht mehr berücksichtigt.

4. Ihre Betreuer beurteilen die Prototypen und den Entwurf und informiert Sie darüber, ob Sie an den folgenden Phasen des Software-Praktikums teilnehmen können.

Termin der Benachrichtigung: **Bis zum 07.06.2002.**

5. Erste Präsenzphase:

Termin: **Vom 29.06. bis zum 07.07.2002.**

In dieser Phase wird die Einzelarbeit von der Teamarbeit abgelöst.

Die Teilnehmer des Software-Praktikums kommen an der FernUniversität Hagen zusammen. Dort können Sie Fragen an die Betreuer stellen. Sie diskutieren Details des Programms, insbesondere die Funktionsweise, den Funktionsumfang und die Bedienung des Programms und treffen dann die diesbezüglichen Entscheidungen.

Sie unterteilen das zu erstellende Programm in Module (bzw. Klassen und Packages) und verteilen diese auf die Gruppenmitglieder. Sie legen die Schnittstellen der Module fest, das heißt Klassen und Methoden mit dem Attribut `public` oder `protected`. Bei den betreffenden Methoden ist der Methodenkopf inklusive Parameter, Rückgabe-Datentyp und Typ der eventuell erzeugten Exceptions zu definieren, bei einer Klasse muss man sich auf ihren Namen, Superklasse, Interfaces und ähnliches einigen.

6. In Heimarbeit implementieren und testen die einzelnen Teilnehmer ihre Module. Dabei unterstützen sie sich bei Fragen und Problemen.

Wir empfehlen Ihnen, auch schon in dieser Phase die bereits implementierten und separat getesteten Module zumindest probeweise zusammenzufügen.

7. Zweite Präsenzphase:

Termin: **Vom 02.09. bis zum 06.09.2002.**

Die Teilnehmer kommen abermals an der FernUniversität Hagen zusammen. Sie fügen die Module zum vollständigen Programm zusammen (Integration), präsentieren ihre Lösung in einem Vortrag mit Folien oder anderen Medien und führen ihr Programm vor. Der Betreuer beurteilt den Vortrag und das Programm. Eventuell führt er Gespräche mit einzelnen Teilnehmern. Die Teilnehmer stellen dem Betreuer eine endgültige Version des fertigen Programms zur Verfügung.

8. Die Gruppe erstellt eine Bedienungsanleitung für das Programm und eine Gesamtdokumentation.
9. Die Gruppe sendet uns die Dokumentation zu.

Termin: **20.09.2001.**

Der Betreuer beurteilt die Dokumentation. Wenn es keine Korrekturwünsche bezüglich der Dokumentation gibt und falls das Software-Praktikum erfolgreich verlaufen ist, werden die Leistungsnachweise ausgestellt und den Teilnehmerinnen und Teilnehmern zugeschickt.

4. Anforderungen zur Teilnahmeberechtigung an den Präsenzphasen

Sie müssen folgende Leistungen erbringen, um an den Präsenzphasen des Praktikums teilnehmen zu können:

- Implementieren Sie einen Prototypen des Spiels, wie in Kapitel 9 beschrieben. Die zu kompilierende Datei muss „CaptureTheFlag.java“ heißen. Das Programm muss sich fehlerfrei mit dem Befehl „javac CaptureTheFlag.java“ übersetzen lassen. Der Aufruf „java CaptureTheFlag testlab.lab“ soll das Programm starten. Dabei bezeichnet „testlab.lab“ eine beliebige Labyrinthdatei (wie in Abschnitt 8.9 beschrieben). Das Programm muss fehlerfrei die in Kapitel 9 beschriebenen Anforderungen erfüllen.
- Erstellen Sie einen Grobentwurf des endgültigen Spiels (**nicht nur des Prototypen**). Geben Sie dazu zuerst die Anforderungen an das Programm an. Benutzen Sie hierzu die fünf Kategorien von Anforderungen, die im Kurs Software Engineering I (1793) eingeführt wurden: *operative Anforderungen*, *Qualitätsanforderungen*, *technische Anforderungen*, *Validitäts- und Wartungsanforderungen* sowie *Realisierungsanforderungen*. Entwerfen Sie dann ein Klassendiagramm der Anwendung in UML (siehe dazu Anhang D). Begründen Sie die wichtigsten Entwurfsentscheidungen.

5. Anforderungen zum Erlangen des Leistungsnachweises

Der Leistungsnachweis für das Software-Praktikum ist unbenotet. Für den Leistungsnachweis müssen Sie folgende Aufgaben erfüllen:

- Die in Kapitel 4 beschriebenen Anforderungen müssen von Ihnen erfüllt werden, um zur weitergehenden Teilnahme am Praktikum zugelassen zu werden.
- Sie müssen an der ersten Präsenzphase teilnehmen und sich aktiv an der Gruppenarbeit beteiligen,
- Sie müssen Ihre eigenen Module implementieren und testen.
- Sie müssen an der zweiten Präsenzphase teilnehmen, sich an dem Vortrag und/oder der Vorführung des von der Gruppe erstellten Programms sowie an der Integration der Module zu einem vollständigen Programm beteiligen.
- Es muss eine Bedienungsanleitung und eine Dokumentation gemäß Kapitel 10 und Anhang C abgeliefert werden.

6. Organisatorisches

Bereiten Sie bitte für die erste Präsenzphase eine Liste Ihrer Fragen vor. Während der ersten Präsenzphase werden dann alle noch offenen Fragen behandelt. Für die Diskussion wurde zusätzlich eine Diskussionsgruppe eingerichtet. Die Gruppe wird auf unserem Server `feunews.fernuni-hagen.de` gehalten und heißt `feu.informatik.kurs.1585`.

Praktikumsbetreuer:



Dipl.-Inform. Dominic Heutelbeck
Tel. 02331-987-2923
E-Mail: dominic.heutelbeck@fernuni-hagen.de



Dipl.-Inform. Stephan Lukosch
Tel. 02331-987-4117
E-Mail: stephan.lukosch@fernuni-hagen.de



Dr. Jörg Roth
Tel. 02331-987-2134
E-Mail: joerg.roth@fernuni-hagen.de

Bei Bedarf kann Ihnen ein Rechner von Seiten der FernUniversität zur Verfügung gestellt werden.

Bei technischen Fragen zur Installation der Software auf diesen Rechnern wenden Sie sich bitte an:

Ute Becker,

Tel.: 02331/987-2887,

E-Mail: ute.becker@fernuni-hagen.de

7. Kurzbeschreibung der Beispielprogramme

7.1. Java 3D

In diesem Software-Praktikum soll ein Spiel mit Hilfe von *Java 3D* implementiert werden. Java 3D ist eine sogenannte Java-Software-Bibliothek, sie besteht aus einer Menge von Paketen (packages). Jedes Paket enthält Klassen mit ihren Methoden. Jedes dieser Pakete bzw. jede Public-Klasse dieser Pakete kann in Java-Programme importiert werden.

Der Einsatz einer 3D-Software-Bibliothek wie Java 3D hat einen entscheidenden Vorteil: Sie erzeugt aus der Definition/Beschreibung einer virtuellen Szene automatisch die graphische dreidimensionale Ausgabe auf dem Bildschirm. Der Entwickler kann mit Hilfe von Java 3D einfach und schnell dreidimensionale Szenen definieren und darstellen lassen. Der Entwickler braucht sich keine Gedanken darüber zu machen, wie aus der Definition der Szene die Ausgabe generiert (gerendert) wird. Beispiel: Um ein Dreieck graphisch auszugeben, müssen im einfachsten Fall nur die Positionen der drei Eckpunkte und ihre Farbe definiert werden. Anschließend sind nur wenige Zeilen Code zur Ausgabe des Dreiecks notwendig.

Um über die ersten Anfangsprobleme mit Java 3D hinwegzuhelfen, stellen wir neben einer Kurzanleitung zu Java 3D im Anhang B ein Beispielprogramm zur Verfügung, in dem intensiv von Java 3D Gebrauch gemacht wird. Insbesondere demonstriert es die Mechanismen der Interaktion und Animation in Java 3D.

Zur Dokumentation von Java 3D: Eine kurze Einführung in Java 3D finden Sie im Anhang A, auf der CD finden Sie auch ein weiteres ausführliches Java 3D Tutorium. Ebenso stellen wir Ihnen die Kurseinheit „Grafik-Bibliotheken“ aus dem Kurs Nr. 1692 „Graphische Datenverarbeitung I“ zur Verfügung. Diese Kurseinheit beschäftigt sich überwiegend mit Java 3D.

7.2. Ausführen des Beispielprogramms

Installieren Sie die notwendige Software (Java und Java 3D) inklusive Dokumentation. Kopieren Sie anschließend das Beispielprogramm von der Software-Praktikums-CD auf Ihre Festplatte. Kopieren Sie dazu das Verzeichnis:

```
beispielprogramm
```

Dort starten Sie das Beispielprogramm mit:

```
java Lab3D
```

Es sollte dann die in Abbildung 2 dargestellte Ausgabe zu sehen sein.

Wir empfehlen, noch genauer zu testen, ob Java und Java 3D ordnungsgemäß installiert worden sind. Dazu können Sie beispielsweise die Quellcode-Dateien von Lab3D erneut kompilieren. Löschen Sie dazu zuvor sicherheitshalber im Verzeichnis Lab3D und dem Unterverzeichnis `forLab3D` alle Dateien mit der Endung „.class“. Denn nur dann können Sie sicher sein, dass auch alle Java-Dateien des Beispielprogramms neu kompiliert werden. Kompilieren Sie anschließend das Programm neu:

```
javac Lab3D.java
```

Wenn das Beispielprogramm ohne Fehler kompiliert wird, können Sie von der ordnungsgemäßen Installation von Java und Java 3D ausgehen.

```
cmd.exe - java Lab3D
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-1999 Microsoft Corp.

D:\>cd javatest\Lab3D

D:\javatest\Lab3D>java Lab3D
Lab3D - Ein dreidimensionales Labyrinth
Lab3D.MyKeyBehavior: Tastenbelegung:
<Pfeil nach oben/unten> : Einen Schritt vor/zurueck
<Pfeil nach links/rechts>: Einen Schritt nach links/rechts
<s> / <d>: Um knapp 2 Grad nach links/rechts drehen
<a> / <f>: Um 30 Grad nach links/rechts drehen
<Enter> : Zurueck in die Startposition
<Leertaste>: In die Vogelperspektive / Zurueck in die Startposition
<h> : Help (diese Ausgabe)
<Esc> : Beenden
```

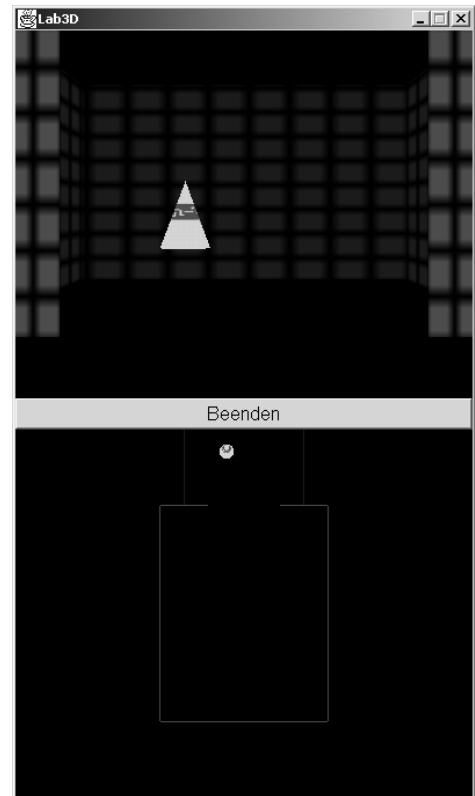


Abbildung 2: Links sehen Sie die Ausgabe auf der Kommandozeile, rechts das automatisch geöffnete grafische Ausgabefenster.

7.3. Bedienung des Beispielprogramms

Starten Sie abermals das Beispielprogramm (Lab3D). Sie erhalten das in Abbildung 3 dargestellte Fenster.

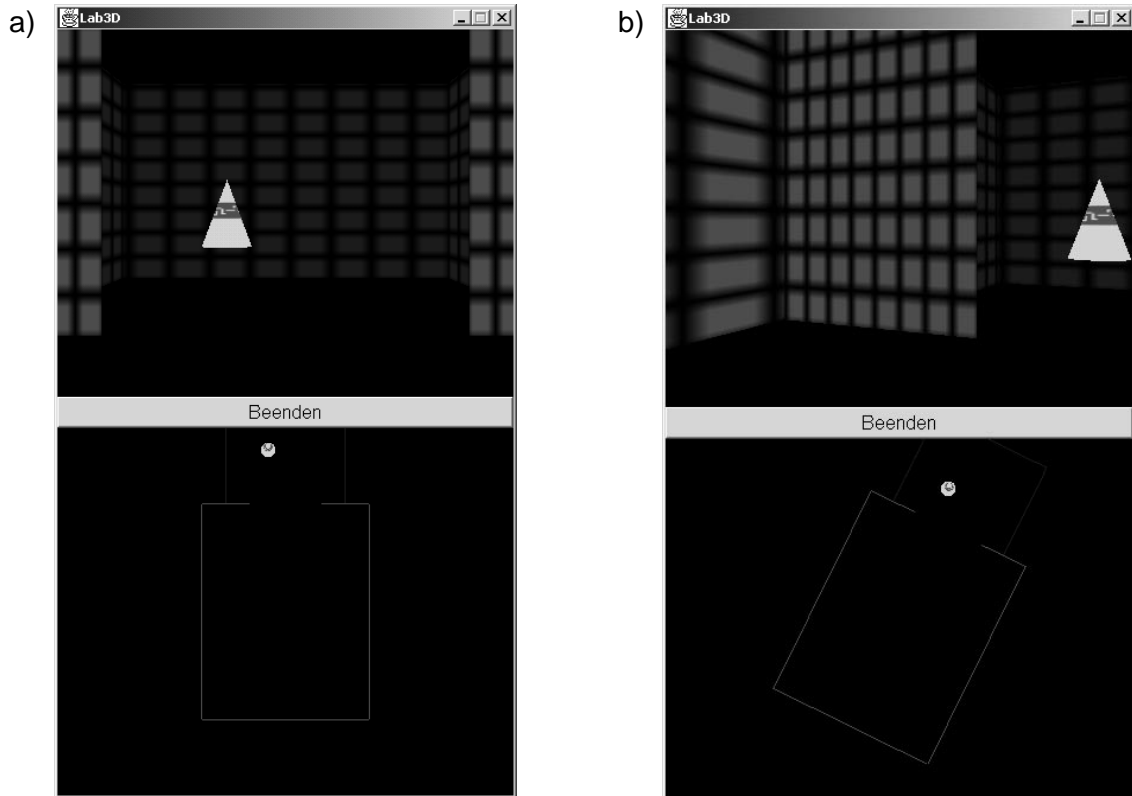


Abbildung 3: a) zeigt das Beispielprogramm Lab3D kurz nach dem Start. In b) ist die Ausgabe nach einer Drehung nach links zu sehen.

Die Ausgabe des Beispielprogramms ist ein Fenster mit einer Titelleiste, in der „Lab3D“ steht. In dem Fenster werden drei Elemente angeordnet:

- oben ein rechteckiger Bereich für eine graphische 3D-Ausgabe,
- darunter ein Button mit der Aufschrift „Beenden“ und
- darunter ein zweiter rechteckiger Bereich für eine graphische 3D-Ausgabe.

Im oberen Ausgabebereich wird das Labyrinth dreidimensional dargestellt. Die Szene ist so dargestellt, dass der Betrachter sich anfangs im Labyrinth befindet.

In dem unteren Ausgabebereich ist permanent ein Ausschnitt des Labyrinths in der Vogelperspektive zu sehen, das heißt man blickt von oben auf das Labyrinth herab. Die Wände des Labyrinths werden als Strecken dargestellt. Der Beobachter steht dabei in der Mitte des Ausgabebereichs, und sein Blick ist immer zum oberen Rand des Ausgabebereichs gerichtet.

In beiden Ausgabebereichen ist ein gelbes kegelförmiges Objekt zu sehen. Es bewegt sich in kleinen Schritten ziellos umher. Dieses Objekt soll als Beispiel dafür dienen, wie mit Java 3D die vom Computer gesteuerten Gegner implementiert werden können. Den gelbe Kegel nennen wir Bot.

In beiden Ausgabebereichen wird ein einfaches Labyrinth dargestellt. Das Labyrinth besteht aus zwei angrenzenden Räumen. Die Räume sind durch insgesamt acht Wände definiert. In der obigen Abbildung können Sie im unteren Ausgabebereich fast alle Wände sehen. Der untere größere Raum besteht aus fünf Wänden. Diese Wände sind rot, allerdings wurden sie mit einer gitterartigen Textur belegt. Das Ergebnis ist ein schwarzes Gitter auf rotem Hintergrund. Der obere Raum entsteht durch die Definition von drei Wänden mit blauer Farbe und der gleichen Textur. Die Textur ist nur im oberen Ausgabebereich sichtbar.

Die acht Wände sind in der folgenden Abbildung von 0 bis 7 durchnummeriert. Zudem sind die x- und z-Koordinatenachsen eingezeichnet. Die nicht dargestellte y-Achse zeigt aus der Abbildung auf den Betrachter der Abbildung. Die drei Wände des oberen (blauen) Raumes sind durch gepunktete Linien dargestellt.

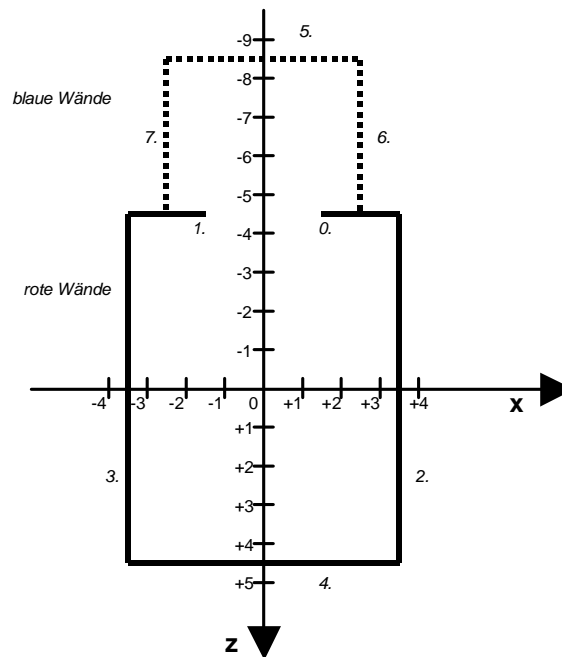


Abbildung 4: Auch in dieser Abbildung wird das Labyrinth von oben betrachtet. Die Zeichnung ähnelt dem Grundriss eines Hauses.

Direkt nach dem Start des Beispielprogramms befinden Sie sich im Ursprung des Koordinatensystems und sehen in entgegengesetzter Richtung der z-Achse.

Sie können sich mit Tastatureingaben durch das Labyrinth manövrieren. Eventuell müssen Sie zuvor mit der Maus in den oberen Ausgabebereich des Fensters klicken. Die Tastenbelegung:

- <Pfeil nach oben/unten>: Einen Schritt vor/zurück
- <Pfeil nach links/rechts>: Einen Schritt nach links/rechts
- <s> / <d>: Um knapp 2 Grad nach links/rechts drehen
- <a> / <f>: Um 30 Grad nach links/rechts drehen
- <Enter>: Zurück in die Startposition

Beispiel: Drückt man auf die Taste a, dreht man sich um 30 Grad nach links. Betrachten Sie bitte dazu die rechte Hälfte der Abbildung 3. Im oberen Ausgabebereich wird links die Wand 3 sichtbar.

Die Tastenbelegung wird direkt beim Start des Programms im „Eingabe-Text-Fenster“ ausgegeben. Sie können jedoch auch jederzeit die Tastenbelegung im „Eingabe-Text-Fenster“

anzeigen lassen, indem Sie in den oberen Ausgabebereich des Java-Fensters klicken und anschließend auf die Taste <h> wie „help“ drücken. Spielen Sie mit dem Programm, indem Sie sich durch die beiden Räume bewegen. Sie können mit den Pfeil-Tasten auch durch die Wände gehen. Betrachten Sie das Labyrinth auch von außen.

8. Spielbeschreibung

Das Ziel des Spiels wurde bereits in Abschnitt 1.2 dargestellt. Im Folgenden geht es um die Details des Spiels.

8.1. Bewegung auf einem horizontalen Raster

Die Spieler bewegen sich in einem dreidimensionalen Labyrinth. Bezüglich der Bewegung gibt es jedoch eine Einschränkung: Die Spieler bewegen sich ausschließlich in horizontaler Richtung, das Überspringen von Wänden oder ähnliches ist also nicht möglich.

8.2. Labyrinth und Wände

Abbildung 5 stellt einen möglichen Grundriss des Labyrinths dar. Ein Labyrinth besteht aus beliebig vielen **Wänden**. Die Wände können weder durchschritten noch durchschaut werden. Das Labyrinth (bzw. Spielfeld) ist von Wänden eingefasst, es gibt also keinen Weg aus dem Labyrinth heraus. Alle Wände sind gleich hoch. In dem Beispielprogramm sind die Wände mit einem Bild texturiert, das ein Gitter zeigt, siehe das Bild auf dem Deckblatt der vorliegenden Aufgabenstellung. Ob die Wände in Ihrem Programm texturiert sind und wenn wie, bleibt Ihnen überlassen.

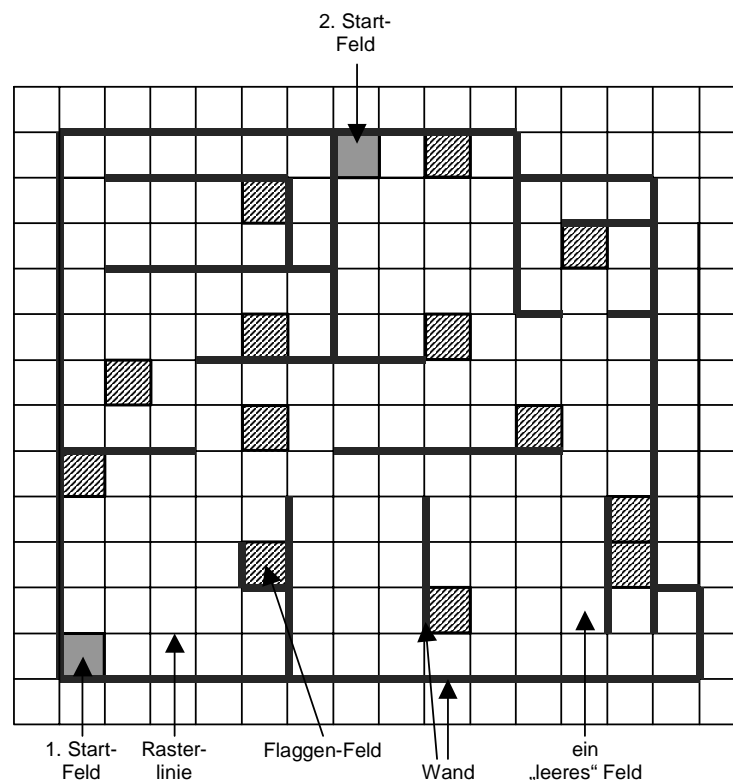


Abbildung 5: Ein beispielhaftes Labyrinth für zwei Spieler. Jeder Spieler hat sein eigenes Startfeld.

8.3. Spielfeld und Felder

Das Labyrinth steht auf einem gerasterten **Spielfeld**. Ob das Raster im Spiel dargestellt wird, bleibt Ihnen überlassen. Durch das Raster wird das Spielfeld in quadratische gleichgroße **Felder** aufgeteilt. Jedes Feld hat vier Ränder. Jede Wand des Labyrinths steht genau auf einem Rand von einem oder mehreren Feldern.

8.4. Arten von Feldern

Es werden drei Arten von Feldern unterschieden:

- Start-Felder
- Flaggen-Felder
- Alle übrigen Felder

Start-Felder: Ein Start-Feld ist der Ausgangspunkt genau eines Spielers. Jeder Spieler hat sein eigenes Start-Feld. Im Übrigen unterscheiden sich die Start-Felder nicht von den anderen Feldern.

Flaggen-Felder: Das Ziel des Spiels ist, schneller als die vom Computer gesteuerten Gegner die Flaggen zu erobern. Auf jedem sogenannten Flaggen-Feld steht genau eine Flagge. Betritt ein Spieler ein solches Feld oder berührt er die Flagge, so wird diese in einer dem Spieler zugeordneten Farbe eingefärbt. Ein Flaggen-Feld kann nicht zugleich ein Start-Feld sein.

Auf den Seiten eines jeden Feldes kann eine Mauer stehen.

8.5. Die Bewegung der Spieler durch das Labyrinth

Die Spieler bewegen sich, indem sie ihre Position verändern oder sich um die eigene Achse drehen.

Bezüglich der Positionsveränderung bewegen sich die Spieler nur in der Ebene. Spieler dürfen nicht durch andere Spieler oder Wände laufen können.

Die einzelnen Spieler haben eine feste Geschwindigkeit, in der sie sich durch das Labyrinth bewegen können. Auch die Geschwindigkeit, mit der ein Spieler sich drehen kann, ist festgelegt. Solange die entsprechende Taste gedrückt ist, bewegt oder dreht sich der Spieler mit der gegebenen Geschwindigkeit. Dabei bestimmt die Blickrichtung die Bewegungsrichtung, der Spieler kann sich immer in diese oder die entgegengesetzte Richtung bewegen.

8.6. Die Auswahl und Steuerung der Computergegner

Der Schwierigkeitsgrad des Spiels soll in einem Startmenü einstellbar sein. Er setzt sich dabei aus drei Faktoren zusammen:

1. In welchem Labyrinth soll gespielt werden ?
2. Wie viele Computergegner soll es geben ? Die maximale Anzahl von Gegnern ist durch die Anzahl der Startfelder des ausgewählten Labyrinths beschränkt.
3. Welche Spielstärke haben die einzelnen Computergegner ? Dies ist ein Parameter, der die Strategie und die Geschwindigkeit des jeweiligen Computergegners beeinflusst.

Ein Computergegner bewegt sich genau wie der Spieler. Ein wichtiger Faktor bei der Steuerung der Computergegner ist die Bedingung, dass dem Computergegner die gleichen Informationen zur Verfügung stehen wie dem Spieler. Der Computergegner beginnt also mit einer

leeren Karte und muss erst das Labyrinth erkunden, um Kenntnisse über dessen Topologie zu erlangen. Verfahren Sie dazu ähnlich wie bei der Kartenfunktion, die im nächsten Abschnitt beschrieben wird. Überlegen Sie, ob für die Planung der Computerstrategie nicht eine weitere Repräsentation des Labyrinths notwendig wird (Stichwort Repräsentation in Graphen und Berechnung kürzester Wege). Vielleicht können Sie sogar anbieten, Computergegner mit verschiedenen Strategien auszuwählen. Es ist sicherlich interessant zu beobachten, wie sich die Strategien von unterschiedlichen Computergegnern gegenseitig beeinflussen. Auch ein Computergegner soll sich nur durch Drehung und Bewegung in oder entgegengesetzt seiner Blickrichtung durch das Labyrinth bewegen, ebenso darf er nicht durch Wände gehen. Ebenso wie der Spieler (siehe Abschnitt 8.7) hat auch der Computergegner Zugriff auf den aktuellen Spielstand.

8.7. Drei Ansichten auf das Labyrinth

Der menschliche Spieler hat zwei mögliche Ansichten auf das Labyrinth zur Verfügung. Des weiteren gibt es eine zusätzliche Ansicht, die das Austesten des Spiels erleichtert:

1. Die **Ego-Perspektive**: Der Spieler steht im Labyrinth. Sein Sichtbereich ähnelt der Ausgabe im oberen Ausgabebereich des Beispielprogramms.
2. Die **Test-Vogelperspektive**: Der Spieler befindet sich zwar im Labyrinth, jedoch kann er wie mit einer Video-Kamera das Labyrinth von oben aus der Vogelperspektive betrachten. Die so erzeugte Karte zeigt alle Elemente des Labyrinths : Wände, Flaggen in ihrer aktuellen Farbe, die Startfelder in der Farbe des jeweiligen Spielers und die einzelnen Spieler mit ihrer aktuellen Blickrichtung. Über Tasteneingaben kann die Entfernung des Betrachteten zum Labyrinth variiert werden. Dies bezeichnen wir als Zoomfunktion. Diese Ansicht dient nur zu Testzwecken und soll im Spiel später nicht zum Einsatz kommen (oder nur durch eine „geheime“ Tastatureingabe aktiviert werden können). Die Spielfiguren müssen optisch deutlich voneinander zu unterscheiden sein.
3. Die **Automatische Karte**: Diese Ansicht ist zu Beginn des Spiels leer, und nur der Spieler und seine Blickrichtung sind zu erkennen. Im Verlauf des Spiels zeigt die Karte alle Wände und Flaggen an, die der Spieler bisher innerhalb seiner Ego-Perspektive (auch nur teilweise) gesehen hat. Diese Elemente sollen wie in der Test-Vogelperspektive dargestellt werden, mit der Ausnahme, dass die aktuelle Farbe der Flaggen nicht zu erkennen ist. Diese Ansicht besitzt auch eine Zoomfunktion. Überlegen Sie, ob Sie dieses Problem schon in einer zweidimensionalen Repräsentation des Labyrinths lösen können.

Zusätzlich soll der Spieler immer den aktuellen Spielstand mitgeteilt bekommen. Der Spielstand besteht aus der Information, welcher Spieler/Computergegner momentan wie viele Flaggen in Besitz hat.

8.8. Der Labyrinth-Editor

Das Spiel soll einen Labyrinth-Editor enthalten. Dieser Editor soll dem Benutzer die Möglichkeit geben, seine eigenen Labyrinthe aus Wänden, Flaggen- und Startfeldern zu erstellen. Dafür soll dem Benutzer eine graphische Oberfläche angeboten werden, die das Erstellen, Laden, Speichern und Editieren von Labyrinthen unterstützt.

8.9. Das Dateiformat

Ein in dem Editor erstelltes Labyrinth soll in einer Textdatei gespeichert werden. Dabei soll das hier beschriebene Dateiformat verwendet werden.

Aus folgender Textdatei soll das in der darauffolgenden Abbildung 7 dargestellte Labyrinth generiert werden.

	a	b	c	d	e	f	g	h	i	j	k
1	-	-	-	-	-	-	-	-	-	-	-
2		1								F	
3											
4					F						
5		-	-	-							
6		F									
7		-	-	-							
8					F						
9											
10		0								F	
11	-	-	-	-	-	-	-	-	-	-	-

Abbildung 6:
Die Textdatei

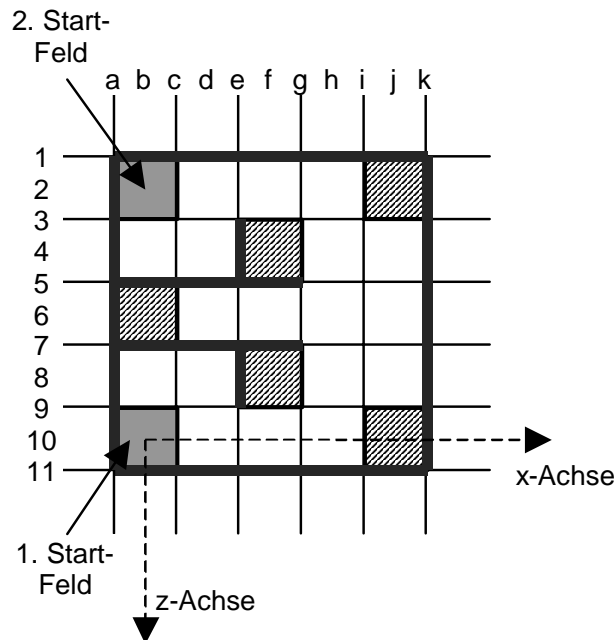


Abbildung 7: Eine einfaches Labyrinth für zwei Spieler

Der Inhalt der Textdatei ist umrahmt. Über der Textdatei sind die einzelnen Spalten der Textdatei mit Buchstaben beschriftet, mit den Zahlen links neben der Textdatei werden die Zeilen der Datei nummeriert. Jede Zeile wird mit einem Zeilenwechselzeichen abgeschlossen. Vor einem Zeilenwechselzeichen darf kein Leerzeichen stehen. Also sind die Zeilen 3 und 9 bis auf das Zeilenwechselzeichen leer.

Ein Startfeld wird durch eine Ziffer gekennzeichnet, die Ziffern beginnen bei 0. Ein f oder F steht für ein Flaggen-Feld. Wände werden durch Gedankenstriche („-“) oder vertikale Striche („|“) gekennzeichnet.

Die Zeilen mit einer ungeraden Zahl links neben dem Rahmen enthalten ausschließlich Informationen zu Wänden, die parallel zur x-Achse verlaufen. Die Spalten mit den Buchstaben a, c, e, ... enthalten ausschließlich Informationen zu Wänden, die parallel zur z-Achse liegen.

Folgen mehrere Gedankenstriche (durch je ein Leerzeichen getrennt) hintereinander, ist nicht für jeden Gedankenstrich eine einzelne Wand zu generieren. Statt dessen sind die Gedankenstriche als eine einzige Wand zu interpretieren.

8.10. Füllen der MyLabData-Klasse

Nach oder beim Einlesen der Textdatei werden die Daten in einer passenden Datenstruktur abgelegt. Die zugehörige Klasse wird im Beispielprogramm `MyLabData`-Klasse genannt.

Das Beispielprogramm liest die Daten nicht aus einer Textdatei, sondern die Daten sind im Konstruktor der Klasse `forLab3D.MyLabData` „hart“ codiert. Im Beispielprogramm werden nur die Wände eines Labyrinths dargestellt, es gibt keine Einteilung des Raums in Felder und auch keine Gegner. Demzufolge werden in der Klasse `MyLabData` des Beispielprogramms nur Daten über die Wände abgelegt.

9. Teilnahmevoraussetzung: Erstellung eines Prototypen

Im Software-Praktikum sollen die einzelnen Teilaufgaben möglichst gleichmäßig auf die einzelnen Teilnehmer verteilt werden. Deshalb müssen alle Teilnehmer von Anfang an in der Lage sein produktiv mitzuarbeiten. Eine Voraussetzung dafür ist eine Vertrautheit mit Java. Um diese Voraussetzung bei allen Teilnehmern zu überprüfen, bitten wir Sie, in Einzelarbeit ein lauffähiges Java-Programm zu erstellen. Dieses Programm ist uns vor der ersten Präsenzphase zur Verfügung zu stellen, den genauen Termin entnehmen Sie bitte Kapitel 3.

Das zu erstellende Programm ist eine erste rudimentäre Version des Spiels. Es erweitert das erste Beispielprogramm (Lab3D) um die in diesem Kapitel beschriebenen Funktionen: Es

- liest eine Textdatei (in dem Format aus Kapitel 8) aus,
- legt die Daten in einer passenden Datenstruktur ab und
- erstellt mit Hilfe der Datenstruktur ein Labyrinth.

Syntaxfehler wie nicht definierte Zeichen oder Zeichen in jeder zweiten Zeile für Wände, die parallel zur x-Achse sind, müssen erkannt werden und zu einer Fehlermeldung führen. In dem aufgebauten Labyrinth soll man sich so frei bewegen können wie in dem Beispielprogramm. Elemente wie Flaggen sollen dabei eindeutig als solche zu erkennen sein. Der Name der einzulesenden Labyrinthdatei soll dem Programm als Kommandozeilenparameter übergeben werden.

Vergessen Sie nicht, dass auch ein Entwurf des endgültigen Spiels zusammen mit diesem Prototypen abgegeben werden muss.

10. Dokumentation

Neben der Fertigstellung des eigentlichen Spiels muss noch eine Dokumentation des Programms und eine Bedienungsanleitung erstellt werden.

10.1. Drei Arten der Dokumentation

Zur Dokumentation gehört:

- Die Dokumentierung des Sourcecodes im Sourcecode mit javadoc und weiteren Kommentaren.
- Die Beschreibung der Implementierung inklusive der Klassendiagramme und der Darstellung der Strategie der vom Computer simulierten Spieler.
- Eine kurze Spielanleitung inklusive der Beschreibung der Benutzung der Anwendung.

10.2. Format der Dokumentation

Die Korrektur wird beschleunigt, wenn Sie Ihren Entwurf per E-Mail einsenden. Erlaubte Formate sind PostScript und PDF. Die Dokumente müssen mit fortlaufenden Seitennummern und einem Inhaltsverzeichnis versehen sein.

10.3. Zur Dokumentierung des Sourcecodes

Da Sie in einem Team arbeiten, sollte Ihr Sourcecode auch für andere gut lesbar sein. Geben Sie sich deshalb insbesondere bei der Dokumentation jeder Klasse und jeder Methode mit den Attributen `public` oder `protected` Mühe. Als Beispiel für die Dokumentierung von Sourcecode kann das Beispielprogramm dienen. Im Beispielprogramm wurden alle Methoden dokumentiert.

Es folgen einige Hinweise zur Dokumentierung des Sourcecodes:

- Stellen Sie jeder Klasse (das heißt in der Regel in jeder Datei) eine **kurze Einführung** (ca. 10 bis 20 Zeilen) voran. Stellen Sie in dieser kurzen Einführung dar, was die Klasse macht. Skizzieren Sie dabei die Grundidee, ohne auf Implementierungsdetails einzugehen.
- **Kommentieren Sie Ihren Sourcecode.** Das soll *nicht* heißen, dass Sie zu jeder Anweisung einen Kommentar schreiben müssen. Jedoch muss Ihr Programm mit Hilfe der Kommentare soweit verständlich sein, dass der Leser Ihre Lösung schnell nachvollziehen kann.
- Verwenden Sie **aussagekräftige Bezeichner** für Ihre Klassen, Methoden, Variablen, und Konstanten. Ein gut gewählter Bezeichner ist so kurz wie möglich, aber lang genug, um seine Funktion verständlich zu beschreiben.

Um eine einheitliche Dokumentation des Sourcecodes sicherzustellen, legen wir fest, dass die Dokumentation mit Hilfe von javadoc (oder „JavaDoc“) erfolgen muss. javadoc ist Thema von Anhang C.

11. Anhang A: Java 3D

Dieser Anhang ist eine kurze Einführung in Java 3D.

11.1. Was ist Java 3D?

Java 3D ist eine Bibliothek aus Java-Paketen (packages) und der Dokumentation dieser Pakete. Mit Hilfe von Java 3D können anspruchsvolle dreidimensionale Szenen modelliert werden. Eine derart programmierte Anwendung stellt nach ihrem Start die visuelle virtuelle Szene in einem Fenster dar.

Werden visuelle Objekte in einem Raum positioniert, sprechen wir von einer Szene. Im Software-Praktikum enthält die Szene das Labyrinth mit seinen Wänden und speziellen Feldern sowie eine Spielfigur für jeden Spieler.

11.2. Literatur

Außer der vorliegenden Aufgabenbeschreibung haben Sie die Kurseinheit 7 „Grafik-Bibliotheken“ des Kurses „Graphische Datenverarbeitung I“ (1692) auf CD erhalten. Diese Kurseinheit enthält das Kapitel 6 des Kurses „Graphische Datenverarbeitung I“. Das Unterkapitel 6.2 trägt die Überschrift „Java 3D“. Auch weitere Abschnitte des sechsten Kapitels sind hilfreich, um sich in Java 3D einzuarbeiten. Wir empfehlen folgende Abschnitte:

- Erster Absatz von Kapitel 6 „Grafik-Bibliotheken“
- Unterkapitel 6.1 „Zweck und Anwendung von Grafik-Bibliotheken“
- Unterkapitel 6.2 „Java 3D“
- Abschnitt 6.8.1 „Literatur zu Java 3D“
- Den Index
- Das Glossar (wird im Kurstext auch als Zusammenfassung bezeichnet)

11.3. Installation

Wie auch bei der Programmiersprache Java war die Firma Sun federführend bei der Entwicklung von Java 3D.

Java 3D ist nicht im Java 2 SDK (Standard Development Kit, früher mit JDK für Java Development Kit bezeichnet) enthalten, sondern eine sogenannte Standard-Erweiterung vom Java 2 SDK: Die Klassen von Java 3D erweitern die Klassenhierarchie von Java 2. Vor der Installation von Java 3D ist zunächst das Java SDK 2 ab Version 1.2 zu installieren.

11.4. Körper, Oberflächen, Dreiecke und Punkte

Bevor der Software-Entwickler mit Hilfe von Java 3D virtuelle Objekte (Körper) darstellen kann, muss er die virtuellen Objekte erst einmal definieren („modellieren“). Im einfachsten Fall wird für jedes virtuelle Objekte eine Instanz der passenden Java 3D-Klasse definiert und dabei initialisiert. Beispielsweise definiert man eine Kugel, indem man eine Instanz der Klasse `Sphere` (Kugel) erzeugt. Dazu ruft man den Konstruktor der Klasse `Sphere` auf. Als Parameter übergibt man dem Konstruktor den Mittelpunkt der Kugel (x-, y- und z-Koordinate) und den Radius.

Java 3D interessiert sich nur für die Oberflächen visueller Objekte, es werden keine Angaben zum Volumen der Objekte gespeichert oder verarbeitet. Nach dem Definieren der Kugel zerlegt Java 3D die Kugeloberfläche automatisch in Dreiecke. Sind die Dreiecke klein genug, erscheint die Oberfläche rund. Dieser Eindruck entsteht selbst dann, wenn die „Kugel“ von einer Seite beleuchtet wird und dadurch die einzelnen Dreiecke unterschiedlich hell sind. In Abbildung 8 werden vier Grundkörper durch ein einfaches Java 3D-Programm dargestellt.

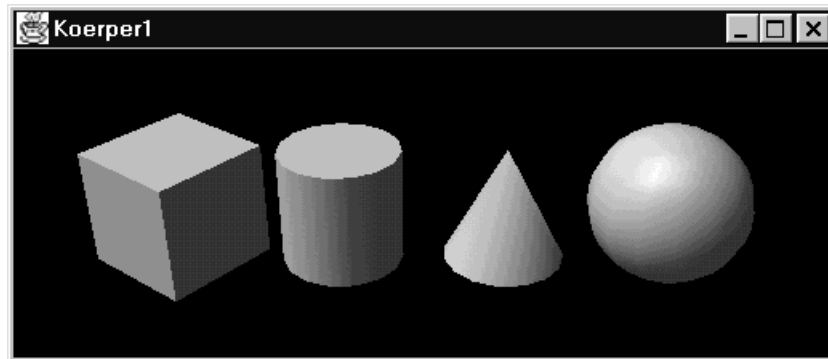


Abbildung 8: Beispiel *Koerper1.java*: Darstellung der vier Grundkörper von Java3D.

Nicht nur Kugeloberflächen werden von Java 3D mit Hilfe eines Netzes von Dreiecken dargestellt, sondern auch sämtliche andere Oberflächenformen. So besteht z.B. jedes Viereck aus zwei Dreiecken.

Die Position eines Dreiecks ist durch die Position seiner drei Eckpunkte definiert. Da jedes visuelle Objekt von Java 3D automatisch durch Dreiecke ersetzt wird, ist die Form jedes visuellen Objekts durch die Position der Eckpunkte seiner Dreiecke bestimmt. In der Java 3D-Implementierung ist somit der Eckpunkt oder allgemein der *Punkt* das zentrale geometrische Element.

Ein Punkt wird im dreidimensionalen Raum durch seine drei Koordinaten definiert.

11.5. Transformationen

Jedes visuelle Objekt kann im Dreidimensionalen

- verschoben (*translatiert*),
- vergrößert/verkleinert (*skaliert*) und
- gedreht (*rotiert*)

werden. Translation, Skalierung und Rotation werden als Transformationen bezeichnet. Auf weitere Arten von Transformationen gehen wir hier nicht ein. Beispiele für die drei Transformationsarten zeigt Abbildung 9.

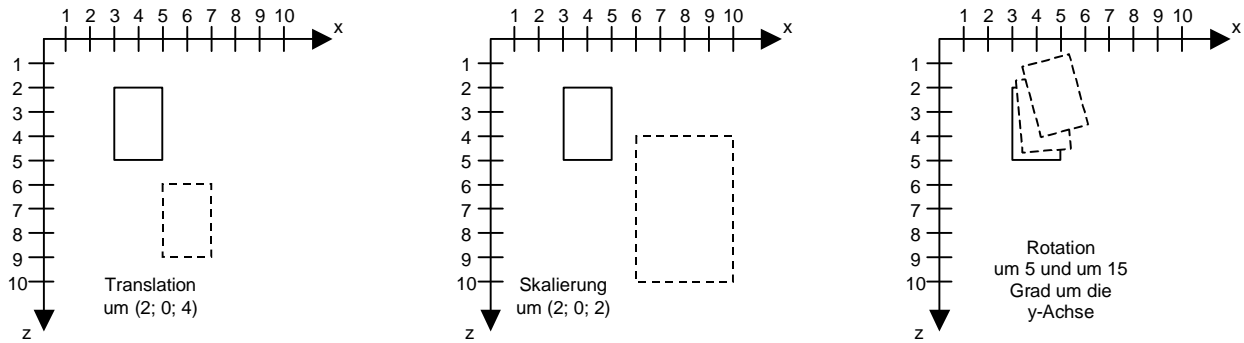


Abbildung 9: Drei Arten von Transformationen im Dreidimensionalen. Die y-Achse zeigt in Richtung des Betrachters. Ein Rechteck mit Kanten, die parallel zur x- und z-Achse liegen, wird links translatiert, in der Mitte skaliert und rechts rotiert. Das Ergebnis ist das Rechteck, dessen Kanten gestrichelt dargestellt werden.

11.6. Transformationen als Matrizen

Translation, Skalierung und Drehung werden durch 4x4-Matrizen beschrieben. Eine zusammengesetzte Transformation kann dann durch das Produkt der Einzelmatrizen beschrieben werden. Die 16 Werte der Matrix haben unterschiedliche Aufgaben, wie die folgende Matrix zeigt. Die neun Werte oben links (die Werte der drei ersten Spalten der drei ersten Zeilen) definieren, in wie fern Skalierungen und Rotationen Teil der Transformationen sind. Die drei Werte in den drei Zeilen der letzten Spalte sind für die Translation zuständig. Die vier Werte der vierten Zeile sind immer 0, 0, 0, 1.

	Skalierung, Rotation	Trans- lation
$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix}$		
	Quasi konstante Werte	

1) *Translation:* Soll ein Objekt um L_x Längeneinheiten in Richtung der x-Achse, L_y Längeneinheiten in Richtung der y-Achse und L_z Längeneinheiten in Richtung der z-Achse verschoben werden, hat die Matrix folgende Gestalt:

$$\begin{pmatrix} 1 & 0 & 0 & L_x \\ 0 & 1 & 0 & L_y \\ 0 & 0 & 1 & L_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

2) *Skalierung:* Soll ein Objekt um den Faktor S_x in Richtung der x-Achse, um den Faktor S_y in Richtung der y-Achse und um den Faktor S_z in Richtung der z-Achse vergrößert/verkleinert werden, ergibt sich folgende Matrix:

$$\begin{pmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

3) *Rotation:* Bei der Rotation wird zwischen drei Arten der Rotation unterschieden: Die Rotation um die x-, y- oder z-Achse.

a) Rotation um die x-Achse: Soll ein Objekt um den Winkel α um die x-Achse gedreht werden, lautet die zugehörige Matrix:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \alpha & -\sin \alpha & 0 \\ 0 & \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

b) Rotation um die y-Achse: Soll ein Objekt um den Winkel α um die y-Achse gedreht werden, lautet die zugehörige Matrix:

$$\begin{pmatrix} \cos \alpha & 0 & \sin \alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \alpha & 0 & \cos \alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

c) Rotation um die z-Achse: Soll ein Objekt um den Winkel α um die z-Achse gedreht werden, lautet die zugehörige Matrix:

$$\begin{pmatrix} \cos \alpha & -\sin \alpha & 0 & 0 \\ \sin \alpha & \cos \alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

11.7. Anwendung einer Transformation auf einen Punkt

Nachdem die 4×4-Matrix erzeugt wurde, wird diese Matrix jeweils mit allen Punkten (der Dreiecke) des Objekts multipliziert. Bei n Punkten entstehen n transformierte Punkte.

Ein Punkt im Dreidimensionalen wird durch seine *drei* Koordinaten definiert. Um ihn mit einer 4×4-Matrix zu multiplizieren, wird der Punkt (eines Dreieckes) des Objekts um eine vierte Komponente mit dem Wert 1 erweitert. Ist beispielsweise der Punkt (3; 0; 2), wird daraus (3; 0; 2; 1). Dieser Punkt ist ein Eckpunkt des nicht gestrichelten Rechtecks in der obigen Abbildung. Wird dieser Punkt um den Vektor (2; 0; 4) translatiert, ergibt sich:

$$\begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 4 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 3 \\ 0 \\ 2 \\ 1 \end{pmatrix} = \begin{pmatrix} 5 \\ 0 \\ 6 \\ 1 \end{pmatrix}$$

11.8. Verknüpfung von Transformationen

Häufig werden mehrere Transformationen hintereinander in einer bestimmten Reihenfolge ausgeführt bzw. verknüpft.

Beispiel: Ein Quader soll 30 Grad *um seinen Mittelpunkt* (Schwerpunkt) gedreht werden, die Drehachse soll parallel zur y-Achse liegen, der Mittelpunkt liegt bei (8; 1; 7). Allerdings kennen wir keine Transformations-Matrix für das Rotieren um einen beliebigen Punkt, sondern nur für die Rotation um eine Koordinatenachse. Wie in Abbildung 10 dargestellt, wird die Rotation um einen beliebigen Punkt mit drei aufeinanderfolgenden Transformationen realisiert. Der Quader in der Position vor den Transformationen ist grau.

1. Transformation: Translation, so dass der Mittelpunkt des Quaders im Ursprung liegt.
2. Transformation: Rotation um die y-Achse um 30 Grad.
3. Transformation: Translation um den gleichen Vektor wie bei der 1. Transformation, jedoch mit umgekehrtem Vorzeichen.

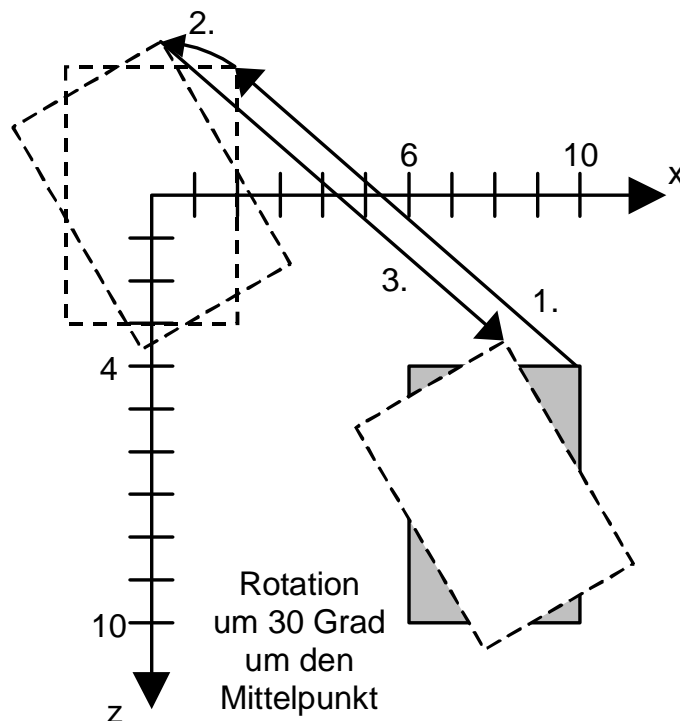


Abbildung 10: Um den grau schattierten Quader um seinen Mittelpunkt zu drehen, werden nacheinander drei Transformationen ausgeführt. Die drei Pfeile stehen für die drei Transformationen, angewendet auf einen Eckpunkt des Quaders.

Widmen wir uns nun der Frage, wie bei dem in Abbildung 10 gezeigten Beispiel die Matrizen miteinander verknüpft werden. Nehmen wir an, ein Eckpunkt des Quaders hätte die Position (10; 0; 4). Wenn mehrere Transformationen hintereinander auf einen Punkt angewendet werden, werden zunächst die 4×4 -Matrizen der einzelnen Transformationen miteinander multipliziert. Das Ergebnis ist wiederum eine 4×4 -Matrix. Anschließend wird diese Matrix mit dem Punkt multipliziert.

$$\begin{aligned}
& \left(\left(\begin{pmatrix} 1 & 0 & 0 & L_x \\ 0 & 1 & 0 & L_y \\ 0 & 0 & 1 & L_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \cos \alpha & 0 & \sin \alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \alpha & 0 & \cos \alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & -L_x \\ 0 & 1 & 0 & -L_y \\ 0 & 0 & 1 & -L_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 10 \\ 0 \\ 4 \\ 1 \end{pmatrix} \right) \\
& = \left(\left(\begin{pmatrix} 1 & 0 & 0 & 8 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 7 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \cos 30 & 0 & \sin 30 & 0 \\ 0 & 1 & 0 & 0 \\ -\sin 30 & 0 & \cos 30 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & -8 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & -7 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 10 \\ 0 \\ 4 \\ 1 \end{pmatrix} \right) \\
& = \left(\left(\begin{pmatrix} 1 & 0 & 0 & 8 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 7 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \frac{\sqrt{3}}{2} & 0 & \frac{1}{2} & 0 \\ 0 & 1 & 0 & 0 \\ -\frac{1}{2} & 0 & \frac{\sqrt{3}}{2} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & -8 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & -7 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 10 \\ 0 \\ 4 \\ 1 \end{pmatrix} \right) \\
& = \left(\begin{pmatrix} \frac{\sqrt{3}}{2} & 0 & \frac{1}{2} & 8 \\ 0 & 1 & 0 & 1 \\ -\frac{1}{2} & 0 & \frac{\sqrt{3}}{2} & 7 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & -8 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & -7 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 10 \\ 0 \\ 4 \\ 1 \end{pmatrix} \right) \\
& = \begin{pmatrix} \frac{\sqrt{3}}{2} & 0 & \frac{1}{2} & -4 \cdot \sqrt{3} + \frac{9}{2} \\ 0 & 1 & 0 & 0 \\ -\frac{1}{2} & 0 & \frac{\sqrt{3}}{2} & -7 \cdot \frac{\sqrt{3}}{2} + 11 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 10 \\ 0 \\ 4 \\ 1 \end{pmatrix} \\
& = \begin{pmatrix} 5 \cdot \sqrt{3} + 2 - 4 \cdot \sqrt{3} + \frac{9}{2} \\ 0 \\ -5 + 2 \cdot \sqrt{3} - 7 \cdot \frac{\sqrt{3}}{2} + 11 \\ 1 \end{pmatrix} = \begin{pmatrix} \sqrt{3} + \frac{13}{2} \\ 0 \\ -3 \cdot \frac{\sqrt{3}}{2} + 6 \\ 1 \end{pmatrix} = \begin{pmatrix} 8,23\dots \\ 0 \\ 3,40\dots \\ 1 \end{pmatrix}
\end{aligned}$$

Das Ergebnis ist der Punkt (8,23...; 0; 3,40...).

Kommen wir nun wieder zu unserem Beispielprogramm Lab3D. In diesem können Wände nicht verändert werden, nach der Festlegung ihrer Position am Anfang der Beispielprogramme werden sie nicht mehr verändert, also auch nicht rotiert. In dem Beispielprogramm Lab3D bewegt sich jedoch der Betrachter durch das Labyrinth. Dabei können sich seine Position und Blickrichtung ändern.

Um die Blickrichtung zu verändern, wird der Betrachter um seine momentane Position rotiert. Die Drehachse liegt parallel zur y-Achse. Die Blickrichtung ist stets horizontal, die y-Komponente des Blickrichtungsvektors hat stets den Wert 0.

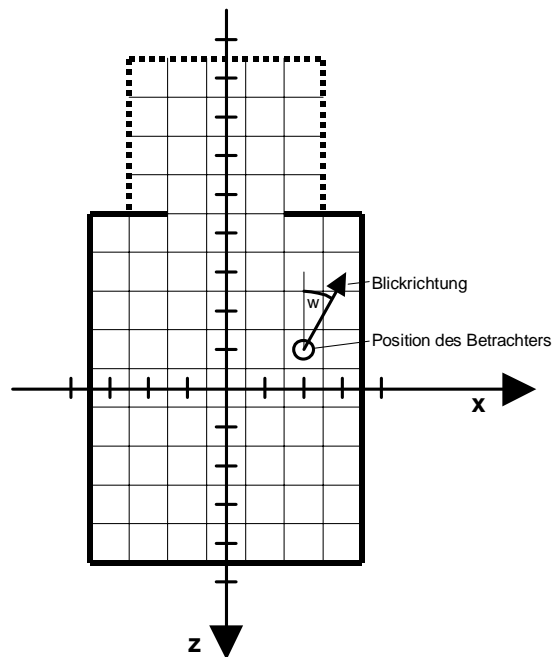


Abbildung 11: Eine Position und Blickrichtung des Betrachters in den beiden Beispielprogrammen.

Wie oben beschrieben, könnte der Betrachter in drei Schritten rotiert werden: Verschieben in den Ursprung, rotieren um den Ursprung und zurückverschieben. In den beiden Beispielprogrammen wird jedoch anders vorgegangen: Das Programm merkt sich die Koordinaten der Position und den Winkel zwischen der negativen z-Achse und dem Vektor der Blickrichtung. Der Winkel wird in der obigen Abbildung mit w benannt. Bei jeder Änderung der Position oder der Blickrichtung wird die Lage des Benutzers neu berechnet: Zunächst wird eine 4×4 -Matrix für die Translation, dann eine 4×4 -Matrix für die Rotation definiert. Anschließend werden beide Matrizen in der richtigen Reihenfolge miteinander multipliziert.

11.9. Transformationen in Java 3D

Eine 4×4 -Matrix wird in Java 3D durch eine Instanz der Klasse `javax.media.j3d.Transform3D` repräsentiert. Eine solche Instanz enthält alle 16 Werte einer solchen Matrix und bietet einige Methoden, um auf die Daten zuzugreifen. Falls man eine Instanz von `Transform3D` mit dem parameterlosen Konstruktor erzeugt, sind die Werte auf den Diagonalen 1 und die übrigen 12 Werte 0.

Im vorigen Kapitel haben wir gesehen, wie Matrizen miteinander multipliziert werden, um Transformationen hintereinander auszuführen. Die Klasse `Transform3D` bietet die Methode `mul`, um dies zu realisieren. Es folgt der Methodenkopf von `mul`:

```
public final void mul(Transform3D t1)
```

Ein Beispiel: Wird eine Instanz von `Transform3D` mit dem Namen `t3d` erzeugt und später ihre Methode `mul` mit dem Parameter `t1` aufgerufen, enthält die Instanz anschließend das Ergebnis `t3d.t1`.

11.10. Szenegraph

Die zentrale Datenstruktur bei Java 3D ist ein Graph mit einer baumartigen Struktur. Der Graph enthält alle Informationen, die eine Szene sowie die Position und die Blickrichtung des Betrachters beschreiben. Die Knoten des Graphen sind Instanzen von Java 3D-Klassen, in Kurseinheit 7 „Grafik-Bibliotheken“ des Kurses „Graphische Datenverarbeitung I“ werden diese Instanzen „Datenelemente“ genannt. Diese Instanzen werden durch den Aufruf bestimmter Methoden miteinander verknüpft. Durch den Aufruf der betreffenden Methode wird eine Verknüpfung zwischen der Instanz der Methode und einer anderen Instanz erzeugt. Die Verknüpfungen sind die Kanten des Graphen. Der gesamte Graph wird als Szenegraph bezeichnet.

Das Beispielprogramm könnte so implementiert sein, dass für jede Wand des Labyrinths eine Instanz einer bestimmten „Wand-Klasse“ erzeugt würde. Dann wäre jede dieser Instanzen ein Knoten des Szenegraphen. Auch die Lage des Betrachters ist in einem bestimmten Knoten festgelegt.

Die Wurzel des Szenegraphen ist eine Instanz der Klasse `VirtualUniverse`. Unter dieser Instanz folgt mindestens eine `Locale`-Instanz. Anschließend teilt sich der Szenegraph in zwei Teilgraphen. Es werden zwei Arten von Verzweigungsgraphen unterschieden:

- Ein Inhaltsgraph (content branch graph) beschreibt ausschließlich den Inhalt der Szene. Der Inhaltsgraph ist auf der linken Seite von Abbildung 12 dargestellt.
- Der Sichtgraph (view branch graph) beschreibt ausschließlich die Sichtweise, das heißt vor allem die Lage (Position und Blickrichtung) des Beobachters. Der Sichtgraph ist auf der rechten Seite von Abbildung 12 dargestellt.

Eine wichtige Rolle haben Instanzen der Klassen `BranchGroup` (BG) und `TransformGroup` (TG):

- `BranchGroup`-Instanzen dienen ausschließlich der Gruppierung von weiteren Knoten des Szenegraphen. Jede `BranchGroup`-Instanz ist die Wurzel eines Teilgraphen des Szenegraphen. Alle Knoten bzw. Instanzen in einem solchen Teilgraphen werden logisch gruppiert.
- `TransformGroup`-Instanzen dienen wie `BranchGroup`-Instanzen der Gruppierung weiterer Knoten des Szenegraphen. Zusätzlich werden die geometrischen Daten der gruppierten Knoten geometrisch transformiert.

Eine weitere Klasse, deren Instanzen in einen Szenegraphen gehängt werden können, ist `Shape3D`. Mit ihr wird die Form eines geometrischen Objekts festgelegt, z.B. eines Würfels, Zylinders oder einer Kugel. Diese Funktion kann übrigens auch von Instanzen der Klasse `Node` übernommen werden. Die ganz unten im Szenegraphen von Abbildung 12 aufgeführten Instanzen der Klassen `Appearance` und `Geometry` legen weitere Eigenschaften des `Shape3D`-Objekts wie z.B. die Textur fest.

Die Abbildung 12 zeigt einen einfachen Szenegraphen. Der Szenegraph ist Thema des Unterkapitels 6.2.7 der beiliegenden Kurseinheit 7 „Grafik-Bibliotheken“ des Kurses „Graphische Datenverarbeitung I“ (Nummer 1692), deshalb gehen wir hier nicht näher auf den Szenegraphen ein.

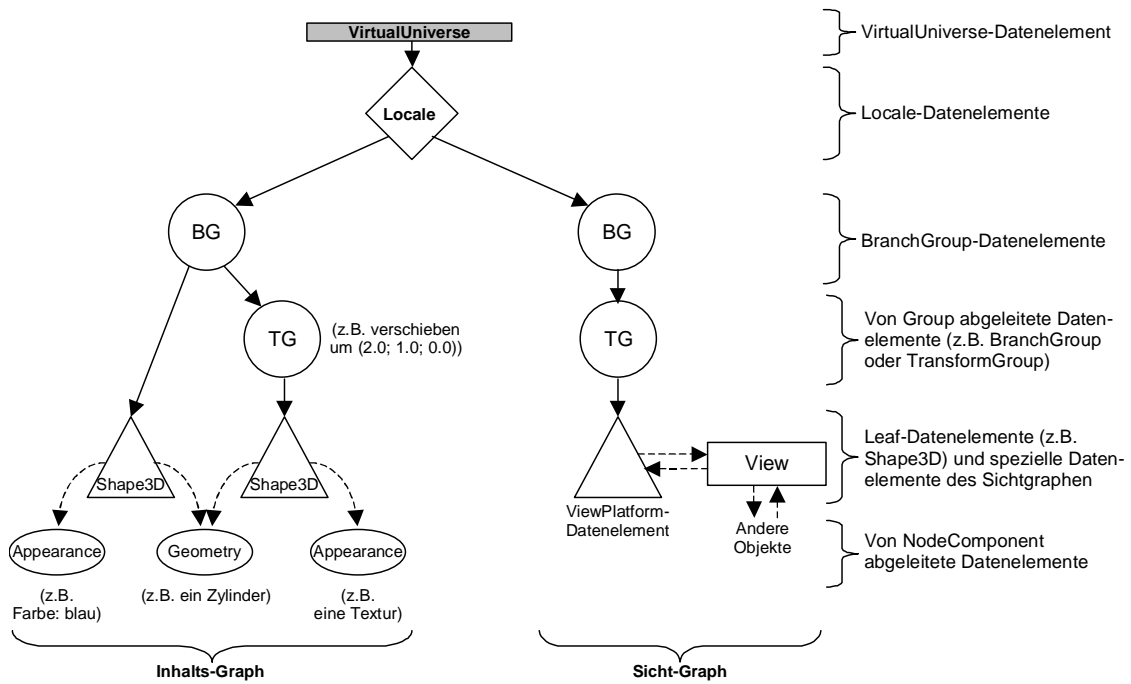


Abbildung 12: Ein Szenegraph, mit dem eine Szene definiert wird, die aus zwei Zylindern besteht. Er besteht aus Inhalts- und Sichtgraph.

12. Anhang B: Das Beispielprogramm

In diesem Kapitel werden einige Details der *Implementierung* des Beispielprogramms erläutert.

12.1. Aussehen und Bedienung des Beispielprogramms

Das Programm Lab3D erzeugt ein Fenster, wie es in Abbildung 13 dargestellt ist.

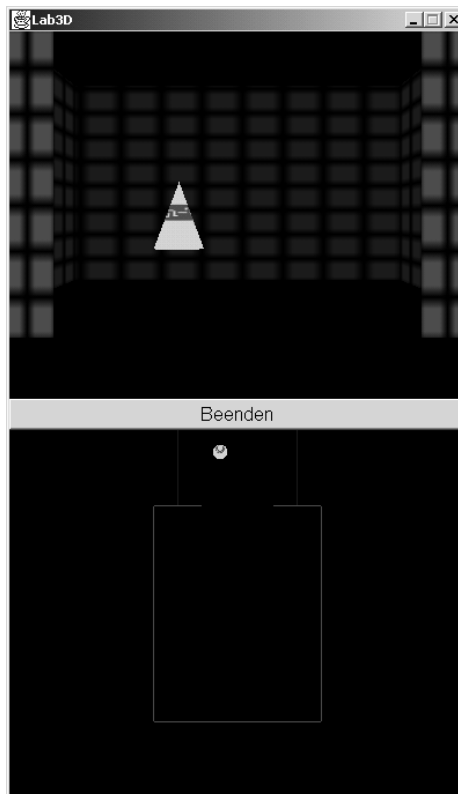


Abbildung 13: Das Beispielprogramm Lab3D kurz nach dem Start.

Im Kapitel 7 wurde bereits das Aussehen und die Bedienung des Beispielprogramms erläutert. Zur Erinnerung fassen wir in den kommenden Absätzen die Erläuterungen zusammen.

Im Fenster der Abbildung 13 werden drei Elemente angeordnet:

- oben ein rechteckigen Bereich für eine graphische 3D-Ausgabe,
- darunter ein Button mit der Aufschrift „Beenden“ und
- darunter ein zweiter rechteckiger Bereich für eine graphische 3D-Ausgabe.

Im oberen Ausgabebereich wird das Labyrinth dreidimensional dargestellt. Die Szene ist so dargestellt, dass der Betrachter sich anfangs im Labyrinth befindet.

In dem unteren Ausgabebereich ist permanent das Labyrinth in der Vogelperspektive zu sehen.

In beiden Ausgabebereichen ist ein gelbes kegelförmiges Objekt zu sehen, der sogenannte Bot.

12.2. Die wichtigsten 5 Klassen

Kommen wir nun zur Implementierung des Beispielprogramms. Das Programm besteht aus der Klasse `Lab3D` und aus dem Package `forLab3D`, das aus weiteren 17 Klassen besteht. Unter den 17 Klassen sind 6 Hilfsklassen, welche zwar die Implementierung erleichtern, jedoch keine wesentliche Funktionalität bieten. Von den übrigen 11 Klassen bilden 4 Klassen und die Klasse `Lab3D` das Skelett des Beispielprogramms. Diese fünf Klassen definieren die grundsätzliche Struktur des Programms:

- `Lab3D`
- `MySceneGraph_EgoEye`
- `MySceneGraph_BirdsEye`
- `MyLab_EgoEye`
- `MyLab_BirdsEye`

Zu unterscheiden sind also:

- 5 Klassen mit grundsätzlicher Bedeutung,
- weitere 7 wichtige Klassen und
- 6 weniger wichtige Hilfsklassen.

Im folgenden werden die fünf Klassen mit grundsätzlicher Bedeutung beschrieben.

Zur Benennung der Klassen: Die Namen aller 17 Klassen des Packages `forLab3D` fangen mit „My“ an. Dadurch sind die Klassen sofort von Java- oder Java 3D-Klassen zu unterscheiden.

12.2.1. Lab3D.java

Das Beispielprogramm heißt `Lab3D`, die zum Programmstart auszuführende Datei heißt dementsprechend `Lab3D.class`. Die zugehörige Sourcecode-Datei `Lab3D.java` enthält ausschließlich die Klasse `Lab3D`. Diese Klasse implementiert das `Lab3D`-Fenster mit den beiden Ausgabebereichen und dem Button. `Lab3D` ist eine Subklasse von `java.awt.Frame`. `Frame` ist wiederum eine Subklasse von `java.awt.Container`. Die Klasse `Container` steht für einen rechteckigen Bereich auf dem Bildschirm, in dem graphische Benutzerschnittstellen-Elemente wie Buttons oder Scrollbars eingefügt werden können. Ein `Frame` ist ein solcher rechteckigen Bereich, allerdings mit einem Rahmen um diesen Bereich und einer Titelzeile am oberen Rand.

`Lab3D` wird mit drei Objekten gefüllt:

- Einer Instanz der Klasse `forLab3D.MySceneGraph_EgoEye` für den oberen Ausgabebereich,
- einer Instanz der Klasse `java.awt.Button` und
- einer Instanz der Klasse `forLab3D.MySceneGraph_BirdsEye` für den unteren Ausgabebereich.

Die Methoden (insbesondere die Methode `init`) der Klasse `Lab3D` erledigen drei Aufgaben:

1. Die drei oben aufgelisteten Instanzen werden erzeugt,
2. die drei Instanzen werden in dem `Frame` `Lab3D` eingefügt,
3. das `Lab3D`-Fenster wird auf dem Bildschirm angezeigt.

Im Folgenden werden wir schrittweise zeigen, welche Klassen von welchen Klassen aufgerufen werden. Dabei werden nur die bereits angesprochenen Klassen berücksichtigt.

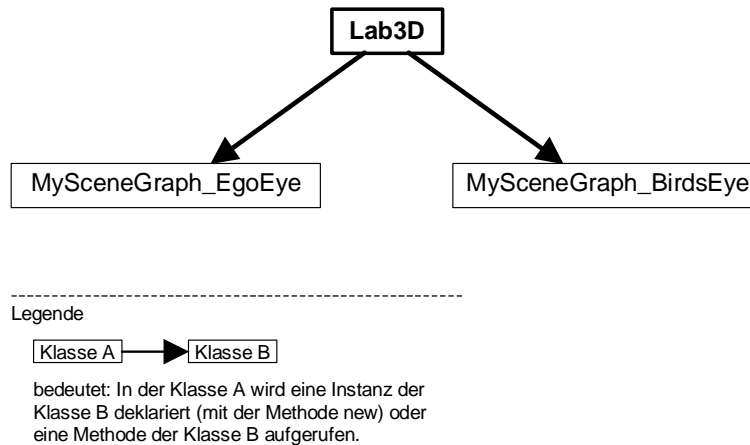


Abbildung 14: Welche Klasse greift auf welche Klasse zu? Die Abbildung stellt kein (!) Klassendiagramm dar.

12.2.2. MySceneGraph_EgoEye, MySceneGraph_BirdsEye

Die Klassen MySceneGraph_EgoEye und MySceneGraph_BirdsEye sind ebenfalls Subklassen von Container. Sie beherbergen jedoch keine Buttons oder Scrollbars, sondern jeweils ein javax.media.j3d.Canvas3D-Objekt. Analog zur java.awt.Canvas ist Canvas3D ein rechteckiger Bereich auf dem Bildschirm, in den ein Java 3D-Programm die (virtuellen) Elemente einer dreidimensionalen Szene „zeichnen“ kann. Es dient also der Darstellung einer dreidimensionalen Szene.

Für jeden der beiden Ausgabebereiche wird ein separater vollständiger Szenegraph aufgebaut. Gemäß den Klassennamen wird mit den Methoden von MySceneGraph_EgoEye der Szenegraph zum oberen Ausgaben-Bereich (EgoEye-Perspektive) aufgebaut, mit den Methoden von MySceneGraph_BirdsEye der Szenegraph zum unteren Ausgaben-Bereich (Vogelperspektive). Die Methoden benutzen weitere 7 Klassen des forLab3D-Packages.

Die Abbildung 15 zeigt die beiden Szenegraphen.

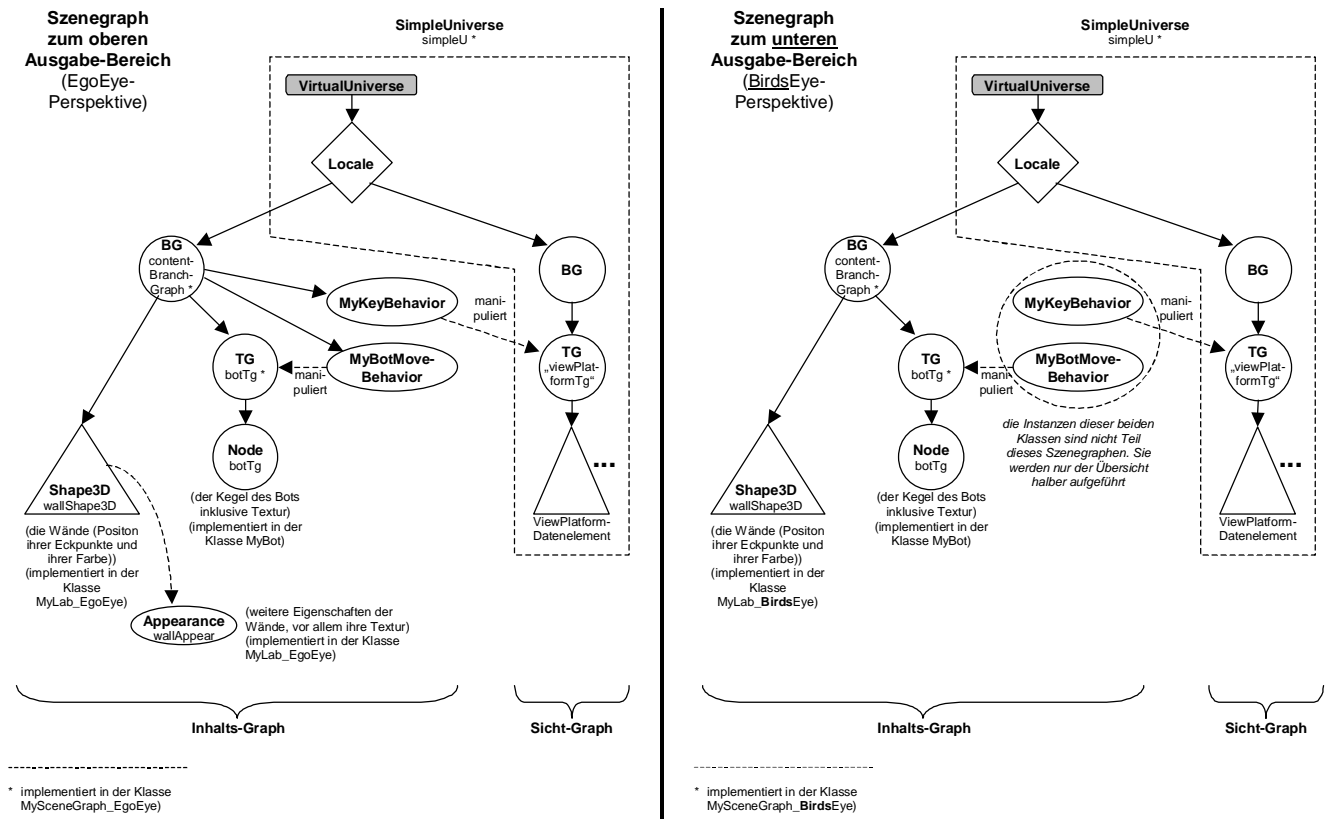


Abbildung 15: Die beiden Szenegraphen des Beispielprogramms.

Bei beiden Szenegraphen wird das gleiche Labyrinth dargestellt sowie der gleiche sich bewegende Bot. Die beiden Szenegraphen unterscheiden sich nur in drei Dingen:

- **Kein Textur-Objekt im rechten Szenegraphen:**

Im Szenegraph zum oberen Ausgabebereich werden die Wände als rechteckige Flächen dreidimensional dargestellt. Im Szenegraph zum unteren Ausgabebereich werden die Wände als Linien dargestellt. Das hat zwei Gründe: Erstens ist die Darstellung von Linien schneller als die von Flächen. Und zweitens würde eine als ebene Fläche dargestellte Wand unsichtbar, wenn der Betrachter sich genau über der Wand aufhält. In diesem Fall liegt seine Position in der Ebene, die auch die Wand enthält.

Der Unterschied der Darstellung ist allerdings im obigen Szenegraphen nicht sichtbar, dafür eine andere Unterscheidung: Die Wände sind texturiert, die Linien nicht. Für die Textur enthält der linke Szenegraph in der linken unteren Ecke eine `Appearance`-Instanz. Der rechte Szenegraph benötigt nichts Vergleichbares.

- **Manipulierung des Beobachters:**

Der linke Szenegraph enthält eine `MyKeyBehavior`-Instanz. Durch die `MyKeyBehavior`-Instanz kann der linke Szenegraph Tastatur-Eingaben verarbeiten. Diese Instanz manipuliert die `TransformGroup`-Instanz `viewPlatformTg` auf der rechten Seite des linken Szenegraphen. Die `viewPlatformTg`-Instanz hat Einfluss auf die Situation des Betrachters, also seine Position und seine Blickrichtung. Die Position des Betrachters im oberen Ausgabebereich liegt innerhalb des Labyrinths, er blickt in Richtung der negativen z-Achse. Die Position des Betrachters im unteren Ausgabebereich liegt oberhalb der Position des Betrachters im oberen Ausgabebereich, er blickt auf das Labyrinth herab.

Der rechte Szenegraph enthält keine `MyKeyBehavior`-Instanz. Jedoch wird seine Instanz `viewPlatformTg` von der `MyKeyBehavior`-Instanz des *linken* Szenegraphen manipuliert. Es gibt also nur eine `MyKeyBehavior`-Instanz, die auf beide Szenegraphen zugreift.

Drückt man beispielsweise eine Taste der Tastatur, die den Betrachter um 30 Grad nach links drehen lässt, passiert folgendes: Die `MyKeyBehavior`-Instanz empfängt dieses Ereignis. Daraufhin erstellt die Instanz eine 4x4-Matrix bzw. eine `Transform3D`-Instanz, welche die neue Lage (nach dem Drehen um 30 Grad) des Betrachters repräsentiert. Die Lage (Position und Blickrichtung) des Betrachters im oberen Ausgabebereich (EgoEye-Perspektive) unterscheidet von der im unteren Ausgabebereich (Vogelperspektive). Deshalb wird für jede der beiden Ansichten eine eigene `Transform3D`-Instanz definiert. Anschließend weist die `MyKeyBehavior`-Instanz die `Transform3D`-Instanzen den beiden `viewPlatformTg`-Instanzen zu, und zwar die `Transform3D`-Instanz der EgoEye-Perspektive dem linken Szenegraphen und die `Transform3D`-Instanz der Vogelperspektive dem rechten Szenegraphen.

- **Manipulierung des Bots:**

Der linke Szenegraph enthält eine `MyBotMoveBehavior`-Instanz. Durch die `MyBotMoveBehavior`-Instanz wird immer wieder nach Ablauf einer bestimmten Zeit (z.B. 0,2 Sekunden) die Position des gelben Kegels (des Bots) verändert. Dazu manipuliert diese Instanz die `TransformGroup`-Instanz `botTg` des linken Szenegraphen. Die `botTg`-Instanz hat Einfluss auf die Lage des Bots.

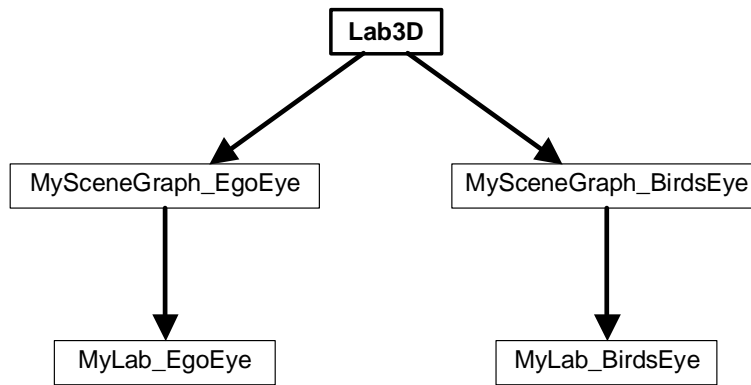
Der rechte Szenegraph enthält keine `MyBotMoveBehavior`-Instanz. Jedoch wird seine Instanz `botTg` von der `MyBotMoveBehavior`-Instanz des *linken* Szenegraphen manipuliert. Es gibt also nur eine `MyBotMoveBehavior`-Instanz, die auf beide Szenegraphen zugreift.

Sind die z.B. 0,2 Sekunden abgelaufen, passiert folgendes: Die Instanz von `MyBotMoveBehavior` empfängt dieses Ereignis. Daraufhin erstellt die Instanz eine 4x4-Matrix bzw. eine `Transform3D`-Instanz, welche die neue Lage des Bots repräsentiert. Die Lage des Bots ist im oberen und unteren Ausgabebereich identisch. Deshalb wird für beide Ansichten die gleiche `Transform3D`-Instanz definiert. Anschließend weist die `MyBotMoveBehavior`-Instanz die `Transform3D`-Instanz den beiden `botTg`-Instanzen zu.

12.2.3. MyLab_EgoEye, MyLab_BirdsEye

Durch die beiden Klassen `MySceneGraph_EgoEye` und `MySceneGraph_BirdsEye` wird jeweils ein Szenegraph aufgebaut. Beide Szenegraphen sind in der Abbildung 19 dargestellt.

Links in beiden Szenegraphen sehen Sie ein `Shape3D`-Objekt. Das `Shape3D`-Objekt repräsentiert die Wände des Labyrinths, genauer gesagt die Positionen der Eckpunkte der einzelnen Wände und die Farben der einzelnen Wände. Das `Shape3D`-Objekt im linken Szenegraphen wird in der Klasse `MyLab_EgoEye` erzeugt. Die Instanz dieser Klasse wird in der `MySceneGraph_EgoEye`-Instanz erzeugt. Das `Shape3D`-Objekt im rechten Szenegraphen wird in der Klasse `MyLab_BirdsEye` erzeugt, die Instanz dieser Klasse in der `MySceneGraph_BirdsEye`-Instanz. Diesen Zusammenhang stellt Abbildung 16 dar.



Legende



bedeutet: In der Klasse A wird eine Instanz der Klasse B deklariert (mit der Methode new) oder eine Methode der Klasse B aufgerufen.

Abbildung 16: Die Abbildung 14 wurde um zwei Klassen ergänzt.

12.3. Kurze Beschreibung aller 18 Klassen

In der folgenden Tabelle wird jede der 18 Klassen des Beispielprogramms Lab3D kurz beschrieben. Der Text in der Spalte mit der Überschrift „Beschreibung“ ist eine Zusammenfassung des Textes der HTML-Dokumentation (javadoc, siehe Anhang C).

In der zweiten Spalte der Tabelle ist für jede Klasse die ungefähre Anzahl Codezeilen (ohne Einrechnung von Kommentaren) angegeben. Insgesamt besteht das Beispielprogramm aus etwa 1500 Codezeilen (ohne Hinzurechnung von Kommentaren).

Klasse	Zeilen	Kurze Beschreibung
<i>Klassen mit grundsätzlicher Bedeutung</i>		
Lab3D.java	100	Lab3D erzeugt ein Fenster (Frame), das den oberen und unteren Ausgabebereich sowie einen Button darstellt. Lab3D ist die Klasse, die mit dem Java-Interpreter („java“) ausgeführt wird.
MySceneGraph_EgoEye	125	Dieser Container ist der obere Anzeigebereich des Lab3D-Fensters des Programms Lab3D. Man sieht das Labyrinth in der EgoEye-Perspektive.
MySceneGraph_BirdsEye	100	Dieser Container ist der untere Anzeigebereich des Lab3D-Fensters des Programms Lab3D. Man sieht das Labyrinth in der BirdsEye-Perspektive.
MyLab_EgoEye	95	Baut ein Shape3D-Objekt für den EgoEye-Szenegraphen auf, das die Geometrie, Farbe und Textur der Wände (Rechtecke) des Labyrinths repräsentiert.
MyLab_BirdsEye	70	Baut ein Shape3D-Objekt für den BirdsEye-Szenegraphen auf, das die Geometrie und Farbe der Wände (Striche) des Labyrinths repräsentiert.
<i>Weitere wichtige Klassen</i>		
MySceneGraph	15	Superklasse von MySceneGraph_EgoEye und MySceneGraph_BirdsEye ohne weitere wesentliche Aufgabe.
MyKeyBehavior	275	Liest die Tastatur-Eingaben und generiert die Reaktionen darauf. Je nach gedrückter Taste wird die Position oder/und die Blickrichtung des Betrachters

		in der Szene verändert. MyKeyBehavior ist eine Subklasse von Behavior.
MyBotMoveBehavior	110	Realisiert die Bewegung des virtuellen „Mitspielers“ (Bot). MyBotMoveBehavior ist eine Subklasse von Behavior.
MyBot	30	Realisiert die Geometrie des virtuellen „Mitspielers“ (Bot). Der Bot besteht aus einem Kegel. Auf dem Kegel ist eine vor allem gelbe Textur aufgetragen. Der Bot wird von der Lab3D-Anwendung animiert, er bewegt sich also von selbst.
MyWallData	65	Jede Instanz dieser Klasse enthält die Position und Farbe einer einzelnen Wand des Labyrinths und bietet Methoden zum Zugriff auf diese Daten. Die Instanzen dieser Klasse werden in einer Instanz der Klasse MyLabData gespeichert.
MyLabData	135	Initialisiert die Position und Farbe jeder einzelnen Wand des Labyrinths und bietet Methoden zum Zugriff auf diese Daten. Für jede einzelne Wand wird ein Objekt der Klasse MyWallData erstellt. Jedes dieser Objekte enthält die Daten (Position und Farbe) einer Wand. Die MyWallData-Objekte werden in einem Objekt der Klasse java.util.Vector abgelegt.
MyLab	5	Superklasse von MyLab_EgoEye und MyLab_BirdsEye ohne weitere wesentliche Aufgabe.
<i>Weniger wichtige Hilfsklassen</i>		
MyStartValues	40	Bietet Methoden zum Auslesen der Startwerte für die Position und die Blickrichtung des Betrachters (das heißt des menschlichen Spielers) und des vom Computer gesteuerten Spielers, des sogenannten Bots.
MySetTransform	150	Bietet Klassenmethoden (statische Methoden), mit denen komfortabel TransformGroup-Objekte mit definierten geometrischen Transformationen erzeugt werden können.
MyConstraint	40	Stellt eine Klassenmethode zur Verfügung, mit der ein Element der graphischen Benutzeroberfläche komfortabel und genau in einem Frame-Objekt angeordnet werden kann.
MyLayout	25	Legt für ein beliebiges java.awt.Frame-Objekt einige Eigenschaften fest.
MyColors	15	Stellt einige Farben als Color3f-Instanzen zur Verfügung. Die Farben können beispielsweise verwendet werden, um den Wänden des Labyrinths eine Farbe zuzuweisen. Besteht nur aus Konstanten und enthält keine Methoden.
MyDebug	100	Bietet Klassenmethoden (statische Methoden) zur komfortablen und strukturierten Ausgabe von Daten auf dem Bildschirm, genauer gesagt auf der Kommandozeile. Die Daten werden als Parameter übergeben. Die Benutzung der Klassenmethoden eignet sich insbesondere zum Testen bzw. Debuggen eines Programms.

Die folgende Abbildung stellt die Zusammenhänge zwischen den Klassen dar.

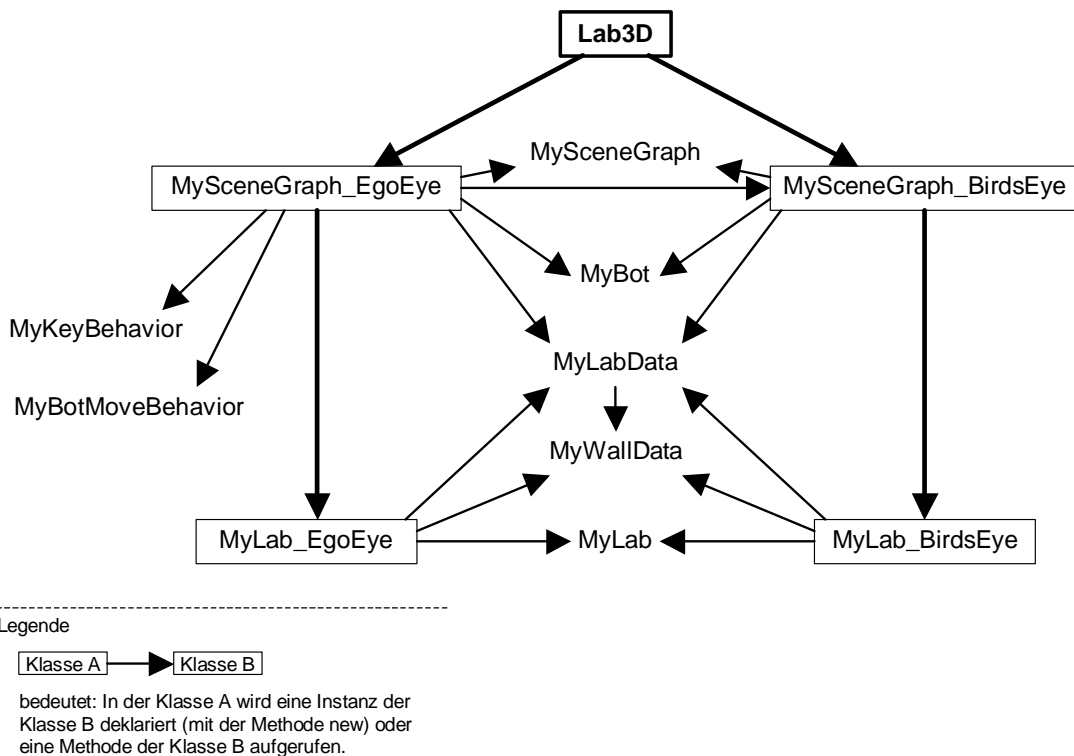


Abbildung 17: Welche Klasse greift auf welche Klasse zu? Diesmal für alle Klassen des Beispielprogramms bis auf die Hilfsklassen.

13. Anhang C: javadoc

Nehmen wir einmal an, Sie wollten den Sourcecode der Datei/Klasse `test.java` dokumentieren. Sie können mehrzeilige Kommentare in den Sourcecode einfügen, die mit `/*` beginnen und mit `*/` enden. Sie können in Java mehrzeilige Kommentare jedoch ebenso mit `/**` beginnen und mit `*/` enden lassen. Falls sie dann

```
javadoc Test.java
```

eingeben, filtert das Programm `javadoc` aus der Datei `Test.java` diese Kommentare heraus. In der Datei ist nur die Klasse `Test` vorhanden. `javadoc` erstellt automatisch eine HTML-Datei, in der die Klasse `Test` dokumentiert ist. `javadoc` ist ein sogenannter Dokumentationsgenerator. Er generiert eine Dokumentation im HTML-Format für das in der Kommandozeile angegebene Paket bzw. für die angegebenen Java-Quelldateien. `javadoc` ist wie `java` oder `javac` eines der im Java SDK 2 enthaltenen Standard-Programme.

Bei `javadoc` werden Leerzeichen und Sternchen (*) am Anfang einer Zeile innerhalb des Kommentars nicht berücksichtigt. Es folgt ein kurze Beispielklasse:

```

/**
 * Die Klasse Test tut nichts. Aber
 * f&uuml;r einen Test reicht das.
 * @author Der Test-Creator
 * @author Der zweite im Bunde
 * @version 1.0
 */
public class Test {

    /**
     * Konstruktor, erster Satz. Zweiter
     * Satz. Und dritter Satz.
     * @author Mr. Nobody
     * @param intInPut Da kann man
     *         jeden int-Wert reintun.
     * @return true, falls der
     *         Parameterwert 0 ist.
     *         sonst false.
     * @see    java.lang.Math
     */
    /* Dieser mehrzeilige Kommentar
     * wird von JavaDoc ignoriert.
     */
    // Auch dieser einzeilige Kommentar.
    public boolean TestEqual0(int intInPut){
        return (intInPut == 0);
    }
}

```

Wenn Sie die von javadoc erstellte Datei „index.html“ mit einem HTML-Browser öffnen, wird die Startseite der HTML-Dokumentation dargestellt

Auch die HTML-Dokumentation des Beispielprogramms Lab3D ist mit javadoc generiert worden. Sie ist im Verzeichnis Lab3D\Dokumentation zu finden. Um die Dokumentation zu Lab3D erneut zu erzeugen, können Sie folgenden Befehl verwenden:

```
javadoc -d Dokumentation Lab3D.java forLab3D
```

14. Anhang D: Kurzeinführung in UML-Klassendiagramme

Die Syntax der Klassenbeschreibungen orientiert sich an UML (Unified Modelling Language). UML ist eine Sammlung von Analyse- und Designnotationen und bietet den Entwicklern die Möglichkeit, den Entwurf und die Entwicklung von Softwaremodellen auf einheitlicher Basis zu diskutieren. UML wird seit 1998 als Standard angesehen. UML umfasst eine Vielzahl von Diagrammtypen, die das Verhalten und den Aufbau eines Softwaresystems beschreiben: Anwendungsfalldiagramme, Zustandsdiagramme, Sequenzdiagramme, Interaktionsdiagramme und Klassendiagramme. Für die Beschreibung einer Klassenbibliothek ist das Klassendiagramm am wichtigsten, deshalb wird die entsprechende Syntax kurz dargestellt. Klassendiagramme in UML erlauben sehr differenzierte Ausdrucksmöglichkeiten, was sich in einer Vielzahl von Symbolen niederschlägt.

Abbildung 19 zeigt die Symbole für Klassen. Es werden drei Klassentypen verwendet:

- Klassen allgemein
- Abstrakte Klassen sind Klassen, von denen keine Instanz gebildet werden kann,
- Interfaces enthalten selbst keinen Code. Es wird nur ein Satz von Methodenköpfen angegeben.

Abbildung 19 zeigt die Symbole für die Beziehungen der Klassen untereinander:

- Vererbung: Eigenschaften werden von einer Klasse an eine Unterklasse vererbt. Die Unterklasse kann zusätzliche Eigenschaften besitzen, ist somit spezieller als die Oberklasse.
- Realisierung: eine Klasse realisiert ein Interface, d.h. sie implementiert alle Methoden, die durch das Interface vorgeschrieben sind.
- Abhängigkeit: eine Klasse ist von einer anderen Klasse abhängig. Ändert sich die Schnittstelle der unabhängigen Klasse, muß die Schnittstelle der abhängigen Klasse angepaßt werden.
- Aggregation: die Aggregation beschreibt die Zusammensetzung eines Objekts aus einer Menge von Einzelteilen. Die Aggregation kann eine Kardinalität haben, d.h. es kann angegeben werden, wie viele Teile dem Ganzen zugeordnet werden.
- Komposition: die Komposition ist ein Spezialfall der Aggregation. Eine Komposition liegt dann vor, wenn die Teile nur im zusammengesetzten Objekt existieren.

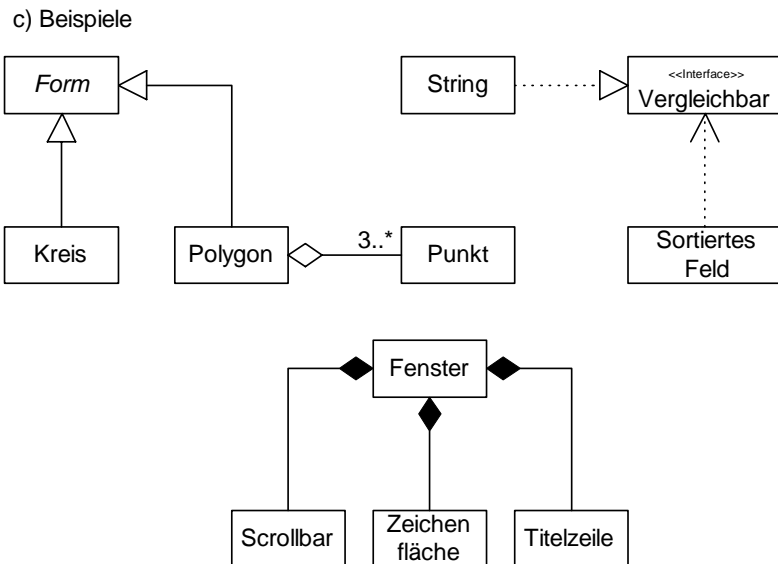
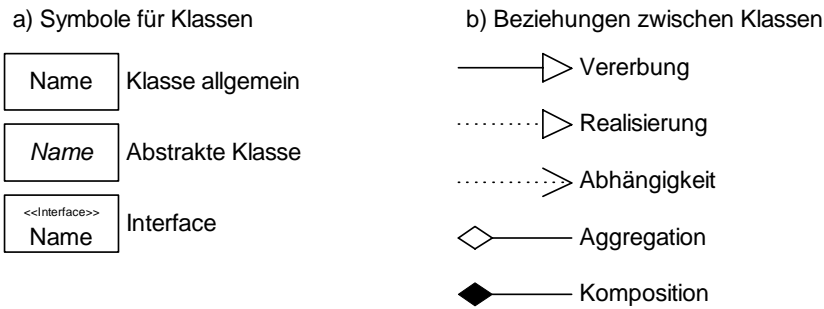


Abbildung 19: Syntax von UML-Klassendiagrammen

Abbildung 19 zeigt einige Beispiele für die Verwendung der Klassen und Relationen.

Im ersten Beispiel gibt es eine abstrakte Klasse Form, von der die Klassen Kreis und Polygon abgeleitet sind. Von der Klasse Form kann keine Instanz gebildet werden. Die Klasse Polygon besteht aus mehreren Objekten der Klasse Punkt.

Das zweite Beispiel zeigt die Realisierung eines Interfaces. Das Interface Vergleichbar wird von solchen Klassen realisiert, deren Inhalte miteinander vergleichbar sind. Die Klasse String realisiert dieses Interface. Benötigt wird diese Eigenschaft von der Klasse SortiertesFeld, die beliebige vergleichbare Objekte aufnimmt und in eine Sortierreihenfolge bringen kann.

Das letzte Beispiel zeigt eine Komposition. Ein Fenster besteht aus einer Scrollbar, einem Zeichenfeld und einer Titelzeile. Die Teile existieren nur solange, wie das Fenster existiert. Deshalb wird eine Komposition statt einer Aggregation eingetragen.