Institut für Infomatik der Humboldt-Universität zu Berlin
Rudower Chaussee 25
12489 - Berlin

Hauptseminar Informatik
Seminarleiter: S. Spolwig
WS 2003/2004

Unterrichtsentwurf / Halbjahresentwurf

Einführung in die Objektorientierte Programmierung

23.03.2004

Oliver Marquardt
174940
marquard@informatik.hu-berlin.de

André Prößdorf
166863
andre-proessdorf@web.de

Inhaltsverzeichnis:

1 Bedingungsanalyse

1.1 Allgemeine Voraussetzungen

Der Grundkurs besteht aus 12 Teilnehmern im Alter von 17 bis 19 Jahren, sechs Schülerinnen und sechs Schüler. Eine Schülerin, Janett wiederholt die 12. Klasse noch einmal. Alexander war in der 11.Klasse als Austauschschüler ein Jahr in Amerika. Bianca hat sehr lang durch Krankheit gefehlt.
Das Gesamtleistungsniveau der Gruppe ist differenziert. Zu den Besten gehören: Stephan Natalie und Andy. Zu den leistungsschwachen Schülern, die große Schwierigkeiten bei der Bewältigung der allgemeinen Informatikaufgaben haben, zählen Janett, Julia und Fabian.

1.1.1 Sitzplan

Matthias Stephan	Andy Ramona
Toni Janett	Julia Natalie Fabian
Sabrina Bianca	Alexander

Lehrer

1.2 Spezielle Voraussetzungen

Alle Schüler der Gruppe bis auf ALEXANDER besuchten im letzten Jahr den Grundkurs Informatik. Dort beschäftigten sie sich mit ausgewählten Problemen der Informatik und gewannen einen tieferen Einblick in den Umgang mit einer Datenbank (db2) und einem Textverarbeitungsprogramm sowie Office-Programme (Open Office) mit Index, Suchfunktion und Maske. Zusätzlich arbeiteten die Schüler mit dem Dateisystem (Linux, ext3) mit integrierter Suchfunktion. Weiterführend wurde in einer Internetdatenbank sowie im WWW gearbeitet. Im Anschluss ging der Kurs auf Datenschutz, Datensicherheit und Urheberrechte ein. Folglich wurde der Algorithmusbegriff eingeführt und einige Algorithmen mit alltäglichen Beispielen erläutert. Es wurde auf die Grenzen der Algorithmisierbarkeit von Problemen, Algorithmen in der Mathematik und in den Naturwissenschaften und Notation von Algorithmen eingegangen (verbal, grafisch, piktografisch).

2 Halbjahresentwurf

2.1 Halbjahresablauf

Stunde	Thema	Inhalt	Lehr-form
1 – 2	Einführung des Objektbegriffs	gezielte Objektbildung aus der realen Welt, Attribute und Methoden	UG
3 – 4	Objektspiel	Die Schüler ,,spielen`` in kleinen Gruppen selbst Objekt, ein Schüler pro Gruppe steuert Objekte Überleitung zu Java Karel, Einführung der UML - Klassennotation an diesem Beispiel	GA
5 – 6	UML 1	Implementierung Java Karel in UMLed www.kubitz-online.de	ZGA
7 – 8	Einführung in Java Teil 1	Die Schüler sollen erste Kenntnisse über Kontrollstrukturen mit Java Karel erarbeiten. http://www.educeth.ch/informatik/interaktiv/javakarel/	ZGA
9 – 10	Einführung in Java Teil 2	Fortsetzung	ZGA
11	LEK	Kontrollstrukturen, Objekte, Attribute und Methoden, Klassen	EA
12	Fangeballspiel	Einführung in die Projektphase	GA
13 – 14	UML 2	Erarbeitung eines Klassendiagramms zum Fangeballspiel in UMLed	GA
15 – 16	Physikalische und mathematische Grundlagen	Erarbeitung der zur Implementation notwendigen physikalischen und mathematischen Grundlagen. Schiefer Wurf, im 2D und 3D…	UG
17 – 18	Implementation 1	zunächst komplett ohne Grafik mittels Editor und javac – Kompiler	ZGA
19 – 20	Implementation 2		
21 – 22	Implementation 3		

23 – 24	Grundlagen der Grafikprogram-mierung	Hierfür müssen neben den Grafikbefehlen zum Zeichnen von Linien und Punkten auch die mathematischen Grundlagen der räumlichen Darstellung erläutert werden. Mit fortschreitender z-Koordinate sollte auch eine Verjüngung des Raumes nach hinten berücksichtigt werden, die dazu benötigte Konstante C sollte in der Größenordnung der halben Bildschirmbreite In Pixel liegen.	UG
25 – 26	Implementation der grafischen Darstellung (1)	Erstellung und Umsetzung einer dreidimensionalen grafischen Darstellung des gegebenen Spielfeldes mit dem Werfer und Steinen.	ZGA
27 – 28	Implementation der grafischen Darstellung (2)		
29 - 30	Implementation der grafischen Darstellung (3)		
31 – 32	Klausur		

2.2 Begründung der Stoffauswahl

An Beispielen aus der realen Welt wird zunächst eine Vorstellung des Objektbegriffes motiviert. Eine Definition des Objektbegriffes ist zum Ende der ersten Doppelstunde vorgesehen. Die Begriffe Attribut und Methode können an diesen Beispielen anschaulich erklärt werden. Diese Herangehensweise ermöglicht das Lernen grundlegender Begriffe anhand von alltäglichen Beispielen. Die Begriffe Klasse und Vererbung lassen sich mit diesen Beispielen sehr gut einführen.

Im Anschluss wird der Objektbegriff anhand eines von dem Lehrer vorgefertigten Spieles gefestigt:

Die Schüler werden in Vierergruppen aufgeteilt, ein Schüler „steuert“ seine drei Mitschüler mit vorgegebenen Methoden (Schritt vor, zurück, drehe, hebe auf, ...). Dieses Spiel soll die in der letzten Stunde gegebenen Grundlagen wiederholen und festigen. Außerdem sollen an dieser Stelle Gelegenheiten für Rückfragen gegeben werden.

Im Anschluss sollen die Schüler im lehrergeleiteten Unterrichtsgespräch ein Klassendiagramm zum Objektspiel erarbeiten. Hiermit soll das Klassendiagramm der UML – Notation eingeführt werden. Die Erarbeitung findet an der Tafel statt.
In der folgenden Doppelstunde wird das Ablaufprinzip von Java Karel erklärt (Was kann Java Karel, welche Programmiersprache steckt dahinter). Danach sollen die Schüler mittels UMLed ein Klassendiagramm zu Java Karel entwickeln.

Mittels Java Karel sollen prinzipielle Kontrollstrukturen vermittelt werden. Siehe Unterrichtsentwurf 1.

Mit der anschließenden Lernerfolgskontrolle soll festgestellt werden, ob die Schüler die Grundlagen der Objektorientierung verstanden haben.

Im Anschluss an den Test wird in aufgelockerter Atmosphäre auf dem Schulhof ein Fangeballspiel gespielt. Während des Spieles sollen die Schüler überlegen welche Attribute und Methoden das Spiel beschreiben könnten. Nach dem Test kann man ohnehin keine sonderliche Lernmotivation der Schüler erwarten.

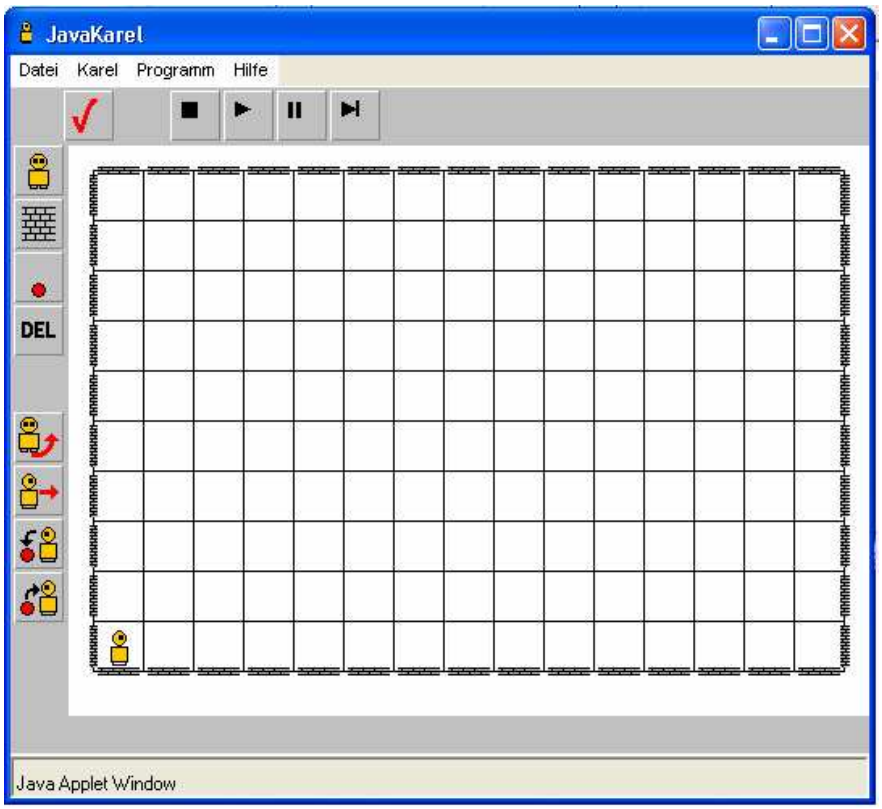
Danach sollen die Schüler mittels UMLed ein Klassendiagramm zu dem Fangeballspiel entwickeln. Die komplette Implementation dieses Projektes ohne grafische Oberfläche erfolgt gemäß Unterrichtsentwurf 2.
Im Anschluss an die grafiklose Implementation beschäftigen sich die Schüler mit der Erstellung einer grafischen Oberfläche. Hierfür werden die mathematischen Grundlagen dreidimensionaler Darstellungen erarbeitet. Anschließend werden grundlegende Grafikbefehle durch den Lehrer erklärt (Linien, Pixel, Kreise, Bildschirm usw.).

Abschließend folgt die Klausur über den Lernstoff des Halbjahres.

Die vorhandenen Reservestunden können gegebenenfalls mit einer Erweiterung des Fangeballspieles (mehrere Spieler, Wind Reibung, etc.) gefüllt werden.

3 Stundenentwurf (Einführungsphase)

3.1 Einführung in Kontrollstrukturen



Benutzeroberfläche von „Java Karel“ der Eth in Zürich

3.2 Sachanalyse

3.2.1 Kontrollstrukturen

Die Anweisungen innerhalb eines Programms werden sequentiell abgearbeitet. Diese Abarbeitungsreihenfolge (der Kontrollfluss) kann durch Kontrollstrukturen gezielt durch den Programmierer beeinflusst werden. Java verfügt über die aus anderen Programmiersprachen bekannten Kontrollanweisungen, und die Syntax ist stark an C bzw. C++ angelehnt.

if-Anweisung

Eine der häufigsten Problemstellungen in einem Programm ist, den Programmfluss in Abhängigkeit von einer oder mehreren Bedingungen zu steuern. Die *if*-Anweisung wird bei bedingten Kontrollflussverzweigungen verwendet. Die Syntax der *if*-Anweisung lautet:

```
if (<logische Bedingung>)
{
<Anweisung-1>
else
{
<Anweisung-2>]
}
```

Die *logische Bedingung* wird bewertet. Ergibt sie wahr (true), werden die Anweisungen direkt hinter dem *if* ausgeführt. Ergibt die *logische Bedingung* falsch (false), werden die Anweisungen hinter dem *else* ausgeführt. Man kann die *if*-Ánweisung auch ohne den *else*-Zweig schreiben. Ist in einem solchen Fall die *logische Bedingung* falsch, wird der geklammerte Anweisungsblock übersprungen, und das Programm wird hinter der geschlossenen Klammer fortgeführt.

Switch-Anweisung

Sollten Mehrfachentscheidungen getroffen werden, so kann die switch-Anweisung verwendet werden. Hier wird untersucht, ob ein Ausdruck einer von mehreren Konstanten gleicht. Gleicht der Ausdruck keiner der angegebenen Konstanten, kann ein sonst-Zweig vorgesehen werden. Das Schlüsselwort case steht vor jeder Konstanten, die mit dem Ausdruck verglichen werden sollen, das Schlüsselwort default leitet den sonst-Zweig ein. Die Syntax der switch -Anweisung lautet:

```
switch (<Ausdruck>) {
case <konstanter Ausdruck>:
<Anweisung> [<Anweisung> ...]
case <konstanter Ausdruck>:
<Anweisung> [<Anweisung> ...]
...
default:
<Anweisung> [<Anweisung> ...] }
```

Zuerst wird der *Ausdruck* hinter dem Schlüsselwort *switch* ausgewertet. Dabei muß der Typ dieses Ausdrucks *char, byte, short* oder *int* sein, ansonsten meldet der Compiler einen Fehler. Der ermittelte Wert wird dann mit jedem der konstanten Ausdrücke hinter den *case*-Anweisungen verglichen. Wird ein passender Ausdruck gefunden, wird die Programmausführung an dieser Stelle fortgeführt und läuft dann bis zum Ende der Switch-Anweisung oder bis zum nächsten break. Wenn kein Fall zutrifft wird die Anweisung hinter *default* ausgeführt.

3.2.2 Schleifen

Um Teile eines Programms mehrfach auszuführen, benutzt man Schleifen. Java verfügt über drei unterschiedliche Schleifenkonstrukte. Alle drei Formen sind aus anderen Programmiersprachen bekannt. Die *for*-Schleife kann alternativ auch mit Hilfe der *while*-Schleife bzw. der *do-while*-Schleife formuliert werden:

for-Schleife

Eine Eigenschaft der *for*-Schleife ist, dass man bereits zu Beginn der Schleife wissen muss, wie oft sie durchlaufen werden soll. Die formale Syntax der *for*-Schleife lautet:

```
for (startwert; [<logischer Ausdruck>]; schleifenwert)
{
<Anweisung>
}
```

In den Klammern hinter dem Schlüsselwort *for* wird festgelegt, wie oft die Anweisung wiederholt wird. Zuerst wird der *startwert* initialisiert. Dann wird *logischer Ausdruck* überprüft. Ist er wahr, wird die *Anweisung* durchlaufen. Im letzten Schritt wird der s*chleifenwert* ausgewertet. Von da an wird immer wieder nacheinander *(logischer Ausdruck - Anweisung – schleifenwert)* abgearbeitet, bis der logische Ausdruck falsch wird.

while-Schleife

Die while-Schleife ist eine Struktur, mit der man Programmteile so lange wiederholen kann, bis eine Bedingung nicht mehr zutrifft. Die formale Syntax der *while*-Schleife lautet:

```
while ([<logischer Ausdruck>])
{
<Anweisung>
}
```

Ein *Logischer Ausdruck* wird bewertet. Ist er wahr (true), wird der Block *Anweisung* direkt hinter dem *while* ausgeführt. Ergibt *Logischer Ausdruck* falsch (false), wird die Programmausführung direkt hinter der schließenden Klammer der *while*-Schleife fortgesetzt. Wenn *Logischer Ausdruck* bereits beim ersten Mal falsch ist, kann es sein, dass sie überhaupt nicht durchlaufen wird. Um die *while*-Schleife zu verlassen, muss man dafür sorgen, dass irgendwann die Bedingung nicht mehr erfüllt ist.

do-while-Schleife

Die do-Schleife ist eng mit der *while*-Schleife verwandt. Der Unterschied zwischen den beiden besteht im Zeitpunkt der Prüfung, ob weitergemacht werden soll oder nicht. Bei *do* findet die Prüfung am Ende der Schleife statt.. Die Schleife wird daher mindestens einmal durchlaufen, während es bei der *while*-Schleife vorkommen kann, dass sie überhaupt nicht durchlaufen wird.
Die allgemeine Form einer *do*-Schleife sieht so aus:

```
Do
{
<Anweisung>
}
while ([<logischerAusdruck>]);
```

3.3 Lernziele

Die Schülerinnen und Schüler:

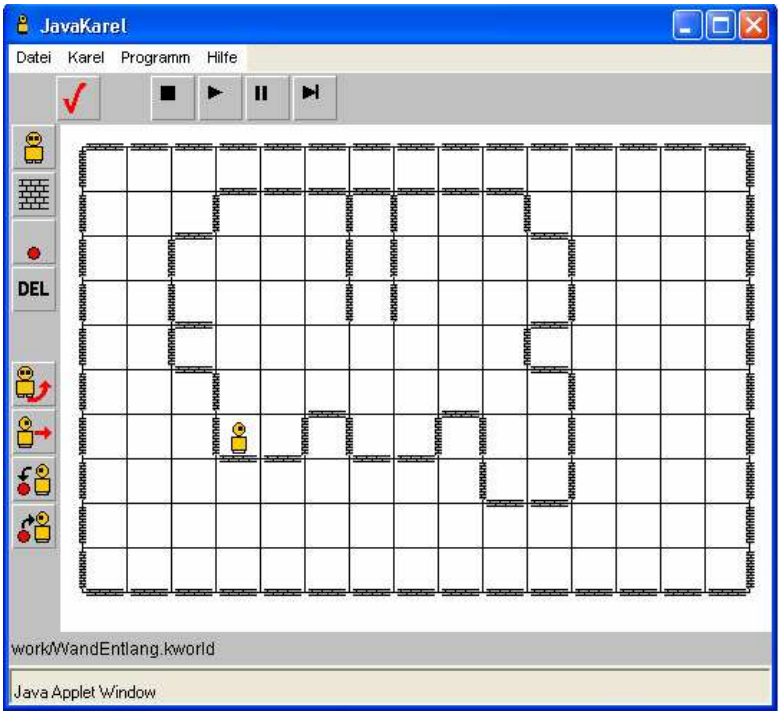
- sollen Kontrollstrukturen und ihre Funktion der Steuerung eines Programmablaufs kennen und anwenden können.
- erkennen in einer Aufgabenstellung das Teilproblem einer zu wiederholenden Anweisung und können die Problemlösung programmierunabhängig darstellen (verbal, grafisch)
- kennen die for-, while- und die do...while-Schleife und können sie voneinander unterscheiden
- nennen mögliche Probleme hinsichtlich der Auswahl einer von den drei Schleifen
- sollen die Umsetzung in „Java Karel“ kennen lernen

3.4 Unterrichtsgestaltung

Die Schülerinnen und Schüler werden in Zweiergruppen eingeteilt. Die Gruppen werden so gewählt, dass alle Gruppen in etwa dem gleichem Leistungsniveau entsprechen. Jede Gruppe soll nun den Roboter (Karel) einmal die Wand entlang (durch Steuerung mit den Buttons) gehen und die Schritte aufzeichnen lassen.

--> Die Schüler werden feststellen, dass dafür sehr viele Schritte notwendig sind. Der aufgezeichnete Quelltext wird dementsprechend lang sein.

Im Lehrer – Klassengespräch soll nun geklärt werden, wie man den Quelltext verkürzen könnte und auch sollte. Auffallen sollten die vielen Wiederholungen, welche eventuell auf eine Schleife zurückzuführen wären. Das ständige „gegen die Wand laufen“ ergibt immer eine Fehlermeldung, dies könnte man als Übergang zur if-Anweisung nutzten.



Nun wird eine Zusammenfassung vom Lehrer über die benötigten Kontrollstrukturen gehalten.

Danach soll jede Gruppe, mit der Hilfe der in der Kontrollleiste (klick Hilfe dann klick Sprachbeschreibung), einen so kurz wie möglich gestalteten Algorithmus implementieren.

Wenn alle Gruppen fertig sind, wird die Lösung von einer oder von zwei Gruppe vorgestellt, falls dies nicht möglich sein sollte, wird eine Musterlösung gegeben. Nach der Vorstellung, sollen die Schüler ihre eigenen Lösungen vergleichen und darüber mit den andern Schülern diskutieren. (Was könnte man besser, schneller oder eleganter programmieren)

Die Schüler die sich keine Lösung der Aufgabe erarbeiten konnten, ergänzen gegebenenfalls die fehlenden Elemente in ihrer Lösung, oder übernehmen sich die Lösung der Mitschüler / des Lehrers (Musterlösung).

Die Schülerinnen und Schüler sollen die Notwendigkeit von Kontrollstrukturen zur Steuerung und Gliederung von Algorithmen verstehen lernen. Sie sollen später diese Erkenntnisse in anderen Problemen oder Aufgaben anwenden können.

3.5 Tabellarischer Unterrichtsentwurf

Zeit	Phase	Inhalt	Methode	Medien
8:00 – 8:15	Java Karel Wiederholung	Die S. sollen sich mit der Handhabung von Karel vertraut machen, erstmal speziell nur die Bedienung über die Buttons. Sie sollen den Roboter einfache Bewegungen ausführen lassen. Lehrer zeigt einige Funktionen des Roboters über den Beamer.	LV/UG	Beamer Tafel
8:15 – 8:25	Gruppen-einteilung			
8:25 – 8:35	Aufgabe 1	Die S. sollen die Welt „*wandentlang.kworld*“ Laden und mit Hilfe der Buttons den Roboter die Innenseite der Wand entlang gehen lassen. Zusätzlich sollen sie alle Schritte die sie machen, aufzeichnen lassen.	GA	Beamer Computer
8:35 – 8:45	Auswertung	Die S. sollen nun die Ergebnisse im Unterrichtsgespräch vortragen. Hier sollen jetzt die Probleme gezeigt werden (wie oben besprochen) zu viele Aufrufe, Wiederholungen und Fehlermeldungen beim „gegen die Wand laufen“. Lehrer führt diese am Beamer zusätzlich vor.	UG	Beamer
8:45 – 9:00	Vorstellung	Der Lehrer stellt nun die Kontrollstrukturen vor und wie man sie in „Java Karel“ implementiert. S. sollen sich dabei beteiligen.	LV UG	Beamer Computer
9:00 – 9:20	Aufgabe 2	S. sollen nun einen möglichst kurz implementierten Algorithmus gestallten, der den Roboter unendlich oft die Wand entlang gehen läst.	GA	Computer
9:20 – 9:30	Auswertung	Alle Ergebnisse werden nun verglichen, die beste Lösung oder eine vom Lehrer wird als Musterlösung gezeigt und das Gesamtergebnis an dieser ausgewertet. Abschlussdiskussion	UG	Beamer
9:30	Pause	Stundenende		

3.6 Möglichkeiten zur Implementierung der Problemstellung

```
program WandEntlang(Karel);

turnRight()
{
   karel.turnLeft;
   karel.turnLeft;
   karel.turnLeft;
}

findWall()
{
   while (karel.frontClear)
   {
        karel.move;
   }
}

{

   findWall;  // Wand finden und links beruehren
   turnRight;

   while (!karel.loaded)    // Um Uhrzeigersinn freien Weg finden
   {
        karel.turnLeft;
        while  (!karel.frontClear)
        {
           turnRight;
        }
        karel.move;
   }
}
```

3.7 Didaktische Kommentare

Der Einstieg ist den vorangegangenen Stunden durch die Vorstellung der Programmier-umgebung Java Karel erfolgt. Hierfür wurde vom Lehrer gezeigt, durch welche Buttons sich der Roboter steuern lässt. Jeder Button hat eine bestimmte Eigenschaft (drehe links, hebe auf, leg ab und Schritt vor).
Die Schüler sollten Karel einige Schritte über den Bildschirm gehen und ihn auch mal drehen lassen. Anschließend wurde gezeigt wie man die Welt in seiner noch leeren Oberfläche gestaltet. Hierfür benötigt man die oberen vier Buttons (setze Karel, setze Wand, setze Ball und lösche Feld).

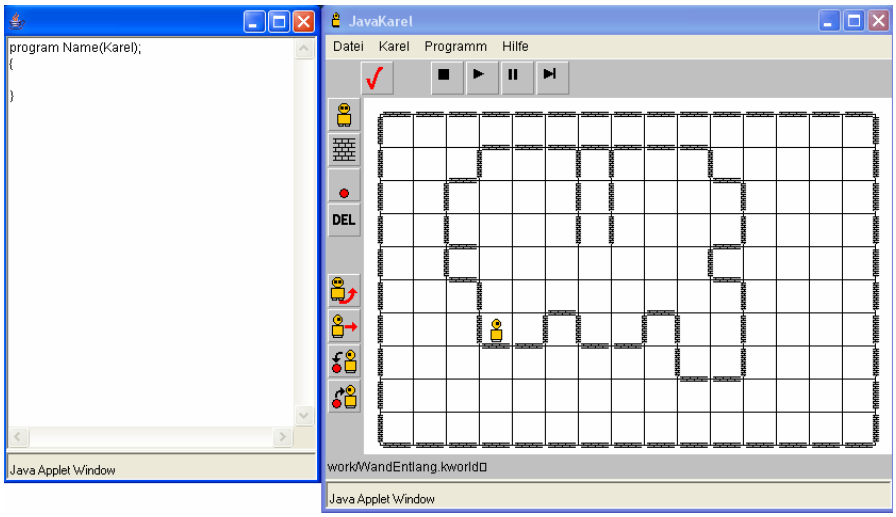
Die Schüler sollen sich in dieser Stunde eine kleine Welt gestallten z.B. ein Haus mit Tür, oder eine Rakete usw. und versuchen Karel in dieser Welt zu bewegen.
Die Schüler bemerken jetzt schon, dass Karel eine Fehlermeldung sendet, wenn er gegen eine Wand läuft.

Der Lehrer gibt nun eine Welt vor, die die Schüler auf ihren Arbeitsplätzen laden können. Dies geschieht wie folgt, in der Kontrollleiste „Datei“ klicken, danach „Welt laden“ klicken und die Datei *WandEntlang.kworld* öffnen.
Anschließend sollen die Schüler den Robotter an der Innenwand der geladenen Welt entlang laufen und zusätzlich ihre Schritte die sie machen aufzeichnen lassen, dies wird wie folgt gemacht, klick in der Kontrollleiste „Datei“, dann klick auf „neues Programm“ und es erscheint ein Fenster mit noch leerem Argument:

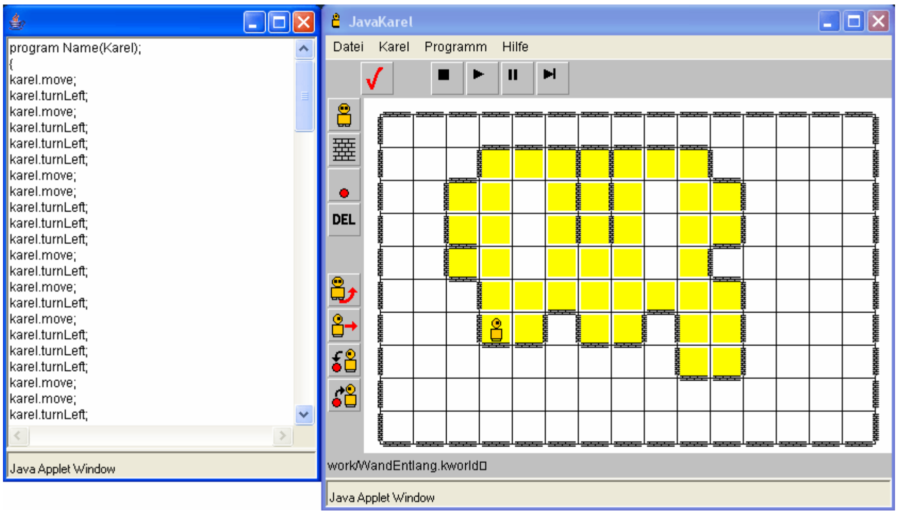
```
program Name(Karel);
{

}
```

Jetzt noch die Aufzeichnung aktivieren, klick in der Kontrollleiste „Karel“ dann „Optionen“ und das Häkchen bei „Kommandos aufzeichnen“ und „besuchte Felder färben“ setzen. Haben die Schüler den Roboter einmal rum laufen lassen, könnte das Programm sowie ihre Welt wie folgt aussehen:



Die Welt auf dem Arbeitsplatz, bevor Karel einmal an der Wand entlanggelaufen ist (das Programmfenster ist noch leer)



Die Welt auf dem Arbeitsplatz,
nachdem Karel einmal an der Wand entlanggelaufen ist
(das Programmfenster ist mit Programmtext gefüllt)

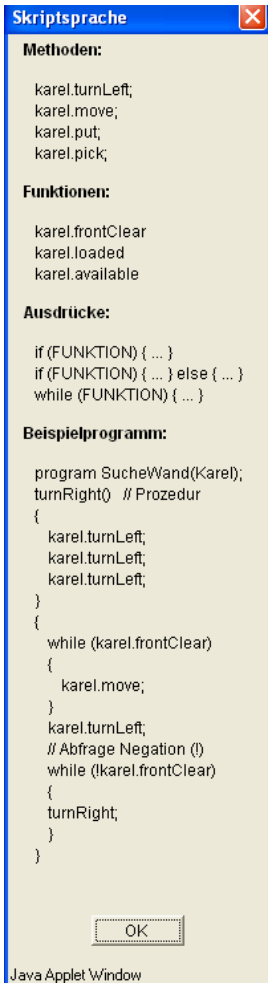
Der Programmtext könnte wie folgt aussehen:

```
program Name(Karel);
{
karel.move;
karel.turnLeft;
karel.move;
karel.turnLeft;
karel.turnLeft;
karel.turnLeft;
karel.move;
karel.move;
karel.turnLeft;
karel.turnLeft;
karel.turnLeft;
karel.move;
karel.turnLeft;
karel.move;
karel.turnLeft;
karel.move;
karel.turnLeft;
karel.turnLeft;
karel.turnLeft;
karel.move;
karel.move;
karel.turnLeft;
karel.turnLeft;
karel.turnLeft;
karel.move;
karel.move;
karel.turnLeft;
karel.move;
karel.turnLeft;
karel.move;
karel.move;
karel.turnLeft;
karel.move;
karel.turnLeft;
karel.turnLeft;
karel.turnLeft;
karel.move;
karel.move;
karel.turnLeft;
karel.turnLeft;
karel.turnLeft;
karel.move;
karel.turnLeft;
karel.move;
karel.turnLeft;
karel.move;
karel.turnLeft;
karel.turnLeft;
karel.turnLeft;
karel.move;
karel.turnLeft;
karel.move;
karel.move;
karel.turnLeft;
karel.move;
karel.move;
karel.move;
karel.turnLeft;
karel.turnLeft;
karel.turnLeft;
karel.move;
karel.turnLeft;
karel.turnLeft;
karel.turnLeft;
karel.move;
karel.move;
karel.move;
karel.turnLeft;
karel.turnLeft;
karel.move;
karel.move;
karel.move;
karel.turnLeft;
karel.turnLeft;
karel.turnLeft;
karel.move;
karel.turnLeft;
karel.turnLeft;
karel.turnLeft;
karel.move;
karel.move;
karel.move;
karel.turnLeft;
karel.move;
karel.move;
karel.turnLeft;
karel.move;
karel.turnLeft;
karel.turnLeft;
karel.turnLeft;
karel.move;
karel.turnLeft;
karel.move;
karel.turnLeft;
karel.move;
karel.turnLeft;
karel.turnLeft;
karel.turnLeft;
karel.move;
karel.turnLeft;
karel.turnLeft;
karel.turnLeft;
karel.move;
karel.turnLeft;
karel.turnLeft;
karel.move;
karel.turnLeft;
karel.turnLeft;
karel.turnLeft;
karel.move;
karel.move;
karel.turnLeft;
}
```

Jetzt sollen die Schüler beschreiben was sie beim Bewegen des Roboters festgestellt haben, explizit sollen sie wiedergeben welcher Programmbefehl, welche Bewegung ausführen lässt. Im Anschluss soll festgestellt werden, was an dem Programmtext besonders auffällig ist. (Viele Wiederholungen, ständige Fehlermeldungen wenn Karel gegen die Wand läuft)
Dieser Einstieg soll als Motivation der Schüler fungieren, er zeigt sehr deutlich, dass es in dieser Welt viele Wiederholungen und ständig Fehlermeldungen auftauchen. Es wurde somit ein Übergang in die Einführung in Kontrollstrukturen geschaffen.

Anschließend werden im Lehrervortrag die nötigen Kontrollstrukturen vermittelt und danach mit dem Schüler im Unterrichtsgespräch wiederholt, so dass sie gefestigt werden.

Zum Ende der Stunde wird eine Aufgabe gestellt, die zum Ziel hat, einen möglichst kurzen Algorithmus zu implementieren der Karel unendlich oft an der Wand entlanglaufen lässt.
Falls die Schüler noch Hilfe bei der Implementierung von Kontrollstrukturen benötigen, können sie mit einem Klick auf „Hilfe“ in der Kontrollleiste die „Sprachbeschreibung“ öffnen. Hier sind kurz alle wichtigen und nötigen Programmstrukturen erläutert. Siehe rechts.



Skriptsprache von Java Karel

Alternative Varianten

Lehrervortrag

Schülervortrag

Vorbereitete Hausaufgabe

Herleitung der Kontrollstrukturen im Unterrichtsgespräch

4 Stundenentwurf (Weiterführende OOP)

Nachdem die Schüler erste Erfahrungen mit der objektorientierten Programmierung gesammelt haben, soll das vorhandene Wissen an einem größeren Projekt gefestigt und erweitert werden. Das Programm „Fangeball“ soll hierzu als Simulation eines schiefen Wurfes im Dreidimensionalen implementiert werden.

4.1 Sachanalyse

4.1.1 Objektorientierung

Das Programm „Fangeball“ implementiert eine Umwelt, die durch Ausdehnung in x- und y- Richtung (Länge ***laenge*** und Breite ***breite***) sowie eine fortlaufende Zeit ***zeit*** und eine Fallbeschleunigung ***g*** beschrieben wird.

In dieser Umwelt befindet sich der Werfer, der durch seine Position ***xw, yw***
seine Körpergröße ***hoehe***, seine Kraft ***kraft*** bzw. Maximalkraft ***maxkraft*** und einen Winkel in der Ebene ***phi*** bzw. einen Abwurfwinkel zwischen Ebene und Wurfrichtung ***theta*** beschrieben wird. Der Werfer hat zudem eine Anzahl ***anz*** von Wurfgegenständen.
Der Werfer hat die Möglichkeit, sich zu drehen, vorwärts zu laufen, Wurfgegenstände aufzunehmen und unter seinem veränderbaren Abwurfwinkel mit variabler Kraft zu werfen.
Auch kann der Werfer seine Position an eine später implemetierte grafische Benutzeroberfläche zurückgeben.

Die Wurfgegenstände befinden sich zunächst auf ihren Positionen ***xg,yg*** in der Ebene,
sie haben jedoch als einzige Objekte in diesem Programm auch eine Koordinate für die Flughöhe: ***zg***. Ein Wurfgegenstand wird zudem mit seiner Masse ***masse***, einer Konstanten ***eta***, die seinen Luftreibungswiderstand berücksichtigt, sowie einer Ausgangsgeschwindigkeit
geschw und einem Abwurfwinkel ***theta_g*** beschrieben.
Ein Wurfgegenstand hat lediglich die Möglichkeit, seine Position (hier allerdings drei Zahlenwerte!) zurückzugeben.

Es besteht die Möglichkeit, zur Erweiterung des Programms noch Wind zu berücksichtigen.
Dieser wäre durch Geschwindigkeit und Winkel über der x-Achse beschrieben und könnte eben diese beiden Variablen verändern.

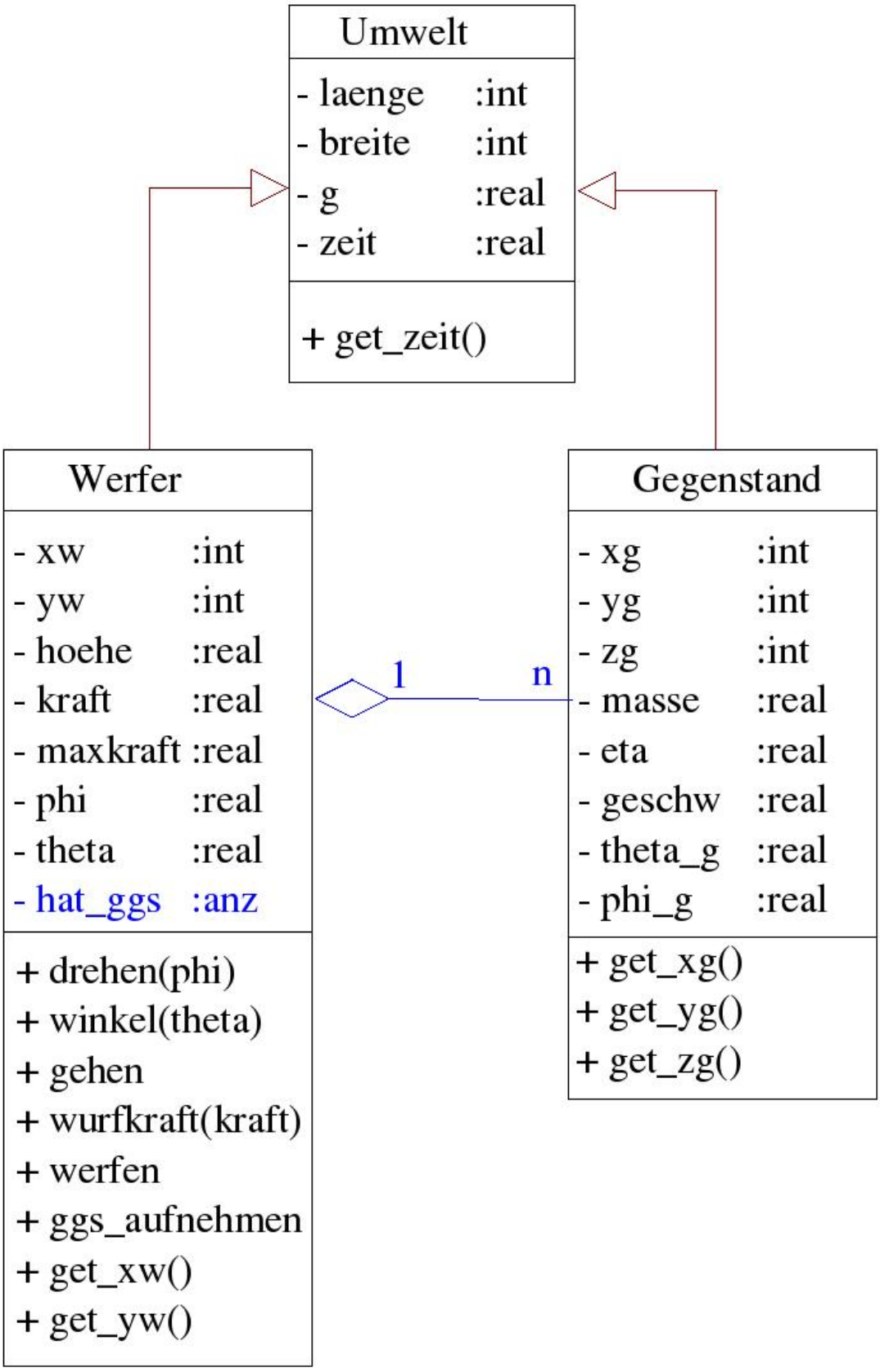


Bild: Klassendiagramm zum Fangeballspiel

4.1.2 Physikalisch

Unser Wurfkörper gehorcht den Gesetzen der geradlinig-gleichförmigen Bewegung in x- und y-Richtung sowie einer gleichmäßig beschleunigten Bewegung in z-Richtung. Seine Bewegung wird durch Abwurfwinkel, Abwurfgeschwindigkeit und Abwurfrichtung sowie die Fallbeschleunigung unserer Umwelt bestimmt.

Die Abwurfgeschwindigkeit ergibt sich aus Körperkraft des Werfers und Masse des Körpers.
Mit **F=m*a** und **v=a*t** ergibt sich eine Abwurfgeschwindigkeit von $\mathbf{v_0=F*t/m}$. Die Zeit **t** ist hierbei die Zeit, die der Werfer den Gegenstand beschleunigt, bevor dieser frei zu fliegen beginnt. Für unsere Bedürfnisse werden wir eine konstante Zeit annehmen, um einen direkten Zusammenhang zwischen Abwurfgeschwindigkeit und Kraft des Werfers bzw. Masse des Flugkörpers herzustellen.

Die Bewegung parallel zur Ebene läßt sich aus zwei geradlinig-gleichförmigen Bewegungen
zusammensetzen. Dabei sind die Koordinaten des Körpers zu berechnen aus:

$$x=v_0*\cos\square \quad \sin\square*-ŞŅĎ *t+x_0$$
$$y=v_0*\sin\square \quad \sin\square*-ŞŅĎ *t+y_0$$

In z-Richtung liegt uns eine gleichförmig-beschleunigte Bewegung vor. Diese setzt sich aus der Ausgangsgeschwindigkeit, der Abwurfhöhe und der Fallbeschleunigung zusammen:

$$z=-g/2*t^2+v_0*\cos\square*-ŞŅĎ *t+h$$

Für eine bereits oben angeschnittene Erweiterung des Programms müßte der Wind berücksichtigt werden: Die Koordinaten parallel zur Ebene, x und y müßten durch eine Superposition mit der Windgeschwindigkeit additiv beeinflußt werden. Dies ist jedoch lediglich ein von Windgeschwindigkeit und Windrichtung abhängiger Summand, der in einer Richtung mit Cosinus-, in der anderen mit Sinusabhängigkeit eingeht.

Als zweite Erweiterungsmöglichkeit kann hier Luftreibung berücksichtigt werden. Diesem Zweck dient die Konstante **η** (Eta). Es kann hier vereinfacht mit einer der Bewegung entgegengerichteten Kraft $\mathbf{\eta *v^2}$ gerechnet werden, da bei für einen Wurf üblichen Geschwindigkeiten turbulente Strömung auftritt.

4.1.3 Mathematisch

Perspektivisch soll im späteren Unterrichtsverlauf eine grafische Benutzeroberfläche mit der Bewegung des Wurfkörpers implementiert werden. Hierbei stellt vor allem die Umrechnung der dreidimensionalen Bewegung in eine zweidimensionale Bildschirmausgabe ein Problem dar. Die benötigten Formeln sind:

$$z_{screen}=z-\sin\pi\ /4*C/(C+y)$$
$$x_{screen}=x+\sin\pi\ /4*C/(C+y)$$

Die Konstante **C** ist hierbei selbstständig zu ermitteln, sie sollte in der Größenordnung der halben Bildschirmbreite (in Pixel) liegen, also bei einer 800 x 600-Auflösung: **C=400**.

4.2 Lernziele

Die Schüler sollen in der beschriebenen Unterrichtseinheit folgende Lernziele erreichen:

- Das Entwerfen von Klassendiagrammen in der UML-Notation soll gefestigt werden.
- Die Schüler können die beschriebenen Klassen Umwelt, Werfer, Gegenstand implementieren.
- Die Schüler kennen den Unterschied zwischen Klasse und Objekt.
- Fächerübergreifendes Wissen aus der Mathematik und der Physik wird gefestigt.
- Das Wissen über Schleifen und Bedingungen wird gefestigt und angewandt.

4.3 Unterrichtsgestaltung

Den Einstieg in die Unterrichtseinheit bietet das in der vorangegangenen Stunde gespielte Fangeball auf dem Schulhof. Die Schüler sollten sich überlegen, welche Parameter dieses Spiel beschreiben. Zu Beginn dieser Doppelstunde wird durch den Lehrer kurz die vereinfachte Fangeballspielversion ohne einen Fänger und zunächst ohne Wind und Luftreibung, erklärt.

In Partnerarbeit erstellen die Schüler anschließend ein Klassendiagramm, in dem das System des Spiels klar erkennbar wird. Im Anschluss hieran sollten zwei Gruppen ihre Diagramme
vorstellen und erklären. Fragen der Mitschüler sollten weitestgehend ohne Lehrerhilfe beantwortet werden.

Als nächster Schritt wird die Implementation der einzelnen durch das Diagramm vorgegebenen Klassen vorgenommen. In diesem Entwurf müßten die Klassen Umwelt, Werfer und Gegenstand implementiert werden. Eine Implementation anderer Klassen nach den von den Schülern erstellten Diagrammen ist ebenso möglich, große Abweichungen sind jedoch nicht wahrscheinlich und können schlimmstenfalls auch als ungeeignet aussortiert werden, wenn z.B. der Sinn der Objektorientierung nicht verstanden scheint.

Anhand der Klassenimplementation können die Attribute private, public und protected erklärt werden. Angesichts unserer Klassensituation rate ich hiervon jedoch ab: uns reichen private und public, diese werden dann als Eigenschaft der Attribute und Methoden erklärt, auch von anderen Klassen aufgerufen bzw. vor diesen verborgen zu werden.

Ein zu implementierendes Hauptprogramm main erstellt aus den vorhandenen Klassen die entsprechenden Objekte: Eine Umwelt, einen Werfer und zunächst zwei Gegenstände. Dies ermöglicht uns gleichzeitig die Erklärung des Unterschieds zwischen Objekten und Klassen.

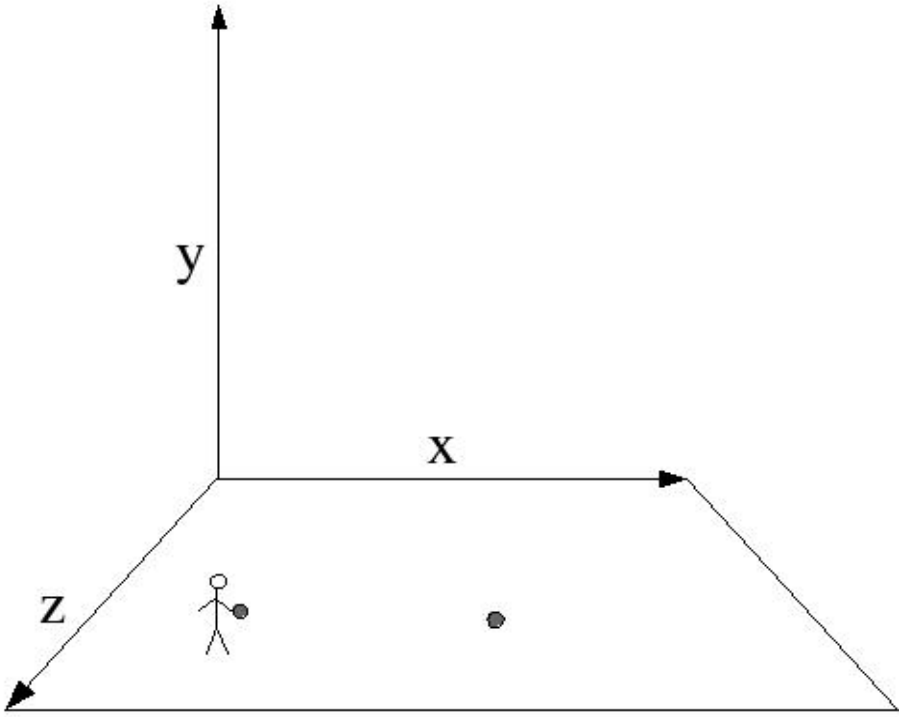
Während der Programmierarbeit, die in einem Texteditor, bevorzugt dem mc unter einer Linux-Konsole und dem javac-Compiler durchgeführt wird, können die leistungsstärkeren Schüler ihre Mitschüler unterstützen. In einem kleinen Kurs wie unserem ist dies durchaus vertretbar. Die Nutzung einer grafischen Programmierumgebung wie z.B. blueJ lehne ich hier ab, da sie meiner Ansicht nach durch unnötigen Quelltext zur grafischen Benutzeroberfläche
für Verwirrung sorgen. Spätestens bei der Programmierung der grafischen Darstellung des Wurfvorgangs treten unweigerlich Verwechselungen der einzelnen Programmfragmente auf.

Eine erste Version des Programms soll lediglich die Koordinaten der einzelnen Gegenstände
wiedergeben. Es wird durch Tastatureingabe der Werfer gesteuert: Er kann sich drehen,
vorwärts laufen, Abwurfwinkel und Abwurfkraft variieren sowie Gegenstände aufnehmen und werfen.

An dieser Stelle werden auch Abbruchbedingungen und Schleifen (sowohl if- als auch do-while-Schleifen) geklärt bzw. gefestigt.

Die Ausarbeitung einer grafischen Darstellung erfolgt in einer weiteren Unterrichtseinheit.
Hier wird der Umgang mit elementaren Grafikbefehlen geklärt. Auch dient diese Phase dazu, den Schülern eine Motivation zu geben, da hier die trockene Darstellung der Koordinaten einer interessanteren grafischen Darstellung weicht.

Bild: Grafische Darstellung des Wurfes



4.4 Tabellarischer Unterrichtsentwurf

Zeit	Phase	Inhalt	Methode	Medien
8:00 – 8:15	Problem-einführung	Die Schüler sollen im Unterrichtsgespräch klären, welche Parameter das in der letzten Stunde gespielte Fangeballspiel beschreiben könnten.	UG	Tafel
8:15 – 8:30	Klassen-diagramm	Aufgrund ihrer erarbeiteten Modelle sollen die Schüler in Zweiergruppen ein Klassendiagramm erarbeiten.	PA	
8:30 – 8:40	Vorstellung der Klassen-diagramme	Je zwei Schüler stellen ihr Klassendiagramm an der Tafel vor und erklären es. Die Auswahl der einzelnen Klassen soll durch die Schüler begründet werden. Der Lehrer kann ein Diagramm als ungeeignet ablehnen, wenn z.B. die Objektorientierung nicht oder nur knapp genutzt wird.	UG	Tafel
8:40 – 8:50	Erklärung des Arbeits- auftrags: Klassen implementieren	Der Lehrer gibt die Aufgabe, zunächst die Klassen Umwelt, Werfer und Gegenstand zu implementieren. Die Schüler arbeiten in Zweiergruppen, leistungsstarke Schüler können ihren Mitschülern helfen, sofern sie selbst fertig sind.	LV	
8:50 – 9:30	Implementation	Die Schüler programmieren die ersten drei Klassen.	PA	Computer

In der folgenden Doppelstunde wird mit der Implementation fortgefahren. Es ist damit zu rechnen, daß leistungsschwache Schüler noch ca. 10 Minuten benötigen, um eventuell aufgetretene Fehler zu beseitigen. Die Programmierung der Klasse Wurf, die die main-Routine enthält, ist, dürfte die gesamte nächste Doppelstunde einnehmen. Hier muß durch eine Schleife, die die Flughöhe überprüft der dynamische Ablauf des Wurfes realisiert werden. Die Ausgabe der Ergebnisse sowie die Eingabe der Anfangsbedingungen dürften in einige Schreibarbeit ausarten, dies läßt sich jedoch nur vermeiden, wenn der Lehrer diese Teile den Schülern abnimmt. Das könnte zur Folge haben, daß letztendlich Unklarheiten entstehen, da die Schüler diese vorgefertigten Programmfragmente nicht verstehen.

4.5 Implementation des Fangeballspiels

Die Implementation des kompletten Programms ist in der ersten Doppelstunde nicht vorgesehen. Hier sollen zunächst die drei Klassen Umwelt, Werfer und Gegenstand von den Schülern erarbeitet werden.

```
class umwelt{

private int breite;
private int laenge;
private int zeit=0;
private int g;

public set_breite(int wert){
                breite=wert;
                }

public set_laenge(int wert){
                laenge=wert;
                }

public zeitlauf(){
                inc(zeit);
                }

public set_g(int wert){
                g=wert;
                }

}

class werfer{

private int xw;
private int yw;
private int hoehe;
private int kraft;
private int maxkraft;
private int phi;
private int theta;
const schrittweite;
private gegenstand[] gegenstd;

public startparameter(int xwert,ywert; real h,max){
                xw=xwert;
                yw=ywert;
                hoehe=h;
                maxkraft=max;
                kraft=0;
                phi=0;
```

```
                theta=0;
                }

public drehen_links(){
                phi=phi+0.01;
                }

public drehen_rechts(){
                phi=phi-0.01;
                }

public wurfwinkel(real wert){
                theta=wert;
                }

public gehen(){
                x=x+cos(phi)*schrittweite;
                y=y+sin(phi)*schrittweite;
                }


}

class gegenstand{

private int xg;
private int yg;
private int zg;
private real geschw;
private real theta_g;
private real phi_g;
private real masse;
private real eta;

public startparameter(int xwert,ywert; real massewert,etawert){
                xg=xwert;
                yg=ywert;
                zg=0;
                masse=massewert;
                eta=etawert;
                }

public abwurfparameter(real geschwindigkeit, theta, phi, hoehe){
                geschw=geschwindigkeit;
                theta_g=theta;
                phi_g=phi;
                zg=hoehe
                }

}
```

4.6 Didaktische Kommentare

Es mag auf den ersten Blick sehr trocken wirken, die Schüler an Konsole und Compiler arbeiten zu lassen. Dies dient jedoch dazu, unnötigen Quelltext, der nicht verstanden wird, zu vermeiden und die Konzentration auf das Wesentliche zu lenken. Eine grafische Darstellung wird in einer späteren Unterrichtsphase erstellt, was dann auch zur Motivation der Schüler angelegt ist. Die hierfür benötigten Grafikbefehle beschränken sich auf das benötigte Minimum, um nicht ein bloßes Abschreiben beim Nachbarn zu provozieren.

Wir haben versucht, fächerübergreifende Aspekte einzubringen, was mit der Wiederholung der Beschreibung vom schiefen Wurf aus der Physik und der Darstellung dreidimensionaler
Sachverhalte in der Cavalliersperspektive aus der Mathematik auch gelungen sein dürfte.

Es soll neben dem Umgang mit der Objektorientierung auch die Implementation komplexer mathematischer Zusammenhänge am praktischen und nachvollziehbaren Beispiel trainiert werden, schließlich ist dies ein wichtiger Anwendungsaspekt der Informatik.

Die Erstellung des Klassendiagramms zu Beginn der Doppelstunde dient der Anwendung und Festigung der UML-Notation für Klassendiagramme. Auch soll diese Herangehensweise an komplexe Probleme gefestigt werden, da wir eine solche Strategie im weiteren Unterrichtsverlauf voraussetzen müssen.

5 Literaturliste

- Rahmenlehrplan - Informatik - Wahlpflichtbereich
Brandenburg Sekundarstufe I

- http://www.educeth.ch/informatik/interaktiv/javakarel

- Helmut Erlenkötter und Volker Reher: Java - HTML, Skripts, Applets und Anwendungen,
Rowohlt: Hamburg 1997, 36-71