Vorwort

Der Kurs „Pinball, Breakout & Friends“ befasste sich mit Themen der Mathematik, Physik und Informatik. Ziel des Kurses war die Programmierung eines funktionstüchtigen Computerspiels.
In zweiwöchiger Arbeit gelang es uns, dies zu erreichen. Die uns gestellten Aufgaben schweißten uns schnell zu einem starken Team zusammen, das fantastisch harmonierte und viel Spaß zusammen hatte. In unserem Kurs herrschte stets eine lockere Atmosphäre, die eine tolle Abwechslung zum sonst so langweiligen Schulalltag bot.
Auf den folgenden Seiten dokumentieren wir unsere gemeinsamen Erlebnisse, um einen Einblick in unsere Arbeit zu geben.

Unser Kurs

Unser Team:

Die Kursleiter

Daniel Jungblut
„So haben die alten Griechen ihre Pyramiden gebaut!“
Wohnort: Heidelberg
Geburtstag: 23.04.1984
Er studiert Mathematik und Informatik in Heidelberg, und ermöglichte uns den Ausflug ins IWR (Interdisziplinäres Zentrum für wissenschaftliches Rechnen). Morgens war er immer bei der Frühsport-KüA beim Joggen zu finden, wobei er sich niemals von seinen Kursteilnehmern überholen ließ. Durch sein lockeres und jugendliches Auftreten fiel es ihm leicht, sich in die Kursgemeinschaft zu integrieren. Er hat eine Gabe fürs Erklären, wenn er auch meint, keine pädagogischen Erfahrungen zu haben. Auch außerhalb der Kursthemen konnte man sich gut mit ihm unterhalten.

Jörg Richter
„Ich bin die gütige Fee“
Wohnort: Heidelberg
Geburtstag: 16.10.1971
Er ist ein sehr kompetenter Lehrer, der es verstand, die Kursteilnehmer zu begeistern. Seine Erklärungen hatten Pfiff und seine Beispiele waren (untypisch für einen Lehrer) immer sehr einleuchtend. Dank ihm und Daniel herrschte immer eine lockere und gelöste Lernatmosphäre, die uns allen Spaß machte.
Er half eifrig und mit viel Elan bei der Skat-KüA aus, was dem Leiter dieser KüA sehr beeindruckte und half. Auch hier überzeugte er durch seine unglaubliche Gabe zu erklären.

Die Kursteilnehmer

Alexander Schlüter
Wohnort: Leutenbach
Geburtstag: 10.02.1989
Der Entertainer (zusammen mit Daniel) des Kurses, der immer einen guten Spruch auf Lager hatte und alle mit seinem Humor zum Lachen brachte. Der Größte unseres Kurses! Alex war Leiter der Skat-KüA und auch ein begeisterter Skatspieler, was sich durch seine Erklärungen bemerkbar machte. Er blieb mit Daniel durch frühmorgendliches Joggen fit. In der Theater-KüA war er auch ein begeisterter Schauspieler. Man konnte sowohl viel Spaß mit ihm haben als auch prima mit ihm zusammen lernen und diskutieren.

Aline Baumeister
Wohnort: Mosbach
Geburtstag: 19.01.1989
Aline designte mit ihrer großen Kreativität die tollsten und schwierigsten Level, die uns allen großen Spaß bereiteten. Sie schaffte es, mit Tamara künstlerisch durch den Raum fliegende Objekte zu kreieren; es muss nicht unbedingt erwähnt werden, dass dies nur auf einem Tippfehler beruhte. Mit ihrer musikalischen Kenntnis versetzte sie uns in Staunen und Bewunderung. Ihre fröhliche Art war einfach mitreißend und ansteckend, was sich durch ihr unbeschwertes Lachen zeigte.

Andreas Messner
Wohnort: Trossingen
Geburtstag: 19.09.1989
Andreas war ein einfallsreicher Programmierer der Goodies, wobei die Cheats auch nicht vor ihm verschont blieben. Milan und er schafften es immer wieder aufs Neue, uns mit ihren Quelltexten zu verwirren. So war er auch immer ein Ansprechpartner bei Problemen, um die er sich oft sehr zeitintensiv und mit Hingabe kümmerte. Mit Humor und viel Engagement ging er ans Werk.

Larissa Blankenburg
Wohnort: Neckargerach
Geburtstag: 22.09.1989
Zusammen mit Louisa entwickelte sie gute Ideen und brachte „Leben“ in unser Spiel.
Auch wenn sie meistens eher ruhig war, fanden wir ihre Beiträge immer produktiv und hilfreich. Wir schätzten alle Larissas liebe und warme Art.

Louisa Adolph
Wohnort: Karlsruhe
Geburtstag: 14.01.1991
Larissa und sie waren ein unzertrennliches Team beim Entwickeln von Ideen und deren Verwirklichung. Außerdem war sie ein leidenschaftliches Mitglied der Theater-KüA.
Ihr nettes Verhalten den anderen gegenüber machte sie uns sehr sympathisch.

Markus Feifel
Wohnort: Murrhardt
Geburtstag: 01.09.1989
Markus war ein begeisterter Entwickler und Tester der Cheats, wobei diese natürlich nur zum Testen des Spiels verwendet wurden. Er war eines der Akademie-Geburtstagskinder, was wir auch ausgiebig feierten. In der Freizeit war er meist an der Tischtennisplatte vorzufinden, wo er immer wieder für Spaß sorgte.

Martin Spitz
Wohnort: Bruchsal
Geburtstag: 26.08.1989
Der „Star“ des Kurses: ein Bild von ihm steht an erster Stelle bei der Google-Suche.
Bei der Arbeit kam er durch seine kreativen Ideen zum Vorschein, obwohl er eher ein ruhiger Typ ist. Alle im Kurs kamen gut mit ihm aus.

Milan Holzäpfel
Wohnort: Sersheim
Geburtstag: 12.10.1988
Milan war genauso ein Freak wie Andreas. Ob er bei uns im Kurs noch viel dazulernte, ist fraglich, da er schon ziemlich viele Vorkenntnisse hatte.
Aufgefallen ist er durch seine etwas außergewöhnliche Tastatur, die er sich mitbrachte (auf „normalen“ kann er nicht so gut und schnell schreiben). Durch sein aufgeschlossenes Verhalten machte er sich bei uns sehr beliebt.

Susanne Szkola
Wohnort: Tannheim
Geburtstag: 21.09.1989
Sie ist unsere kreative Photoshop-begeisterte Designerin verschiedenster Grafiken. So hatte sie auch ein ausgesprochenes Talent beim Erstellen der Plakate. Mit ihr konnte jeder gut umgehen und ihr sympathisches Auftreten lockerte die Atmosphäre. Am Klavier ist sie ein Ass, was sie uns am Konzertabend eindrücklich bewies.

Tamara Buck
Wohnort: Albstadt - Tailfingen
Geburtstag: 23.08.1989
Gemeinsam mit Susi entwickelte auch sie einige spannende Level. Tamara lacht sehr gerne, was man im Kurs auch des Öfteren bemerkte. So verbreitete sie während der Arbeit stets eine gute Stimmung, die uns alle mitriss. Auch war sie eine talentierte Schauspielerin in der Theater KüA, wo sie ihrer Kreativität freien Lauf ließ („Es hat mein Nashorn zertrampelt“).

Vorkurs

Da einige der Teilnehmer noch keine Programmierkenntnisse besaßen, wurden wir zuerst während des Vorbereitungswochenende in die Programmiersprache C++ eingewiesen. Diese Programmiersprache ist die neuere Version von C. Sie erlaubt es, objektorientiert zu programmieren. Für die Grafiken benutzten wir OpenGL, eine Grafikbibliothek, die Befehle zum Zeichnen eines Objektes kennt.
Während dieser Einführung schrieben wir erste Programme. Bei manchen konnte man mehrere Zahlen eingeben, die der Computer addierte, subtrahierte, multiplizierte oder dividierte. Bei einem anderen Programm gab man zwei Zahlen ein, und der Computer teilte einem mit, welche die größere Zahl ist. Unser erstes Programm mit den OpenGL-Befehlen zeichnete eine farbige Kugel auf den Bildschirm.
Nach dem Vorbereitungswochenende ging der Kurs per E-Mail weiter. Durch diese Übungen lernten wir, wie man noch andere grafische Objekte programmiert oder wie man schwierigere Rechenprogramme schreiben kann.

Einige dieser Aufgaben:

1. Programmiere ein Programm, das zuerst deinen Geburtstag, dann das aktuelle Datum abfragt und anschließend dein Alter in Tagen ausgibt.
2. Schreibe eine Funktion, die überprüft, ob eine natürliche Zahl eine Primzahl ist. Startet man das Programm, so kann man eine Zahl eingeben. Man erfährt dann, ob es sich um eine Primzahl handelt. Ist die eingegebene Zahl eine Null, so beendet sich das Programm.
3. Eine Kugel soll sich auf einem rechteckigen Tisch bewegen und an den Wänden reflektiert werden.

Farben

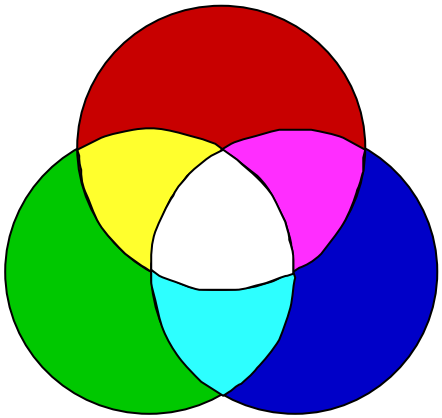
```
glColor3f(0.0, 1.0, 0.0);
```

Mit diesem Befehl gibt man die Farbe des Objektes, das man als nächstes zeichnen will, an – in diesem Fall grün.
3f deshalb, da man zu der Farbwahl 3 Variablen des Typs **f**loat angeben muss.
Die Intensität der einzelnen Farben- oder auch Zwischentöne/Mischfarben wird durch verschieden hohe Werte der drei Parameter erreicht, das heißt,

jede Variable kann einen Wert zwischen 0 und 1 annehmen – je nachdem, wie groß dieser Wert ist, desto kräftiger ist der Farbton.
Der erste Wert entspricht dem **R**ot, der zweite dem **G**rün und der dritte dem **B**lau.
Deshalb heißt dieses „Farbmischsystem“ **RGB**-System – man spricht auch von den drei Grundfarben. Dass man alle Farben mit drei Grundfarben darstellen kann, liegt an der Funktion unserer Augen.

Sehen von Farben mit dem Auge

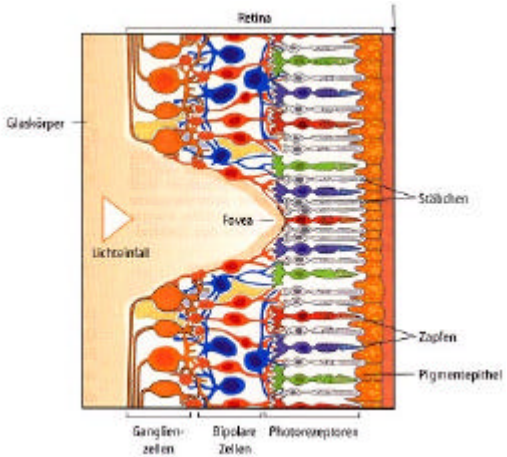
Weißes Licht enthält alle Farben. Wenn dieses Licht auf einen Gegenstand fällt, wird ein Teil davon verschluckt (absorbiert) und ein Teil wird zurückgeworfen (reflektiert). Die Farbanteile, die reflektiert werden, legen die Farbe des Gegenstandes fest. Dieses Licht gelangt in das Auge. Durch die Hornhaut und die Linse wird das Licht im Auge gebündelt. Die Pupille kann durch ihre Größe die Menge des Lichtes, das ins Auge gelangt, regeln. Die Linse kann das Bild „scharfstellen" indem sie ihre Form mit Hilfe der Muskeln verändert. Das Bild wird schließlich auf der Netzhaut abgebildet. Das kann man mit dem Ablichten eines Bildes in der Fotokamera vergleichen.
Auf der Netzhaut befinden sich drei verschiedene Sorten von farbempfindlichen Strukturen (Farbrezeptoren), die sogenannten Zapfen. Jeder dieser Zapfen reagiert empfindlich auf bestimmte Farbbereiche. Bei den Farbbereichen handelt sich um die sogenannten Primärfarben des Auges: Rot, Grün und Blau. Aus diesen Farben kann das Auge jede weitere beliebige Farbe mischen.



Mischt man Rot und Grün, entsteht Gelb.
Mischt man Rot und Blau, entsteht Magenta.
Mischt man Blau und Grün, entsteht Cyan.

Die Zapfen in der Netzhaut werden von den in einer Farbe enthaltenden Farbbestandteilen angeregt. Sie geben dann einen elektrischen Impuls an das Gehirn weiter, falls ein Bestandteil der Farbe, auf die sie empfindlich reagieren, vorhanden ist. Je nachdem, wie stark dieser Impuls eines Farbrezeptors ausfällt, erkennt das Gehirn, wie groß der Bestandteil der Farbe, für die der Farbrezeptor zuständig ist, in dem Licht ist. Indem das Gehirn alle Farbanteile in der richtigen Proportionalität mischt, erkennt es die Farbe des Gegenstandes.

Aufbau der Netzhaut

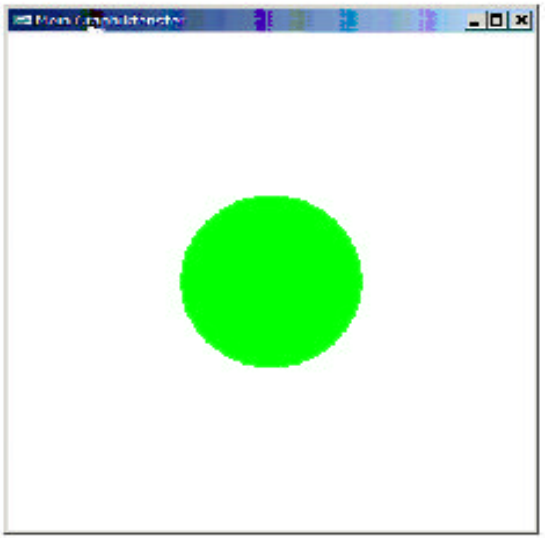


Beispiel:

Ist ein Gegenstand gelb, wird durch das ins Auge einfallende Licht der rotempfindliche Zapfen und der grünempfindliche Zapfen angeregt. Handelt es sich mehr um ein Orange, wird der rotempfindliche Zapfen mehr und der grünempfindliche Zapfen weniger stark angeregt.

Bilder

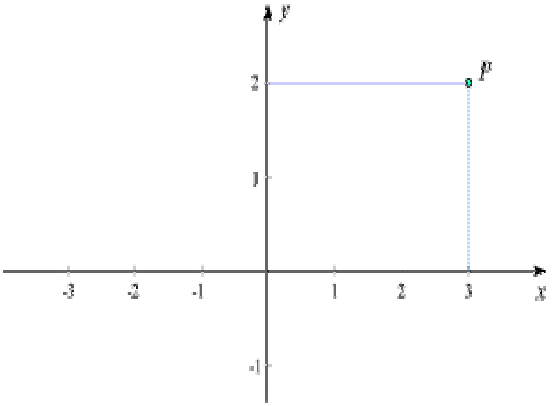
In diesem Abschnitt wird erklärt, wie man mit Hilfe von OpenGL Bilder zeichnen kann. Als erstes muss betrachtet werden, wie man die Position des Objekts in diesem Fenster angeben kann. Dies wird durch ein kartesisches Koordinaten-system erreicht.



Unsere Interpretation des Grünen Punktes

Zur Erinnerung: dieses Koordinatensystem besitzt zwei senkrecht zueinander stehende Achsen, die x-Achse und die y- Achse.
Die x-Achse verläuft von links nach rechts und die y-Achse von unten nach oben (beide sind unendlich lang).

Durch die Koordinaten kann man nun jeden beliebigen Punkt in diesem Fenster genau angeben. Dies erreicht man, indem man jedem Punkt einen x-und einen y-Wert zuweist.
In unserem Fall würde man in der mathematischen Schreibweise von dem Punkt P (3|2) sprechen, wobei 3 dem Zahlenwert auf der x-Achse und 2 dem Wert auf der y-Achse entspricht.



Nun können wir also Punkte im Koordinatensystem genau angeben.

Zeichnen einer Kugel

Um eine Kugel zu zeichnen, müssen wir ihren Mittelpunkt angeben (natürlich mit x-und y-Wert) und ihren Radius.
Hierzu benutzt man Variablen, um, wie der Name schon sagt, die Werte schneller und einfacher ändern zu können, ohne den ganzen Text verändern zu müssen.
Schauen wir uns die Syntax in C++ einmal näher an:

```
float x = 50;
float y = 80;
int Radius = 5;

void display()
{
glClear(GL_COLOR_BUFFER_BIT);

glColor3f(0.0, 1.0, 0.0);
grafikKugel(x, y, Radius);


glFlush();
glutSwapBuffers();
}
```

float bezeichnet eine neue Variable, die eine Gleitkommazahl sein soll. Diese Variablen heißen hier x und y (um den x- und y-Wert anzugeben).
Nun gibt es in diesem Beispiel noch einen anderen Variablentyp:
int (integer - ganzzahlig)
Mit diesem Typ wird hier die Variable „Radius“ versehen, die, wie der Name schon sagt, den Radius der Kugel bestimmt.
Nun folgt die *display-Funktion:*
Diese Funktion beinhaltet die Befehle, die für das Zeichnen der Objekte verantwortlich sind.
Die verschiedenen Befehle bedeuten:

`glClear(GL_COLOR_BUFFER_BIT);`
Löscht am Anfang den ganzen Grafikfensterbereich, so dass die neuen Objekte gezeichnet werden können.

`grafikKugel(x, y, Radius);`
grafikKugel ist in unserem Fall der Befehl, um die Kugel zu zeichnen. Dieser Befehl benötigt drei Parameter, um, wie oben schon beschrieben, den Mittelpunkt und den Radius anzugeben.

Die Reihenfolge dieser Werte spielt hier eine große Rolle: als erstes gibt man den x-Wert an, als zweites den y-Wert. Als letzte Angabe folgt der Radius. Vertauscht man diese Werte, entstehen zwar auch Kreise, aber nicht an der beabsichtigten Stelle und die Größe ist auch nicht mehr die gewünschte.

```
glFlush();
glutSwapBuffers();
```
sind spezielle Befehle an die Grafikkarte, die später genauer erläutert werden.

Zeichnen eines Rechtecks

Bei einem Rechteck muss man nur die vier Eckpunkte im Koordinatensystem angeben, um es zu zeichnen.
Die Syntax hierzu lautet:

```
void display()
{
glClear(GL_COLOR_BUFFER_BIT);

glColor3f(0.0, 0.0, 1.0);

glBegin(GL_QUADS);
      glVertex2f(120, 50);
      glVertex2f(170, 50);
      glVertex2f(170, 100);
      glVertex2f(120, 100);
glEnd();

glFlush();
glutSwapBuffers();
}
```
Dieses Beispiel zeichnet ein blaues Rechteck.
Um die Farbverläufe im Objekt zu erreichen, gibt man anstatt der Farbe für das gesamte Rechteck je eine andere Farbe für jede Linie an.

```
glBegin(GL_QUADS);
```

bedeutet, dass angefangen wird, ein neues Rechteck zu zeichnen. Die Beschreibung in der Klammer erklärt, um was es sich dabei handelt - ein Rechteck.

```
      glVertex2f(120,50);
      glVertex2f(170,50);
      glVertex2f(170,100);
      glVertex2f(120,100);
```

Der `glVertex`-Befehl zeichnet die Ecken des Rechtecks.

```
glEnd();
```
zeigt an, dass der Befehl für das Zeichnen eines Rechtecks zu Ende ist.

Man kann mit diesem Befehl (ein wenig abgewandelt) auch **Geraden** zeichnen, in dem man statt `GL_QUADS` **`GL_LINES`** benutzt. Wie der Name schon sagt zeichnet man nun **Linien.** Allerdings

muss man dann nur Start- und Endpunkt dieser Geraden angeben.

Nun würde man aber von den gezeichneten Kugeln und Rechtecken noch nichts sehen, da sie zwar in der Displayfunktion gezeichnet worden sind, aber diese Grafik noch nirgends aufgerufen worden ist.
Dies erreicht man, in dem man in der *main-Funktion* die Grafikroutine startet:

```
int main(int argc, char *argv[])
{

grafikInit(argc, argv, "Breakout",
            300, 300, 1.5, display);

glutMainLoop();
return 0;
}
```

```
grafikInit(argc, argv, "Breakout",
            300, 300, 1.5, display);
```

initialisiert ein Fenster, in dem dann die Objekte gezeichnet werden.
Der Befehl hat einige Argumente:
argc und *argv* sind Standartwerte und werden immer verwendet.
Danach folgt der Name des Fenster, der frei wählbar ist. Nun wird die Größe des Fensters angegeben. Die Werte können beliebig groß vergeben werden. Als erstes gibt man die Breite an, danach die Höhe.

Nun kann man einen Zoomfaktor eingeben, der bestimmt, um wie viel vergrößert das Fenster dargestellt werden soll.
Die letzte Angabe ist die Displayfunktion, also diejenige Funktion, die zum Zeichnen eines Bildes aufgerufen werden soll.

```
glutMainLoop();
```

führt die Befehle der Grafikroutine aus und überprüft, ob noch andere Funktionen neben der Displayfunktion ausgeführt werden müssen oder vorhanden sind.
Sie reagiert auch auf Tastatureingaben.

```
return 0;
```

Beendet das Programm.

Bewegte Bilder

Wir können nun also Bilder auf dem Monitor darstellen lassen. Für ein bewegtes Computerspiel gilt es nun, den noch statischen Bildern Leben einzuhauchen.

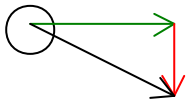
idle- Funktion

Für die Bewegung verwenden wir in unserem Programm ein Zusammenspiel der *display-Funktion* und der sogenannten *idle-Funktion*.
Diese Funktion wird in einem Programm immer dann aufgerufen, wenn der Computer nicht rechnen muss.

Sie enthält Rechenschritte, die das Aussehen des neuen Bildes bestimmen. Neben etlichen Rechnungen enthält die Funktion den Befehl `GlutPostRedisplay()`, welche dem Computer den Befehl gibt, dass Bild bei nächster Gelegenheit neu aufzubauen.
Immer wenn ein neues Bild gezeichnet wurde, wird die *idle-Funktion* aufgerufen, die das neue Bild berechnet. Anschließend wird das Bild von Glut wieder neu gezeichnet. Ist dies abgeschlossen wird wieder die *idle-Funktion* aufgerufen, usw.

Ein Beispiel:
Ein Grafikprogramm soll eine Kugel darstellen, die sich bewegt. Die Kugel wird wie gewohnt in der *display- Funktion* gezeichnet. Die x- und y- Werte der Kugel werden allerdings als Variablen *x* und *y* deklariert. Den Variablen werden bestimmte Werte zugewiesen, diese entsprechen den x- und y- Werten des Startpunkts der Kugel.
In der der *idle-Funktion* wird nun dem x- und/oder y- Wert der Kugel ein Wert hinzuaddiert
(z.B. x = x + 1).
Wenn nun die Kugel gezeichnet ist, wird die *idle-Funktion* aufgerufen, die im obigen Fall dem alten x- Wert 1 hinzuzählt. Man erhält einen neuen x- Wert.
Durch den Befehl `GlutPostRedisplay()` wird nun wieder die *display-Funktion* aufgerufen, welche das Bild neu zeichnet. Die Kugel wurde um den Wert 1 (entspricht einer Einheit auf dem Koordinatensystem) nach rechts (auf der x-Achse) verschoben.

Selbiges kann man selbstverständlich auch mit einer Bewegung in y-Richtung machen; ebenso, wie man die beiden Geschwindigkeiten x und y gleichzeitig verändern kann. Hierbei gibt man in der *idle-Funktion* an, dass beispielsweise dem Kugelmittelpunkt in x-Richtung 1 und in y-Richtung 2 hinzuaddiert wird. Die Kugel würde sich nach rechts oben bewegen.
Somit definiert man für die Kugel zwei Geschwindigkeiten: eine in x- Richtung, und eine in y- Richtung. Sie stellen die Bewegungen nach rechts bzw. oben dar. Möchte man eine Bewegung nach links oder unten haben, so muss man von den x- bzw. y-Werten etwas abziehen. Die Kugel hat also zwei Geschwindigkeitskomponenten, die jeweils positiv oder auch negativ sein können. Somit kann man der Kugel (und natürlich auch jedem anderen beliebigen Objekt) jede mögliche Richtung und Geschwindigkeit geben.



Obige Zeichnung stellt die beiden Geschwindigkeiten in x Richtung (grün) und y Richtung (rot) und die daraus resultierende Gesamtgeschwindigkeit dar.

„Flimmernder Bildschirm“

Bei den Bewegungen der Objekte stößt man über kurz oder lang auf ein Problem:
Es müssen immer wieder neue Bilder gezeichnet werden. Ein neues Bild kann jedoch nur gezeichnet

werden, wenn das vorherige Bild komplett gelöscht wurde. Dies geschieht mit dem Befehl `glClear`.
Der Computer zeichnet also ein Bild, löscht es, rechnet in der *idle-Funktion* das neue Bild aus und zeichnet erst dann das neue Bild.
Dies führt zu einem Flimmern, da zwischen den einzelnen Bildern schwarze Lücken bleiben und man das Zeichnen des Bildes „sehen“ kann. Je nachdem, wie schnell der Computer das neue Bild berechnet, bzw. wie oft das Bild in einer Sekunde gezeichnet wird, ist das Flimmern mehr oder weniger stark (hohe oder niedrige Frequenz).
Dies kann einem schnell den Spaß am Spielen rauben.
Deshalb verwenden wir in unserem Programm zwei Grafikpuffer. Diese befinden sich als Speicher auf der Grafikkarte und werden auch in kommerziellen Spielen verwendet, um das Flimmern zu verhindern.
Im Prinzip funktionieren die Puffer folgendermaßen:
Ein Puffer (Speicher) zeigt das Bild an, während im Hintergrund der andere gelöscht und neu beschrieben wird. Ist das Zeichnen des Bildes auf dem zweiten Puffer abgeschlossen, werden die Puffer ausgetauscht: Puffer zwei zeigt das Bild an, während der Puffer, der eben noch das Bild anzeigte, im Hintergrund gelöscht und neu beschrieben wird. Danach werden die Puffer abermals ausgetauscht, usw.: Es entsteht eine flimmerfreie Bewegung.

Geschwindigkeit?!
Nach dem Beseitigen des Flimmerns hat man nun eine Kugel, die sich auf dem Bildschirm flimmerfrei in eine bestimmte Richtung bewegt. Die Geschwindigkeit der Kugel wird bestimmt, indem man bei jedem Rechenschritt zu den Koordinaten der Kugel einen festen Wert addiert. Testet man das Spiel nun jedoch auf einem anderen Computer, so stellt man fest, dass sich die Kugel im Vergleich zum vorherigen System nicht gleich schnell bewegt.
Dies hat folgende Ursache:
Je nachdem, wie schnell ein Computer rechnet, wird die *idle-Funktion* unterschiedlich häufig pro Sekunde durchlaufen; bei einem schnellen Rechner häufiger als bei einem langsamen. Somit bewegt sich auch die Kugel auf den verschiedenen Systemen, je nach Leistung, unterschiedlich schnell.
Dies sollte jedoch nicht der Fall sein, da man sich bei einem Computerspiel ja auch objektiv messen will. Spieler mit besseren Computern hätten es deutlich schwerer.
Bei genauerem Betrachten des Programms fällt auf, dass die Kugel gar keine „richtige“ Geschwindigkeit besitzt. Physikalisch gesehen ist die Geschwindigkeit (v), eine zurückgelegte Strecke (?s) pro Zeitabschnitt (?t). Aus der Physik ist die Formel

$$v = \frac{\Delta s}{\Delta t}$$

bekannt.
Die Zeit wird im Programm bis jetzt jedoch noch nicht berücksichtigt. Um also eine exakte, auf allen Computern gleich bleibende Geschwindigkeit zu erlangen, müssen wir banal gesprochen „wissen, wie viel Uhr es ist“.
Hierzu verwenden wir in unserem Programm die Systemzeit. Diese ist bei MS Windows auf die

Millisekunde genau und wird über den Befehl `GetTickCount()` aufgerufen.
Da wir zum Zeichnen des neuen Bildes nicht die Zeit oder die Geschwindigkeit wissen müssen, sondern die Strecke, die die Kugel zurücklegt, müssen wir die obige Gleichung

$$v = \frac{\Delta s}{\Delta t} \quad \text{zu} \quad \Delta s = v \cdot \Delta t$$

umformen.
Wir müssen also eine angemessene Geschwindigkeit definieren. Mit dieser und der Zeit kann der Computer die zurückgelegte Strecke der Kugel bestimmen.
Ein weiterer Vorteil: Rechnet der Computer (wegen laufender Hintergrundanwendungen) nicht immer gleich schnell, so bewegt sich die Kugel trotzdem mit gleich bleibender und konstanter Geschwindigkeit.

Der Computer fragt also die Systemzeit zu Beginn des *„idle-Funktion“*-Durchlaufs ab, und speichert diese als Variable ab. Beim nächsten Durchlauf zieht er diese Zeit von der aktuellen Zeit ab; diese Differenz gibt an, wie viel Zeit seit dem letzten Durchlauf der Funktion vergangen ist. Sie wird nun mit der vorher definierten Geschwindigkeit multipliziert: Man erhält die Strecke, die die Kugel zurückgelegt hat. Diese wird nun den alten x- und y-Werten hinzuaddiert. Anschließend wird das Bild neu gezeichnet.

Tastaturabfrage

Um aktiv in das Computerspiel eingreifen zu können, haben wir eine Tastaturabfrage programmiert, die dem Computer sagt, welche Tasten vom Spieler gedrückt sind, und was diese bewirken.
Hierzu wird ein Array `bool *Tastatur` deklariert, in dem für jede Taste gespeichert ist, ob sie gedrückt ist oder nicht. Es ist ein boolsches Feld, da die Tasten der Tastatur entweder gedrückt sein können oder nicht gedrückt sein können.
Eine Variable des Types *bool* kann nur zwei Werte annehmen: *true* oder *false* (wahr oder falsch). Alternativ können *true* und *false* auch durch die Ziffern 1 für *true* und 0 für *false* ersetzt werden.
Ob eine bestimmte Taste gedrückt ist oder nicht, wird durch eine if- Abfrage geklärt. Diese beginnt beispielsweise mit `if(Tastatur['a'])…` . Wenn also die Taste ’a’ gedrückt ist, dann führt der Computer eine bestimmte Aktion aus. Drückt man nun eine bestimmte Taste, so werden bestimmte Funktionen aufgerufen, die zum Beispiel das Spielpaddle bewegen, die Pausefunktion aktivieren, usw.

Reflexion, aber wann?

Da unser Ziel darin bestand, ein Breakoutspiel zu programmieren, ist es notwendig auch eine Kollisionsabfrage zu schreiben, die die Kugel an den verschiedenen Banden oder am Paddle abprallen lässt. Hierbei ist selbstverständlich zu beachten, dass diese Reflexion den Gesetzen der

Physik folgen und nicht einfach willkürlich ablaufen soll.
Der erste Schritt hierbei liegt darin, dem Computer zu signalisieren, wann die Kugel reflektiert werden muss und wann nicht. Hierzu verwendet man eine if-Abfrage, bei der festgestellt wird, ob die Kugel eine Bande berührt. In Worten ausgedrückt könnte man diese Abfrage folgendermaßen beschreiben: „**Wenn** die Kugel eine Bande berührt, **dann** muss eine Reflexion durchgeführt werden."
Dies lässt sich mit den bereits vorhandenen und eingeführten Variablen für die Position der Kugel (x- und y- Wert) durchführen. Dabei muss beachtet werden, dass sich die Position der Kugel durch deren Mittelpunkt definiert, die Kollision allerdings schon ausgeführt werden soll, wenn der Rand der Kugel eine Bande oder das Paddle berührt. Also muss der Radius mit in die Rechnung einbezogen werden. Ebenso verhält es sich mit den Blöcken, deren Position durch den Mittelpunkt, also den Schnittpunkt der Diagonalen definiert wird. Die Ränder der Banden und Blöcke lassen sich berechnen, indem man zu den Koordinaten des Mittelpunktes die halbe Höhe bzw. die halbe Breite hinzuaddiert.

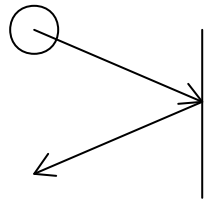
Wenn also der Kugelmittelpunkt plus Radius der Kugel gleich dem Mittelpunkt eines Blockes plus halber Höhe/Breite entspricht, dann wird die Kollision ausgeführt.

Dieser Befehl klingt kompliziert, ist allerdings programmiertechnisch leicht umzusetzen.

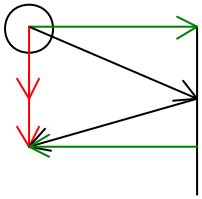
Reflexion, aber wie?

Wenn nun also die obige Bedingung erfüllt ist, dann muss die Kugel reflektiert werden. Im folgenden Abschnitt wird nur die Reflexion einer Kugel an senkrechten oder waagrechten Wänden erläutert. Die Kollision an schräg verlaufenden Wänden wird später beschrieben.

Wenn eine Kugel von links oben nach rechts unten fliegt (siehe Bild), dann wird sie nach links unten reflektiert. Hierbei gilt das Reflexionsgesetz (Einfallswinkel gleich Ausfallswinkel).



Die Geschwindigkeit der Kugel bzw. der Betrag der Geschwindigkeit bleibt gleich, nur die Richtung ändert sich.
Betrachtet man jetzt nur die y-Geschwindigkeit vor und nach der Reflexion so stellt man fest, dass sie sich nicht verändert hat: Die Kugel fliegt immer noch mit der selben Geschwindigkeit nach unten, nur anstatt sich nach rechts zu bewegen, fliegt sie nun nach links.



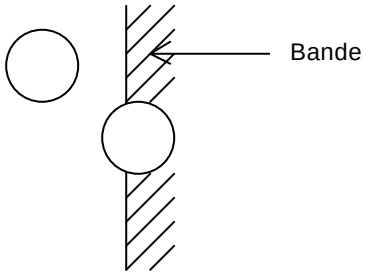
Die y- Geschwindigkeit bleibt gleich (rot), die x-Geschwindigkeit wird umgedreht

Das bedeutet, dass sich das Vorzeichen der x-Geschwindigkeit ändert.
Hatte die Kugel vorher eine positive x-Geschwindigkeit (siehe Bild), dann hat sie nach der Reflexion eine negative, und umgekehrt.
Mathematisch gesehen wird also nur ein Minus vor die Geschwindigkeit gesetzt; die Kugel wird Reflektiert.
Trifft die Kugel auf eine waagerechte Bande, so ändert sich die x-Geschwindigkeit nicht, und das Vorzeichen der y-Geschwindigkeit ändert sich.

Reflexion, oder doch nicht?

Nachdem die Idee der Reflexion in das Programm integriert wurde, liefen die Tests weniger befriedigend. Die Kugel wurde, trotz integrierter Kollision, schlicht und einfach nicht an der Bande zurückgeworfen, sondern flog durch diese hindurch.
Bei der Untersuchung des Problems fiel uns auf, dass die Kollision nur ausgeführt wird, wenn die Koordinaten der x- und y-Werte der Kugel und der Bande exakt übereinstimmen. Es ist jedoch sehr unwahrscheinlich, dass dies der Fall ist.

Kurz gesagt: In einem Bild ist die Kugel noch vor einer Bande und im nächsten ist sie schon ein kleines Stück in der Bande. Die Koordinaten stimmen aber nicht überein; die Kollisionsabfrage wird umgangen.



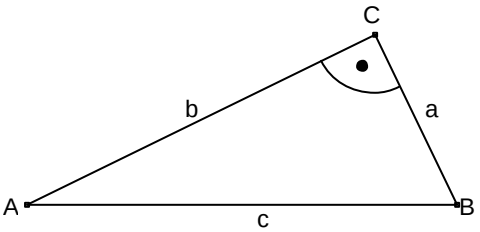
Dieses Problem lässt sich jedoch leicht lösen, indem man die Kugel nicht nur reflektieren lässt, wenn sie die Bande berührt, sondern auch, wenn sie schon ein kleines Stück in ihr „drin steckt“.
Programmiertechnisch lässt sich dies mit dem, aus der Mathematik bekannten, Größergleich (>=) verwirklichen.
Eine Kollision an einer Bande wird nun also nicht nur durchgeführt, wenn die x- bzw. y-Werte der Kugel gleich der Bande sind, sondern wenn sie größergleich bzw. kleinergleich sind.

Fliegt eine Kugel also von links nach rechts und fliegt „in“ eine Bande hinein, so ist der x-Wert ihres Randes größer, als die der Bande: Die Kollision wird durchgeführt.
Nach Verbesserung der Funktion konnten wir uns endlich über eine funktionierende Kollision freuen.

Die Mathematik zum Spiel

Natürlich brauchen wir für unser Spiel auch ein wenig Mathematik. Um es zu perfektionieren, müssen wir uns mit Vektoren auseinandersetzen.

Satz des Pythagoras



a und b: Kathete
c: Hypotenuse

Der Satz des Pythagoras dient zur Längenberechnung im rechtwinkligen Dreieck und wird im Folgenden des Öfteren verwendet.
Mit ihm lässt sich aus den Längen zweier Seiten die Länge der dritten Seite ausrechen.

$$a^2 + b^2 = c^2 \quad \Leftrightarrow \quad c = \sqrt{a^2 + b^2}$$
$$c^2 - b^2 = a^2 \quad \Leftrightarrow \quad a = \sqrt{c^2 - b^2}$$
$$c^2 - a^2 = b^2 \quad \Leftrightarrow \quad b = \sqrt{c^2 - a^2}$$
$$(a > 0,\ b > 0,\ c > 0)$$

Umgekehrt gilt:: Gilt diese Längenbeziehung in einem Dreieck, dann handelt es sich um ein rechtwinkliges Dreieck.

Vektoren

Skalare sind reelle Zahlen mit einer Dimension; 10 ist z. B. ein Skalar. Wenn man aber ein Objekt, das durch mehr als einen Skalar definiert wird angeben möchte, dann sind Vektoren sehr praktisch. Ein Vektor ist definiert als "Reihe" von beliebig vielen Skalaren.
Einen Vektor, der aus 2 Skalaren besteht, notiert man folgendermaßen:

$$\vec{v} = \begin{pmatrix} 3 \\ -4 \end{pmatrix}$$

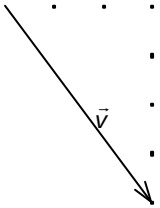
Um den Vektor von einem Skalar unterscheiden zu können, notiert man ihn mit einem Pfeil über dem Buchstaben.
Dabei sagt man, dass der Vektor die Dimension 2 hat. Seine 1. Komponente wird durch die 3 und seine zweite Komponente durch die -4 repräsentiert.
Allgemeine Schreibweise mit beliebig vielen Dimensionen:

$$\vec{v} = \begin{pmatrix} v_1 \\ v_2 \\ ... \\ v_n \end{pmatrix}$$

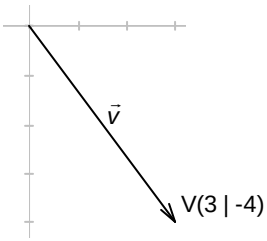
Vektoren als Pfeile

Vektoren finden in der Mathematik und der Physik viele Verwendungszwecke; darunter ist auch unserer: Man kann Vektoren als Repräsentation von Pfeilen verwenden. Bei unserem obigen zweidimensionalen Vektor würde es sich dann um einen Pfeil handeln, der 3 Längeneinheiten in die

x-Richtung und -4 Längeneinheiten in die y-Richtung verläuft:



Nun kann man Vektoren auch zur Angabe von Punkten benutzen. Dabei wird der Punkt einfach durch einen Pfeil bzw. Vektor vom Ursprung zum Ort des Punktes angegeben:



$$\vec{v} = \begin{pmatrix} 3 \\ -4 \end{pmatrix}$$

Länge/Betrag eines Vektors

Für jeden Vektor ist ein Betrag definiert:

$$|\vec{v}| = \sqrt{v_1^2 + v_2^2 + ... + v_n^2}$$

In unserem Fall, in dem der Vektor als Pfeil verwendet wird, entspricht der Betrag des Vektors nach Pythagoras der eigentlichen Länge des Pfeils.

Beispiel:

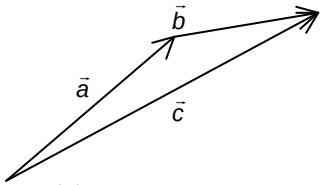
$$\vec{v} = \begin{pmatrix} 3 \\ -4 \end{pmatrix}$$

$$|\vec{v}| = \sqrt{3^2 + (-4)^2} = \sqrt{25} = 5$$

Rechnen mit Vektoren

Addition

Die Addition zweier Vektoren, $\vec{a} + \vec{b}$ wird als Aneinanderreihung der beiden Pfeile betrachtet:



$$\vec{a} = \begin{pmatrix} 1 \\ 3 \end{pmatrix}$$

$$\vec{b} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$$

Die Berechnung erfolgt einfach durch Addieren der jeweiligen Komponenten:

$$\vec{c} = \vec{a} + \vec{b} = \begin{pmatrix} 1 \\ 3 \end{pmatrix} + \begin{pmatrix} 2 \\ 1 \end{pmatrix} = \begin{pmatrix} 3 \\ 4 \end{pmatrix}$$

Allgemein:

$$\vec{v} + \vec{u} = \begin{pmatrix} v_1 \\ v_2 \\ ... \\ v_n \end{pmatrix} + \begin{pmatrix} u_1 \\ u_2 \\ ... \\ u_n \end{pmatrix} = \begin{pmatrix} v_1 + u_1 \\ v_2 + u_2 \\ ... \\ v_n + u_n \end{pmatrix}$$

Skalare Multiplikation

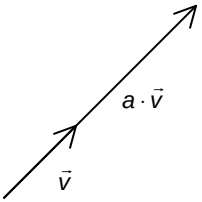
Bei der skalaren Multiplikation wird ein Vektor mit dem Faktor *a* gestreckt. Falls *a* negativ ist, wird die Richtung des Vektors umgekehrt.

$a \cdot \vec{v}$ heißt skalare Multiplikation

a : Zahl (Skalar)

$\vec{v}$: Vektor

$$a \cdot \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} a \cdot v_1 \\ a \cdot v_2 \end{pmatrix}$$



Parameterdarstellung von Geraden

Bis jetzt haben wir Vektoren verwendet, um Pfeile und – über Ursprungspfeile – Punkte zu definieren. Nun wollen wir Vektoren verwenden, um Geraden zu beschreiben.

Dies lässt sich folgendermaßen bewerkstelligen: Zuerst gibt man einen beliebigen Punkt der Geraden mit einem Ursprungsvektor an. Dieser Vektor heißt Stützvektor. Um nun alle Punkte der Gerade erreichen zu können, gibt man zunächst mit einem weiteren Vektor die Richtung der Geraden an - dieser Vektor wird Richtungsvektor genannt. Wenn man nun den Richtungsvektor mit der skalaren Multiplikation verlängert (was auch die Umkehrung in die Gegenrichtung einschließt), so erreicht man jeden beliebigen Punkt der Geraden.

Als Formel heißt das:

$g: \vec{x} = \vec{v} + r \cdot \vec{u} \qquad r \in \mathrm{R}$

- $\vec{x}$ sei ein beliebiger Punkt der Geraden, den wir erreichen wollen.
- $\vec{v}$ sei der Stützvektor, der auf einen beliebigen Punkt der Geraden zeigt.
- $\vec{u}$ sei der Richtungsvektor der Geraden, der mit
- $r \in \mathrm{R}$ als Parameter beliebig verlängert oder verkürzt und/oder umgekehrt werden kann, um jeden Punkt der Geraden zu erreichen.

Veranschaulichung

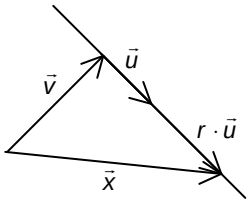
$\vec{x}$: Punkt auf der Geraden

$\vec{v}$: Stützvektor

r : Parameter

$\vec{u}$: Richtungsvektor

$\vec{x} = \vec{v} + r \cdot \vec{u}$



Anwendungsbeispiel

Über die gerade beschriebene Parameter-darstellung einer Geraden kann man jeden beliebigen Punkt einer Geraden errechnen. Also kann man damit auch überprüfen, ob ein gegebener

Punkt, wie z. B. P(7|10) auf einer beschriebenen Geraden liegt.

$$\vec{v} = \begin{pmatrix} 5 \\ 5 \end{pmatrix} \qquad \vec{u} = \begin{pmatrix} 2 \\ 1 \end{pmatrix} \qquad \vec{x} = \begin{pmatrix} 7 \\ 10 \end{pmatrix}$$

$$\vec{x} = \vec{v} + r \cdot \vec{u}$$

$$\begin{pmatrix} 7 \\ 10 \end{pmatrix} \overset{?}{=} \begin{pmatrix} 5 \\ 5 \end{pmatrix} + r \cdot \begin{pmatrix} 2 \\ 1 \end{pmatrix}$$
$$\Rightarrow \begin{pmatrix} 7 \\ 10 \end{pmatrix} \overset{?}{=} \begin{pmatrix} 5 \\ 5 \end{pmatrix} + \begin{pmatrix} r \cdot 2 \\ r \cdot 1 \end{pmatrix}$$

Jetzt können wir die Gleichung mit Vektoren in ihre einzelnen Komponenten aufteilen:

$7 = 5 + r \cdot 2$ (1)
$10 = 5 + r \cdot 1$ (2)

Wir müssen also überprüfen, ob sich für r ein Wert so finden lässt, dass diese beiden Gleichungen erfüllt sind.

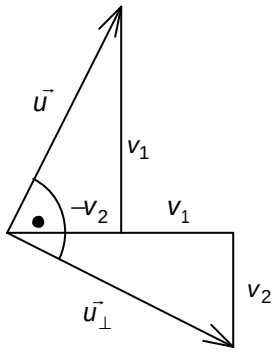
(1) vereinfacht: $r = 1$
(2) vereinfacht: $r = 2{,}5$

Da die Werte für *r* nicht übereinstimmen, kann der Punkt nicht auf der Geraden liegen, denn durch einsetzen eines passenden *r* in die Parameterdarstellung der Geraden ließe sich der Punkt erreichen, wenn er auf der Geraden liegen würde.

Lot

Ein zu $\vec{v} = \begin{pmatrix} v_1 \\ v_2 \end{pmatrix}$ senkrechter Vektor $\vec{u}$ ist $\vec{u} = \begin{pmatrix} -v_2 \\ v_1 \end{pmatrix}$ oder $\vec{u} = \begin{pmatrix} v_2 \\ -v_1 \end{pmatrix}$

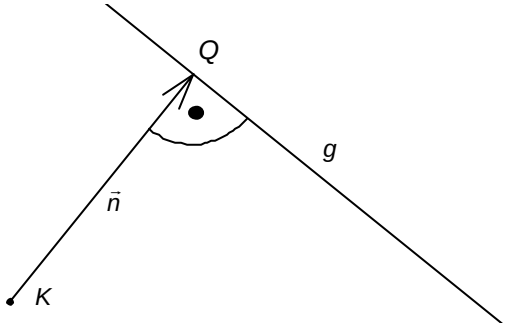
Beweis durch Veranschaulichung:



Abstand Punkt-Gerade
Variante 1

Gegeben ist eine Gerade *g* und ein Punkt *K* als Kugelmittelpunkt. Nun wollen wir den Abstand von *K* zu *g* berechnen. Dazu denken wir uns eine Senkrechte *l* auf *g* durch *K*. Diese Senkrechte *l* schneidet *g* in *Q*; $\overline{KQ}$ ist dann der gesuchte Abstand.

Um nun diesen Abstand zu berechnen, benötigen wir zunächst die genaue Position von *Q*.



Wir definieren:

$g: \quad \vec{x} = \vec{v} + t \cdot \vec{u}$

$l: \quad \vec{x} = \vec{K} + s \cdot \vec{n}$

Schnittpunkt zweier Geraden

Beispielwerte:

$$\vec{v} = \begin{pmatrix} 3 \\ 2 \end{pmatrix} \quad \vec{u} = \begin{pmatrix} 2 \\ 1 \end{pmatrix} \quad \vec{k} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \quad \vec{n} = \begin{pmatrix} -1 \\ 2 \end{pmatrix}$$

Der Schnittpunkt sei *Q*. Da *Q* auf beiden Geraden liegt, muss es ein t_q und ein s_q geben, sodass gilt:

$$\left.\begin{aligned} \vec{q} &= \vec{v} + t_q \cdot \vec{u} \\ \vec{q} &= \vec{k} + s_q \cdot \vec{n} \end{aligned}\right\} \vec{v} + t_q \cdot \vec{u} = \vec{k} \cdot s_q \cdot \vec{n}$$

Jetzt schreibt man die Vektorgleichung als zwei Gleichungen für die *x*- und *y*-Koordinate und rechnet t_q und s_q aus.

Mit t_q (oder s_q) berechnet man den Schnittpunkt $\vec{q} = \vec{v} + t_q \cdot \vec{u}$.

Mittels Linearem Gleichungssystem ergeben sich bei unserem obigen Beispiel diese Werte: $\vec{q} = \begin{pmatrix} 1 \\ 3 \end{pmatrix}$

Jetzt: $\overrightarrow{KQ} = \vec{q} - \vec{k} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$

$$\left|\overrightarrow{KQ}\right| = \sqrt{5}$$

Variante 2

Mit Hilfe des Skalarprodukts

Skalarprodukt

Mit Hilfe des Skalarprodukts wird zwei Vektoren ein Skalar, d. h. eine Zahl, zugeordnet.
Das Skalarprodukt kann sehr nützlich sein, da es bei zwei senkrecht aufeinander stehenden Vektoren 0 ist und weitere interessante Eigenschaften besitzt.

Definition:

$$\vec{a} \circ \vec{b} = \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} \circ \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} = a_1 b_1 + a_2 b_2$$

Offenbar gilt das Kommutativgesetzt, d. h. $\vec{a} \circ \vec{b} = \vec{b} \circ \vec{a}$, da nur die Faktoren vertauscht werden.

Eigenschaften

Senkrechte Vektoren

Beispiel:

$$\begin{pmatrix}5\\4\end{pmatrix}\circ\begin{pmatrix}-4\\5\end{pmatrix}=0$$

Allgemein:

Sei $\vec{a}=\begin{pmatrix}a_1\\a_2\end{pmatrix}$ gegeben, dann ist $\vec{n}=\begin{pmatrix}-a_2\\a_1\end{pmatrix}$ mit $\vec{n}\perp\vec{a}$.

Ebenso gilt:

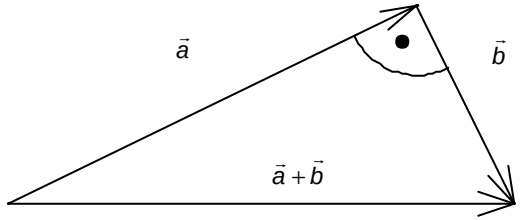
$$\vec{a}\circ r\cdot\vec{n}=\begin{pmatrix}a_1\\a_2\end{pmatrix}\circ\begin{pmatrix}-r\cdot a_2\\r\cdot a_1\end{pmatrix}=-ra_1a_2+ra_1a_2=0$$

Das zeigt, dass für jeden Vektor, der orthogonal auf $\vec{a}$ ist, das Skalarprodukt mit $\vec{a}$ 0 ergibt.

Einen auf $\vec{a}$ senkrechten Vektor $\vec{n}$ nennt man im Übrigen auch Normalenvektor.
Hat dieser den Betrag 1, spricht man auch von einem Einheitsnormalenvektor. (Schreibweise $\vec{n}_0$

Behauptung:
Das Skalarprodukt zweier Vektoren ist genau dann 0, wenn sie senkrecht aufeinander stehen.

Sei $\vec{a}\perp\vec{b}$



$$\vec{a}=\begin{pmatrix}a_1\\a_2\end{pmatrix}\qquad\vec{b}=\begin{pmatrix}b_1\\b_2\end{pmatrix}$$

$\left|\vec{a}+\vec{b}\right|^2=\left|\vec{a}\right|^2+\left|\vec{b}\right|^2$ (Satz des Pythagoras)

Weiter umgeformt:

$$\left|\begin{pmatrix}a_1+b_1\\a_2+b_2\end{pmatrix}\right|^2=a_1^2+a_2^2+b_1^2+b_2^2$$

$$a_1^2+2a_1b_1+b_1^2+a_2^2+2a_2b_2+b_2^2=a_1^2+a_2^2+b_1^2+b_2^2$$

$$2a_1b_1+2a_2b_2=0$$

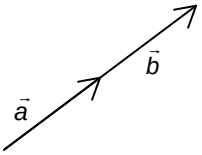
$$2\left(a_1b_1+a_2b_2\right)=0$$

$$\vec{a}\circ\vec{b}=0$$

Da im Obigen nur Äquivalenzumformungen vorgenommen wurden, sind die einzelnen Schritte 'rückwärts' genauso gültig. Also gilt auch der Kehrsatz.

Parallele Vektoren

$\vec{a} \circ \vec{a} = a_1^2 + a_2^2 = |\vec{a}|^2$



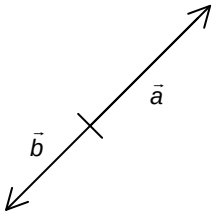
Wegen $\vec{a} \parallel \vec{b}$ ist $\vec{b} = r \cdot \vec{a}$ mit $r > 0$

$\Rightarrow \vec{a} \circ \vec{b} = \vec{a} \circ r \cdot \vec{a} = r \cdot a_1^2 + r \cdot a_2^2 = r\left(a_1^2 + a_2^2\right) = r|\vec{a}|^2$

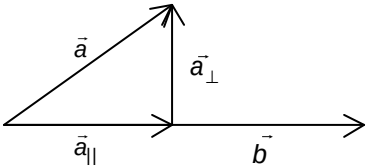
$r|\vec{a}|^2 = |\vec{a}| \cdot r|\vec{a}| = |\vec{a}| \cdot |\vec{b}|$

$\vec{b} = -x\vec{a}$

$\vec{a} \circ \vec{b} = -|\vec{a}| \cdot |\vec{b}|$



Allgemein



Der Vektor $\vec{a}$ wird in zwei Vektoren $a_{||}$ und $a_\perp$ aufgeteilt.

$\vec{a} \circ \vec{b} = ?$

$\vec{a} = \vec{a}_{||} + \vec{a}_\perp$

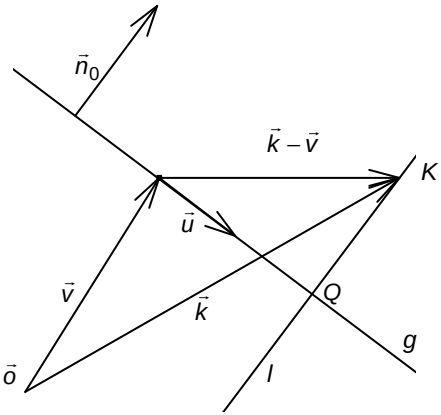
$\vec{a} \circ \vec{b} = \left(\vec{a}_{||} + \vec{a}_\perp\right) \circ \vec{b} = \vec{a}_{||} \circ \vec{b} + \underbrace{\vec{a}_\perp \circ \vec{b}}_{0} = \vec{a}_{||} \circ \vec{b}$

$\vec{a} \circ \vec{b} = \vec{a}_{||} \circ \vec{b}$

Abstand Punkt-Gerade

Nachdem wir diese kleinen Vorbereitungen getroffen haben, kommen wir zu unserem eigentlichen Problem zurück: Das Berechnen des Abstandes zwischen einem Punkt *K* und einer Geraden *g*.

g sei durch *g* : $\vec{x} = \vec{v} + r \cdot \vec{u}$ gegeben. Der Schnittpunkt des Lots durch *K* auf *g* werde wieder mit *Q* bezeichnet. Nun kann man mit Hilfe eines Einheitsnormalenvektors $\vec{n}_0$ auf *g* der Abstand sehr einfach berechnen: $\left(\vec{k} - \vec{v}\right) \circ \vec{n}_0 = \pm d$



Es ist $\overline{QK} = d$

d ist gesucht, da es der Abstand des Kugelmittelpunktes *K* zur Geraden *g* ist.

$(\vec{k} - \vec{v}) \circ \vec{n}_0 = \pm d$

$(\vec{k} - \vec{v}) \circ \vec{n}_0$ ist positiv, wenn der Punkt *K* auf derjenigen Seite von *g* liegt, auf die $\vec{n}_0$ zeigt. Somit kann man sogar feststellen, auf welcher Seite der Geraden sich die Kugel befindet.

Die Physik zum Spiel

Um unser Spiel möglichst realistisch darstellen zu können, mussten wir uns mit einigen physikalischen Gesetzmäßigkeiten und Einheiten vertraut machen. Um etwa in unser Spiel die Schwerkraft mit einbauen zu können, war es nötig, die Beschleunigung einzuführen.

Beschleunigte Bewegung

Aber was ist Beschleunigung eigentlich genau? Dieser Frage mussten wir auf den Grund gehen. Jeder weiß, dass ein Stein immer schneller wird, wenn man ihn aus dem Fenster fallen lässt, er fällt also „beschleunigt“. Zur beschleunigten Bewegung haben wir einige Versuche durchgeführt.

1. Versuch:

Wir haben ein kleines Auto auf den Tisch gestellt und ein Gewicht „fallen“ gelassen, dessen Gewichtskraft über eine Umlenkrolle am Tischrand auf das Auto übertragen wurde. Neben dem Auto lag ein Meterstab, an dem man nach bestimmten Zeitintervallen die Position ablesen konnte. Als wir den Versuch durchführten, bemerkten wir, dass sich das Auto nicht gleichförmig bewegte, sondern schneller wurde.

Nach der 1. Sekunde hatte das Auto eine Geschwindigkeit von ungefähr $2\frac{\text{LE}}{\text{s}}$ (LE = Längeneinheit), nach der 2. Sekunde $4\frac{\text{LE}}{\text{s}}$, nach der 3. $6\frac{\text{LE}}{\text{s}}$ und so weiter. Das heißt, das Auto beschleunigte.

Als Maß für die Beschleunigung *a* dient die Geschwindigkeitsänderung pro Zeitintervall. Die Geschwindigkeitsänderung wird als Δv, die vergangene Zeit als Δt bezeichnet. Man definiert: $\bar{a} = \frac{\Delta v}{\Delta t}$. Der Strich über dem *a* kennzeichnet eine durchschnittliche Beschleunigung.

Als Einheit der Geschwindigkeit verwenden wir $1\,\frac{\text{m}}{\text{s}}$, für die Zeit s (Sekunden). Damit ergibt sich als Einheit für die Beschleunigung: $[\bar{a}] = \frac{\frac{\text{m}}{\text{s}}}{\text{s}} = \frac{\text{m}}{\text{s}^2}$.

Hier ein Beispiel, wie man die Beschleunigung $\bar{a}$ berechnet:

Ein Auto beschleunigt von $0\,\frac{\text{km}}{\text{h}}$ auf $100\,\frac{\text{km}}{\text{h}}$ in 10s.

Die Geschwindigkeitsänderung Δv beträgt $100\,\frac{\text{km}}{\text{h}}$.

Da wir aber die Geschwindigkeit in $\frac{\text{m}}{\text{s}}$ angeben, müssen wir die $100\,\frac{\text{km}}{\text{h}}$ erst in $\frac{\text{m}}{\text{s}}$ umrechnen.

$$100\,\frac{\text{km}}{\text{h}} = \frac{100\,\text{km}}{\text{h}} = \frac{10000\,\text{m}}{3600\,\text{s}} = \frac{1000}{36}\cdot\frac{\text{m}}{\text{s}} \approx 28\,\frac{\text{m}}{\text{s}}$$

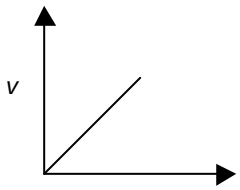
Dieses Δv noch durch 10 s teilen. $\frac{28\frac{m}{s}}{10s} = 2{,}8\frac{m}{s^2}$.

Das Auto hat also eine Beschleunigung von $\bar{a} = 2{,}8\,\frac{\text{m}}{\text{s}^2}$.

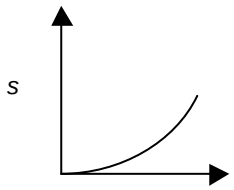
Beschleunigte Bewegung in Diagrammen

Nun stellten wir konstante Beschleunigung in einem *v – t* und einem *s – t* Diagramm dar, um herauszufinden, was die Werte, die wir bei unserem Versuch erhielten, für ein Schaubild ergeben. Wir stellten fest, dass eine konstante Beschleunigung in einem *v – t* Diagramm eine Gerade, in einem *s – t* Diagramm eine Kurve, die wie eine Parabel aussieht, ergibt.

v - t Diagramm

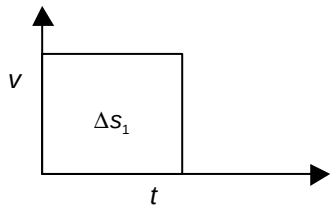


s - t Diagramm



(*a* jeweils konstant)

Wir fragten uns, ob dies nur so aussieht oder ob es sich tatsächlich um eine Parabel handelt.
Dazu stellten wir zunächst eine Vorüberlegung an und zeichneten ein *v - t* - Diagramm für eine Bewegung mit konstanter Geschwindigkeit:

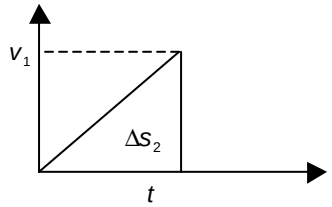


Dafür gilt: $\Delta s = \bar{v} \cdot \Delta t$

Δs entspricht also der von der v - Kurve und der t-Achse eingeschlossenen Fläche. Wir erhalten ein Rechteck.

Wir überlegten uns, dass Δs auch dann diese Fläche ist, wenn v nicht konstant ist.

Uns interessierte der Fall, dass a konstant ist (und $s_0 = 0$, $v_0 = 0$, $t_0 = 0$).



Δs entspricht wiederum der von der v-Kurve und der t - Achse eingeschlossenen Fläche, allerdings erhalten wir nun ein rechtwinkliges Dreieck; sein Flächeninhalt ist genau die Hälfte des Flächeninhalts des vorherigen Rechtecks.

Hierbei errechnet sich v_1 aus $\overline{a} \cdot \Delta t$. Die Fläche von Δs_2 entspricht genau der Hälfte von Δs_1.
Daraus folgt:

$$\Delta s = \frac{1}{2} \cdot v_1 \cdot \Delta t \text{ mit } v_1 = a \cdot \Delta t$$

$$\Delta s = (a \cdot \Delta t) \cdot \Delta t \cdot \frac{1}{2}$$

$$\Delta s = \frac{1}{2} \cdot a \cdot \Delta t^2$$

bzw. $s = \frac{1}{2} \cdot a \cdot t^2$ für s = Endposition, t = Endzeit

Dies gilt also für den Fall, dass Anfangszeit, Anfangsposition, Anfangsgeschwindigkeit jeweils 0 sind und die Beschleunigung konstant ist.

Kraft und Beschleunigung

Wir hatten nun allerdings noch nicht geklärt, wodurch eine Beschleunigung hervorgerufen wird. Dazu machten wir uns noch einmal klar:

Ohne Einwirkung einer Kraft ändert sich eine Bewegung nicht!
Umgekehrt:
Kräfte sind die Ursache von Bewegungsänderungen.
Da stellte sich natürlich die Frage, wie das genau funktioniert.

Das 2. Newtonsche Gesetz

Anhand von Versuchen und Nachdenken kamen wir zu folgendem Ergebnis (F = Kraft, m = Masse, a = Beschleunigung, c = Konstante):

1. Bei konstanter Beschleunigung ist $F \sim m$.

2. Bei konstanter Masse ist $F \sim a$.

Zusammen bedeutet dies: $F \sim m \cdot a$.

$$\Rightarrow F = c \cdot m \cdot a$$

Nun wollten wir natürlich herausfinden, wie groß die Konstante in unserer Formel ist. Dazu griffen wir noch einmal auf unseren ersten Versuch mit dem Wagen und den Gewichten zurück. Wir setzten die uns bereits bekannten Werte in die neu erarbeiteten Formeln ein und fanden heraus, dass die Konstante ungefähr 1 beträgt. Dies ist kein Zufall, denn in Wirklichkeit ist es gerade andersherum: Die Einheit „1 Newton“ ist gerade so gewählt, dass $c = 1$ gilt.
Folglich fällt unser Proportionalitätsfaktor c aus unserer Formel heraus:

$$\Rightarrow F = m \cdot a$$

Und das ist das 2. Newtonsche Gesetz!

Es lautet in Worten: Um einen Körper der Masse m mit der Beschleunigung a zu beschleunigen, muss auf den Körper die Kraft $F = m \cdot a$ einwirken.
Dies bedeutet folgendes für die Einheiten:

$$\mathrm{F = m \cdot a \Rightarrow N = kg \cdot \frac{m}{s^2}}$$

Also ist 1 N die Kraft, die nötig ist um einen Körper der Masse $m = 1\ \mathrm{kg}$ in $1s$ um $1\frac{\mathrm{m}}{\mathrm{s}}$ zu beschleunigen.

Zu Ehren des Physikers wurde also $1\ \mathrm{kg} \cdot \frac{\mathrm{m}}{\mathrm{s}^2}$ zu 1 N abgekürzt.

Die Erdanziehungskraft

Um berechnen zu können, wie groß die durch die Erde hervorgerufene Beschleunigung bei einem beliebigen Massestück ist, benötigen wir noch die Kraft, die den Körper beschleunigt, also die Erdanziehungskraft, auch Schwerkraft genannt. Die Anziehungskraft der Erde hängt aber vom Ort ab, an dem man steht, man kann sie mit Hilfe des Ortsfaktors g berechnen: $G = m \cdot g$. (G ist die Gewichtskraft, m die Masse des Objekts). Dass die Anziehungskraft vom Ort abhängt, beruht unter anderem darauf, dass die Erde nicht ganz rund, sondern ellipsenförmig ist. Je weiter man sich vom Erdmittelpunkt wegbewegt, desto geringer die Anziehungskraft. Bei uns beträgt der Ortsfaktor $g = 9{,}81\frac{\mathrm{N}}{\mathrm{kg}}$ d. h. auf einen Körper der Masse 1 kg wirkt eine Kraft von 9,81 N ein.
Diese Gewichtskraft beschleunigt nun den Körper gemäß $F = m \cdot a$, daher können wir $m \cdot g$ für F einsetzen.

$$\Rightarrow m \cdot g = m \cdot a \Rightarrow a = g$$

Das heißt, dass die Beschleunigung, die Körper auf der Erde erfahren, gleich dem Wert des Ortsfaktors ist.

Hier noch ein paar Beschleunigungswerte von anderen Orten:

Am Äquator (Erde):	$9{,}787\frac{m}{s^2}$
100 km über der Erde:	$9{,}53\frac{m}{s^2}$
auf dem Mond:	$1{,}62\frac{m}{s^2}$
auf dem Mars:	$3{,}71\frac{m}{s^2}$
auf der Sonne:	$274\frac{m}{s^2}$

Nach dieser Rechnung müsste ja eigentlich eine Feder genauso schnell zu Boden fliegen wie ein Stein. Jeder weiß, dass das zumindest auf der Erde nicht so ist. Aber warum?
Das hängt von etwas ganz anderem ab, nämlich vom Luftwiderstand. Dies haben wir in einem kleinen Versuch veranschaulicht. In einem Glasrohr befanden sich ein Metallkügelchen und eine kleine Feder. Als wir dieses Glasrohr umdrehten, fiel das Metallkügelchen viel schneller herunter als die Feder. Anschließend haben wir die Luft mit Hilfe einer Pumpe aus dem Glasrohr gepumpt und den Versuch wiederholt, doch dieses Mal fiel das Ergebnis anders aus. Das Metallkügelchen und die Feder waren gleich schnell, was die vorher berechnete Formel für die Erdbeschleunigung bestätigt.

Horizontale Bewegung

Nun hatten wir aber noch ein weiteres Problem: Die Kugel würde sich in unserem Spiel ja nicht nur vertikal, sondern auch horizontal bewegen. Da stellt sich die Frage, wie sich die vertikale Beschleunigung durch eine horizontale Bewegung ändert. Dazu haben wir einen weiteren Versuch durchgeführt. Wir hatten ein Gerät, das gleichzeitig eine Kugel senkrecht zu Boden fallen ließ und eine weitere waagerecht abschoss. Das Ergebnis war verblüffend: Beide Kugeln kamen gleichzeitig auf dem Boden an. Dieses Resultat war sehr erleichternd für uns, da wir die horizontale in Bezug auf die vertikale Bewegung nicht weiter berücksichtigen mussten.
Nun mag man aber einwenden, dass bei einem Pinballspiel die Kugel nicht senkrecht zu Boden fällt, sondern eine schräge Ebene hinabrollt. Natürlich verringert sich dann auch der Beschleunigungswert. Das ist aber kein größeres Problem, da wir einfach mit einem bestimmten Prozentsatz des ursprünglichen Beschleunigungswertes rechnen. Dieser Prozentsatz errechnet sich aus dem Winkel der schrägen Ebene zur Horizontalen.

Mit diesen Erkenntnissen hatten wir die physikalischen Grundlagen erarbeitet, die wir für eine realistische Simulation der Bewegung benötigten. Wir mussten unsere Ergebnisse also nur noch programmiertechnisch umsetzen.

Unser Spiel

Nach soviel trockener Theorie konnten wir uns wieder ans Programmieren unseres Spiels machen. Das Eigentliche, was ein richtiges Spiel ausmacht, fehlte nämlich noch: die zu entfernenden Steine, der Punktestand und noch viele weitere Details, die das Ganze interessanter gestalten.
Neben der Mathematik und der Physik lernten wir noch weitere Programmiertechniken.

Klassen

Um unser Programm übersichtlicher zu gestalten lernten wir, wie man sogenannte Klassen verwendet. Klassen sind das grundlegende Konzept der objektorientierten Programmierung. Eine Klasse (man könnte auch sagen: Der Bauplan für ein Objekt) besteht aus Attributen (was beschreibt das Objekt?) und Methoden (was kann man mit dem Objekt machen?). Das klingt zwar kompliziert, aber man kann sich dieses Konzept an einem einfachen Beispiel verdeutlichen: Ein Auto hat ein bestimmtes Aussehen (Farbe, Form usw.) und bestimmte „Fähigkeiten“, an erster Stelle wohl das Fahren (also auch Beschleunigung, Bremsen, Parken).

Hier ein Auszug der `KugelKlasse` unseres Spiels:

```
class KugelKlasse
{
public :  // hier werden die Attribute
             und die Methoden der
             KugelKlasse  deklariert

// Attribute

float x_mitte;
// x-Koordinate der Kugel
float y_mitte;
// y-Koordinate der Kugel
float radius;
// Radius der Kugel
float farbe[3];
// Farbe der Kugel
float geschwindigkeit_x;
// Geschwindigkeit in x-Richtung
float geschwindigkeit_y;
// Geschwindigkeit in y-Richtung
// Methoden
void init(float X_mitte,
float  Y_mitte, float Radius, float
       Farbe_R, float Farbe_G, float
       Farbe_B,
float  geschwindigkeit_X,
float  geschwindigkeit_Y);
void bewegen(double diff_t);
void zeichnen();

};
```

In dieser `KugelKlasse` ist alles enthalten, was man für die Kugel benötigt: sie wird initialisiert, gezeichnet und bewegt.

In einer weiteren Klasse wird dann, um unser Beispiel mit dem Auto noch einmal aufzugreifen, eine Tankstelle mit ihren Eigenschaften beschrieben usw.
Diese Attribute und Methoden werden ein Mal festgelegt und können dann für das ganze Programm verwendet werden.
Man kann diese Eigenschaften dann auch auf andere Objekte übertragen (*„vererben“*), die ähnliche Eigenschaften haben. So muss nicht jedes Mal alles neu in den Computer eingegeben werden. Also kann man beispielsweise eine Klasse mit den allgemeinen Eigenschaften eines Fahrzeuges machen und in weiteren Klassen dann die zusätzlichen Eigenschaften eines PKWs, Lastwagens, Zuges etc. hinzufügen. Die Methoden müssen unter Umständen auch neu eingegeben werden, und zwar dann, wenn neue Attribute dazu gekommen sind.
Den in den Klassen aufgelisteten Attributen werden dann im Hauptprogramm Werte zugewiesen, d.h. die Eigenschaften wie Größe, Farbe usw. bestimmt.

Eine Frage stellt sich aber bei den Klassen natürlich noch: Wenn nun zwei Objekte in irgendeiner Weise eine Beziehung zueinander haben (also wenn dann zum Beispiel das Auto an der Tankstelle tankt), muss man sich überlegen in welcher Klasse es sinnvoll ist, die entsprechende Methode zu implementieren.

Für unser Programm haben wir dann für jede Klasse auch eine (genau genommen sogar zwei) eigene Datei(en) erstellt. Auch dies hatte natürlich einen Grund. Ein Vorteil des Unterteilens in Dateien ist, dass man beim Kompilieren eine Menge Zeit sparen kann, da immer nur die Datei neukompiliert (berechnet) wird, die auch verändert wurde. Des Weiteren können so mehrere Leute an einem Programm arbeiten, was bei einer einzigen Datei schwierig ist.

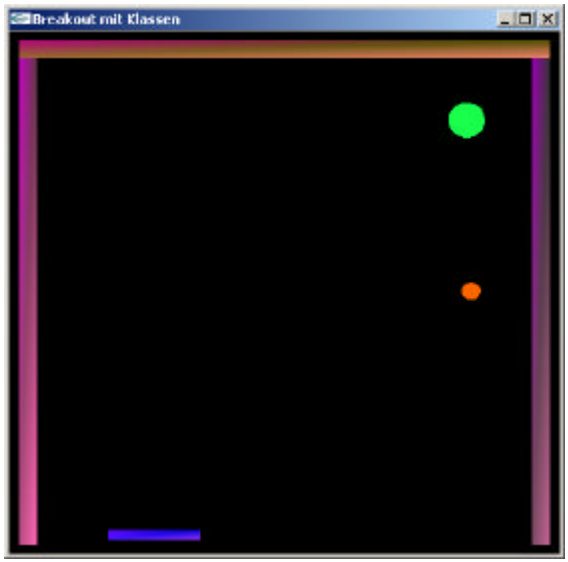
Elemente

Nun kommen wir zu den Elementen des Spiels. Damit der Spieler selbst auch etwas zu tun hat, mussten wir zuerst ein Paddle kreieren, das durch die Tastatur gesteuert wird. Das Paddle bekam die Grundeigenschaften des Rahmens vererbt, jedoch auch noch zusätzlich die Bewegung. Dazu wird dann eine Tastaturabfrage gestartet. Wenn also eine bestimmte Taste gedrückt wird, folgt der Befehl des Verschiebens nach rechts, links, oben oder unten.
Nun hatte das Spiel jedoch noch nicht sein eigentliches Ziel: die Steine. In einer neuen Klasse wurden also Steine festgelegt, die bei der Kollision mit der Kugel oder aber auch erst nach mehrmaligem Treffen verschwinden sollten. Die Position der einzelnen Steine hängt vom jeweiligen

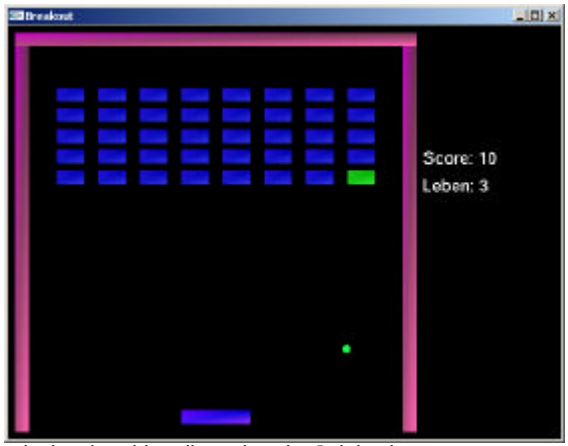
Level ab, welches dann in der `LevelKlasse` bestimmt wird. Das nächste Level wird (wie sollte es auch anders sein..?) nach Beenden des Levels, also wenn alle Steine weg sind, aufgerufen.
Mit den Steinen verbunden entstand dann auch gleich der Punktestand, der bei jeder Kollision der Kugel mit einem Stein hochgesetzt wird. Jedoch gibt es noch keinen wirklichen Grund, weshalb man die Kugel innerhalb der Spielfläche halten sollte – sie kommt leider wieder von selbst zurück, sobald man sie verliert. Deshalb fügten wir noch die Leben ein, was bedeutet, dass nach dreimaligem Verlust der Kugel das Spiel zu Ende ist (bzw. vorerst direkt neu gestartet wird).
Damit konnte man doch schon mal etwas anfangen. Aber etwas eintönig war das ja leider noch, weshalb wir die sogenannten Goodies hinzufügten. Diese „fallen“ zufällig aus den Steinen, wenn diese verschwinden, und können dann mit dem Paddle aufgesammelt werden. Das Wort „Goody“ setzt jedoch nicht vorraus, dass sich hinter dem Gegenstand ein positiver Effekt verbirgt. So gibt es welche, die einfach nur Bonuspunkte einbringen, andere, die einem ein zusätzliches Leben hinzufügen, noch andere, die das Paddle breiter werden lassen, jedoch auch wieder andere, durch die es schmaler wird.

Soweit sind wir bisher mit unserem Spiel gekommen. Die Theorie zu schräg liegenden Hindernissen steht bereits auch, zum Programmieren hat die Zeit dann jedoch leider nicht mehr gereicht. Aber wir werden trotzdem weiter daran arbeiten!

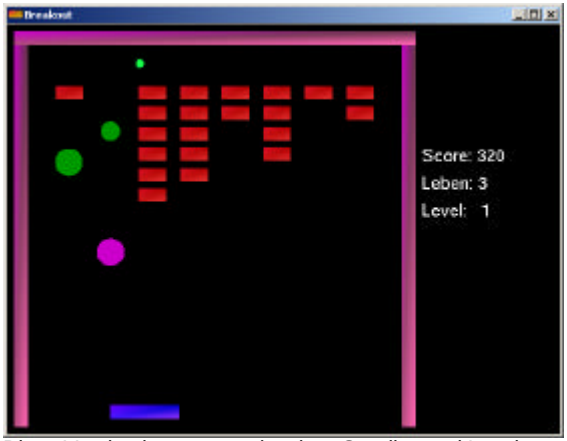
Hier einige Stationen unseres Spiels:



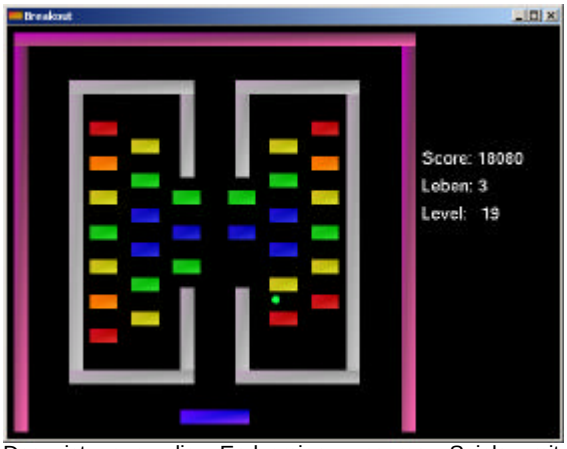
Noch hat es nur Rahmen, Paddle und Kugeln..



..doch schon hier gibt es bereits Spielsteine



Diese Version hat nun auch schon Goodies und Level



Das ist nun die Endversion unseres Spiels mit verschiedenen Steinen, die beliebig angeordnet werden können

Die Spielanleitung

Diese Spielanleitung entwickelten wir während dem Kurs, damit auch andere unser Spiel problemlos testen konnten.

Breakout

Ein Computerspiel des Mathematik-Kurses der Science Academy Baden-Württemberg 2004

1) Ziel des Spiels:

Das Ziel des Spieles besteht darin, so viele „Score-Punkte“ wie möglich zu sammeln. Punkte erhält man durch das Abschießen und Zerstören von Steinen mit der Kugel, sowie durch die sogenannten „Goodies“.

2) Das Spielfeld:

Zu Beginn des Spiels ist die Kugel auf dem Paddle platziert. Anschließend bewegt sie sich nach oben in Richtung Steine. Das Paddle lässt sich mit Hilfe der Tasten „a“ und „d“ nach links bzw. nach rechts bewegen. Die Tasten „w“ und „s“ dienen der Steuerung in vertikaler Richtung.
Die Kugel fällt unter dem Spielfeld in das „Nichts“. Dies kann man nur verhindern, indem man die

Kugel mit dem Paddle wieder nach oben „schlägt“. Jeder Kugelverlust führt zum Abzug eines Lebens. Bei null Leben ist das Spiel verloren und beginnt von vorne.

Tastenbelegung:

Taste	Funktion
a	Paddle nach links
d	Paddle nach rechts
w	Paddle nach oben
s	Paddle nach unten
p	Pause
n	Zurücksetzen der Kugel
q	Beenden

3) Die Steine:

Die Steine haben verschiedene Farben. Diese reichen von blau, über grün, gelb und orange bis hin zu rot. Blaue Steine müssen fünf Mal von der Kugel abgeschossen werden, um vollständig zerstört zu werden, grüne Steine viermal, gelbe dreimal usw. Rote Steine werden mit der nächsten Berührung der Kugel automatisch gelöscht. In jedem Fall (auch bei der Löschung der Steine) wird die Kugel vom Stein reflektiert. Hierbei folgt die Kugel den Gesetzen der Physik (Einfallswinkel = Ausfallswinkel, ...). Bei jeder Kollision zwischen Stein und Kugel werden dem Punktekonto zehn „Score Punkte“ zugeschrieben.
Graue Steine verschwinden nie und geben auch keine Punkte. Sie dienen lediglich als Hindernisse.

blau → grün → gelb → orange → rot

4) Spielstandsanzeige:

Die Spielstandsanzeige besteht aus den drei Anzeigen: Score, Leben und Level. Die Anzeige ist zum größten Teil selbsterklärend.
Die „Score-Anzeige“ Zeigt den Gesamtspielstand an (also auch den der bereits gespielten Level) an.

5) Die „Goodies“:

Um das Spiel zu vereinfachen (oder auch zu erschweren) gibt es in „Breakout“ sogenannte „Goodies“. Diese werden als Kugeln dargestellt, die von oben nach unten fallen. Die „Goodies“ erscheinen zufällig nach der Zerstörung der Steinen. Sie starten am Ort des (soeben verschwundenen) Steins und fallen langsam nach unten. Werden sie mit dem Paddle aufgesammelt, so führt dies zu folgenden Boni:

Goody	Bonus
	Das „Punkte-Goody“ fügt dem Score Punkte hinzu. Je größer das Goody, desto mehr Punkte gibt es für das Einsammeln.
	Das „Leben-Goody“ bringt ein zusätzliches Leben.
	Das „Paddle-größer-Goody“ vergrößert das Paddle. Es kann im Spiel nur zwei Mal eingesammelt werden, da sonst das Paddle zu groß würde.
	Das „Paddle-kleiner-Goody“ ist ein „Bady“ und verkleinert das Paddle. NICHT EINSAMMELN!

Viel Spaß beim Spielen!

Rotation

Rotation? Was ist das überhaupt? Die Rotation wurde durchgeführt, damit alle Teilnehmer der Science Academy auch etwas von dem mitbekamen, was denn in den anderen Kursen so ablief oder was der aktuelle Stand der jeweiligen Kurse war. Dazu bereitete jeder Kurs verschiedene Vorträge vor. Der Vormittag war in halbstündige Blöcke zerteilt, dabei wurden jeweils 10 Minuten für den Vortrag verwendet, weitere 10 Minuten für Diskussionen und die restlichen 10 Minuten, um zu den anderen Vorträgen zu gelangen. Da wir vier Vorträge vorzubereiten hatten, gliederten wir in Themenblöcke, die von verschiedenen Gruppen behandelt und zu einem Referat ausgearbeitet wurden. Bis alles am Tag der Rotation perfekt stand, gab es noch einiges zu tun. Plakate mussten gestaltet werden, die letzten Vorbereitungen für die Vorträge wurden getroffen. Dann hielten wir natürlich auch noch Probevorträge, um unser Auftreten zu verbessern und einen reibungslosen Ablauf zu garantieren.

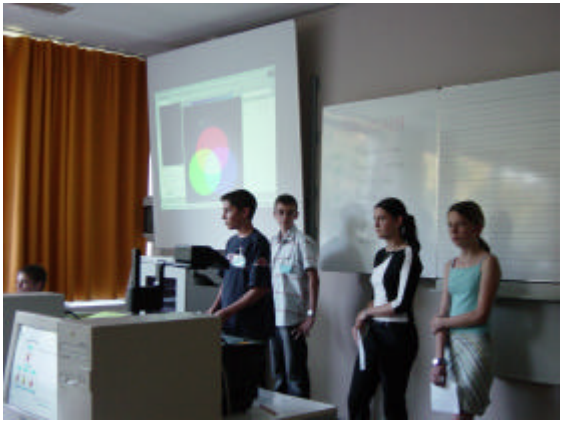
Dieser sah folgendermaßen aus: erst wurde zu den bestimmten Themen, wie z. B. Grafiken mit OpenGL und der Kugelbewegung, der Kollision usw. ein gehalten und danach gaben wir den Zuschauern noch einmal 10 Minuten Zeit, um Fragen zu stellen oder um unser selbst programmiertes Spiel zu testen, was dann auch großem Anklang fand.

Bei jedem Vortrag waren immer mindestens zwei Leiter anwesend, so konnten sie unseren Kursleitern ein Feedback zu den jeweiligen Vorträgen geben. Dies geschah in einer Sitzung, während wir am Mittag die „Highland Games“ abhielten. Das Feedback wurde als gut gemeinte Kritik an uns weitergegeben und sollte uns helfen, die beanstandeten Punkte zu verbessern.

Abschlusspräsentation

Die Abschlusspräsentation sollte unseren Eltern die Begeisterung vermitteln, die wir die ganzen zwei Wochen der Science Academy erlebt hatten. Deshalb wurde sie uns zum besonderen Anliegen. Wir hatten sie ähnlich geplant wie die Rotation, nur dass in der Abschlusspräsentation alle Teilnehmer über die selben Themen referierten. Außerdem stellte sich uns die Aufgabe, die Stellwand, die jeder Kurs zur Verfügung hatte, mit Artikeln usw. zu gestalten. Auf der Mathekurs-Stellwand waren eine Spielbeschreibung, ein Glossar, eine Beschreibung des Ausflugs nach Heidelberg ins Interdisziplinäre Zentrum für wissenschaftliches Rechnen (IWR) und vieles mehr zu finden. Natürlich mussten auch noch Werbeplakate und Wegweiser hergestellt werden. Die Wegweiser benötigten wir dringend, denn das Eckenberg-Gymnasium liegt nicht direkt im Zentrum und die Leute sollten unsere Vorträge rechtzeitig und leicht finden. Außerdem sollten sie die Zuschauer animieren, zu unseren Vorträgen zu kommen, denn es gab ja schließlich noch 5 andere Vorträge zur selben Zeit. Und so wollten wir den Besuchern die Entscheidung etwas erleichtern ☺... Doch durch Zusammenarbeit und Aufteilung der Aufgaben war das schnell erledigt. Die jeweiligen

Gruppen hielten auch dieses Mal wieder einen Probevortrag. Bald trafen auch schon die ersten Eltern, Verwandten und Geschwister ein. Die Vorträge liefen toll und die Zuschauerzahl ließ nicht zu wünschen übrig. Auch das „freie Spielen“ wurde wieder zum Erfolg. Wir spürten alle deutlich, dass uns die Rotations-Vorträge schon weitergebracht hatten und konnten im Gegensatz zum letzten Vortrag schon deutlich freier vor dem Publikum stehen. Alles in allem hat es uns sehr viel Spaß gemacht zusammen zu arbeiten und wir werden bestimmt noch etliche Male von unserer Erfahrung profitieren.



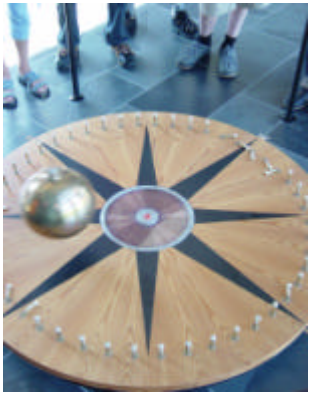
Bei der Abschlusspräsentation.

Exkursion nach Heidelberg

Am 06.09.2004 unternahmen wir eine ganztägige Exkursion nach Heidelberg.
Zuerst waren alle Teilnehmer und Leiter der Akademie im Hörsaal des DKFZ (Deutsches Krebs-Forschungszentrum) zu zwei Vorträgen, über Krebs & Krebsforschung und über Astronomie, die Entstehung von Sternen und Planetensystemen, eingeladen.
Danach ging jeder der sechs Kurse in verschiedene Fakultäten auf dem Universitätsgelände.

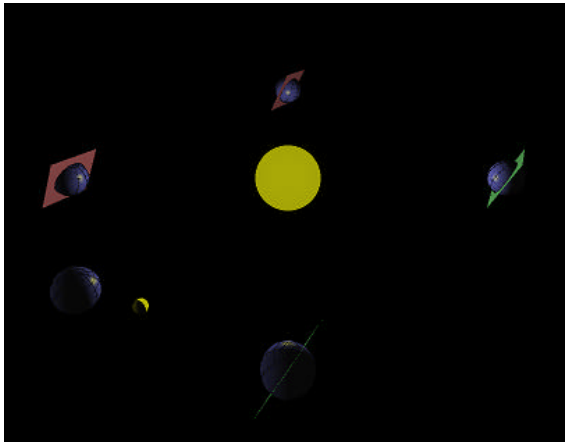
Unser Kurs im KIP

Unser Kurs unternahm zunächst einen kurzen Abstecher in das Kirchhoff-Institut für Physik (KIP), um das dortige Focault-Pendel zu besichtigen. Das Pendel beweist die Drehung der Erde: Es ist mit einer mechanischen Aufhängung an der Gebäudedecke befestigt und wird durch kurze magnetische Schwingungen in Bewegung gesetzt. Dabei schwingt es immer in derselben Ebene; da sich die Erde aber unter dem Focault-Pendel wegdreht, scheint es jemandem, der das Pendel beobachtet, als würde sich die Pendelebene drehen. Dies wird bei dem Pendel im KIP mit Hilfe einer Platte, auf der umklappbare Nägel befestigt sind, veranschaulicht.



Das Focault-Pendel im KIP

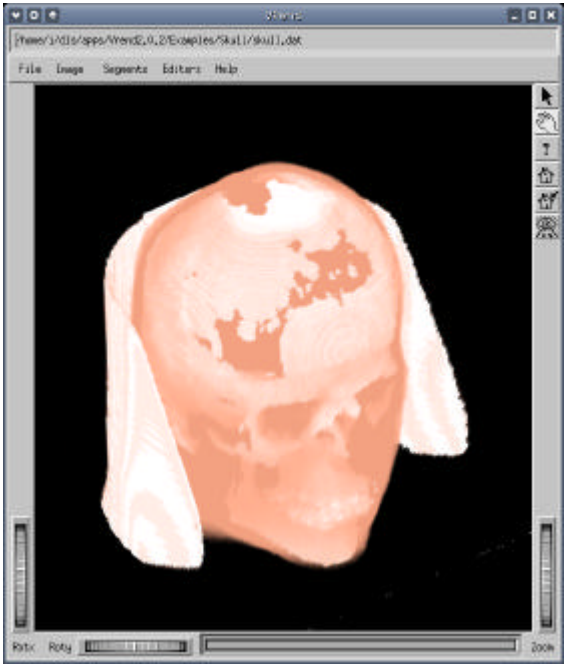
Später besuchten wir das Interdisziplinäre Zentrum für wissenschaftliches Rechnen (IWR).
In einem der Computerpools des IWRs bekamen wir zu Beginn eine Einführung in die notwendigsten Befehle von Linux durch Frau Dr. Susanne Krömker, die unsere Gastgeberin im IWR war.
Als wir soweit eingewiesen waren, sahen wir uns die Homepage von Frau Krömkers Arbeitsgruppe „Visualization and Numerical Geometry“ an. Von dieser Homepage luden wir uns dann die Musterlösung einer Programmierübungsaufgabe der Vorlesung „Lichtmodelle in der Computergrafik“ herunter. In diesem Programm wurde ein Solar-/ Sonnensystem auf einfache Weise dargestellt um Lichteffekte in OpenGL testen zu können, da mit dieser Grafikbibliothek auch beleuchtete Gegenstände dargestellt werden können. Frau Krömker erläuterte uns die dazugehörigen Quelldateien. In dem 3D-Modell des Sonnensystems enthalten sind die Sonne als Mittelpunkt, eine Erde die sich um die Sonne und ihre eigene Achse dreht, ein Mond der sich um die Erde und um seine eigene Achse dreht, sowie vier weitere Erden, die sich nur um ihre eigene Achse drehen und so platziert sind, dass sie die vier Jahreszeitenanfänge darstellen. Hierbei ist schön zu erkennen, wie die Sonne die Erde zu den verschiedenen Tages- und auch Jahreszeiten beleuchtet. Später durften wir auch selber einige Dinge, wie z.B. Farben, Blickwinkel oder Bewegungsgeschwindigkeit im Quelltext verändern und die dadurch entstehenden Effekte beobachten.



Das Sonnensystem-Programm aus dem IWR

Im zweiten Arbeitsblock bekam unser Kurs noch Einsicht in die Arbeit mit VRend. Mit diesem Programm konnten wir die Röntgenaufnahme des Kopfes einer Frau, der in der Kopfklinik durch eine Computertomographie aufgenommen wurde, in 3D betrachten und am Computer bearbeiten. Mit bearbeiten ist z.B. das Ändern von Farben, der Ansicht oder das entfernen von Haut und Fleisch

und der anschließenden Betrachtung des Schädels gemeint.



VRend-Programm aus dem IWR

Man kann also mit Hilfe des Programms zum Beispiel die Haut und Muskelschicht des Kopfes auf transparent stellen, so dass dann der Knochen des Schädels zu erkennen ist.

Dieses Verfahren kann Ärzten bei der Erkennung von verschiedensten Verletzungen helfen und sie bei der Behandlung dieser Verletzungen unterstützen.
Danach konnten wir dieses Verfahren noch an einem anderen Gegenstand ausprobieren. Dabei war zuerst allerdings nicht auszumachen, worum es sich dabei handelt, da nur ein viereckiger Kasten zu erkennen war. Doch mit Hilfe des Programms konnten wir das Objekt in dem Kasten schließlich identifizieren. – Es war ein Teddybär!
Nach diesen interessanten Einblicken in fortschrittliche medizinische Computerprogramme war unser Aufenthalt im IWR leider schon beendet.

Anschließend hatten alle Zeit, um in kleinen oder auch größeren Gruppen die Heidelberger Fußgängerzone zu erkunden und zum Ausklang des Tages gemeinsam im Clubhaus des Heidelberger Ruderklubs gemütlich zu speisen.

(Die Bilder auf dieser Seite sind mit freundlicher Genehmigung von Frau Dr. Krömker, IWR-Heidelberg abgedruckt)

Nachwort

Der gemeinsame Weg vieler Teilnehmer und der Leiter scheint sich nun zu trennen. Jedoch arbeiten wir immer noch an unserem Programm und werden uns sicherlich noch oft treffen, um das Erlebte zu reflektieren und die gemeinsamen Erfahrungen auszutauschen.
Unser besonderer Dank gilt vor allem unseren Kursleitern *Jörg Richter* und *Daniel Jungblut*, die uns immer unterstützt haben und uns immer mit viel Geduld halfen.
Es war eine schöne Zeit, die wir alle sicherlich nie vergessen werden!