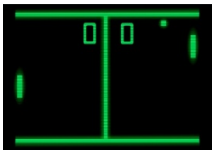


Vorwort

Computer sind aus dem Alltag der Jugendlichen (und natürlich auch der Erwachsenen) nicht mehr wegzudenken. Dabei spielen neben dem klassischen Desktop oder Laptop immer mehr auch neue mobile internetfähige Varianten wie Handy oder Tablet eine wichtige Rolle. Darüber hinaus sind sehr viele technische Geräte oder Anlagen mit programmierbaren oder fest programmierten Prozessoren ausgestattet. Dazu gehören Ticketautomaten, Spielkonsolen, Fernseher, Waschmaschinen, Flugzeuge, Züge, Kraftwerke und sehr vieles mehr.

Trotzdem haben die wenigsten Menschen je selber ein Programm für einen Computer oder ein solches Gerät geschrieben. Die meisten würden sich dies wohl auch nicht zutrauen. Eigentlich schade, dass uns die Mechanismen dieser für uns so wichtigen Welt verborgen bleiben. Auch im ICT-Unterricht an den Schulen kann - auch aus Zeitgründen - nur beschränkt auf das Programmieren eingegangen werden.

Diese Unterrichtseinheit bietet eine Möglichkeit über das Thema Grafiken bzw. Computerspiele bei Jugendlichen das Interesse für das Programmieren zu wecken. Die Programmierung einfacher grafischer Objekte, inklusive Farben und Bewegungen ermöglicht einen sehr anschaulichen Einstieg. Dabei sind auch mit sehr einfachen Befehlen viele kreative Umsetzungen möglich. Darauf aufbauend kann das Spiel „Pong“ programmiert werden, welches das erste kommerziell erfolgreiche Computerspiel war, das entwickelt wurde. Das Spiel simuliert ein Tischtennisspiel und besteht aus zwei Schlägern und einem Ball, der hin und her gespielt wird.

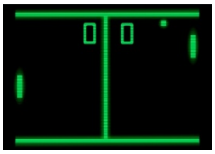
Das übergeordnete Lernziel besteht darin, dass die Schülerinnen und Schüler eine Vorstellung von der komplexen Arbeit eines Programmierers erhalten, und ein grundsätzliches Verständnis dafür entwickeln, wie der Computer auf Benutzerinput reagieren kann. Insofern ist es auch nicht entscheidend ob bzw. dass alle Jugendlichen bis zur vollständigen Programmierung von "Pong" fortschreiten. Die Unterrichtseinheit bietet viele Möglichkeiten zwischen unterschiedlichen Schwierigkeitsgraden und Lerntempi zu differenzieren. Zudem ist nach jeder Aufgabe auch ein Ausstieg möglich. Die Unterrichtseinheit erfüllt die Mehrzahl der Ziele auch wenn sie nicht vollständig durchgearbeitet wird. Sie können das Vorgehen somit beliebig Ihren Vorstellungen und Ihrer Klasse anpassen.

Lernziele

- Die Lernenden verstehen was ein Programm ist, und woraus ein Programm besteht.
- Die Lernenden lernen einige Grundbegriffe des Programmierens kennen.
- Die Lernenden sind in der Lage nach Anleitung einfache grafische Elemente zu programmieren (zeichnen) und ihre Farbe, Position und Bewegung auf dem Bildschirm zu steuern.
- Die Lernenden können die Bewegungen oder Farben der programmierten Objekte von der Steuerung mittels Maus oder Tastatur abhängig machen.
- Die Lernenden können ein einfaches Computerspiel (PONG) programmieren und anschliessend spielen.

Ablauf der Unterrichtseinheit; Rolle der Lehrperson

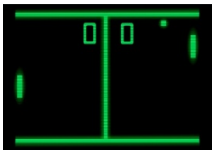
Die Unterrichtssequenz beginnt mit einem Einstieg (Video) zur Programmierung von Videospielen und ihrer Geschichte, gefolgt von einer aufbauenden Sequenz von Aufgaben, die in der Programmierung von "Pong" mündet. Den Schülerinnen und Schülern wird dabei immer zunächst das Prinzip des nächsten Schrittes erklärt. In der Folge können sie die dazu nötigen Befehle an einigen Beispielen einüben. Die Jugendlichen können somit, nach einer begleiteten Einführung des nächsten Schrittes, mehrheitlich selbständig arbeiten. In den meisten Fällen sind die Übungsbeispiele so konzipiert, dass langsamere Schüler nicht alle Aufgaben zu lösen brauchen.



Für die Unterrichtseinheit wird ein Computerraum benötigt. Idealerweise steht jedem Schüler / jeder Schülerin ein Computer zur Verfügung. Die vollständige Unterrichtseinheit dauert ca. 6 Lektionen, wobei an unterschiedlichen Stellen unterbrochen oder ausgestiegen werden kann.
Alles Weitere, z.B. Hinweise zu den Aufgaben und die Lösungen der Aufgaben (in rot) finden Sie in diesem Dokument. Die Aufgabenblätter für die Schülerinnen und Schüler finden Sie als fertige Kopiervorlage im Dokument *Schuelerunterlagen.docx*.

Wir bedanken uns für die finanzielle Unterstützung durch die Fachhochschule Nordwestschweiz, die Kabelwerke Brugg AG und das Paul Scherrer Institut und wünschen Ihnen einige interessante und lehrreiche Stunden.

Barbara Scheuner, Hochschule für Technik der FHNW
Matthias von Arx, Pädagogische Hochschule FHNW.



1. Vorarbeiten und Vorüberlegungen

1.1 Zur Programmierumgebung

Die Programmierumgebung Processing wurde entwickelt, um einfache interaktive Programme zu erstellen. Obwohl es sich hierbei nicht um eine Programmierumgebung handelt, welche kommerziellen Einsatz findet, wird innerhalb der Umgebung mit Java-Syntax gearbeitet. Somit ist es für interessierte Jugendliche kein Problem nachher auf „reales“ Java zu wechseln, wenn Interesse besteht. Zusätzlich ist die Umgebung frei verfügbar und einfach zu bedienen, da sie auf das nötigste reduziert wurde. Auf der Homepage von Processing (www.processing.org) sind zusätzlich viele Beispiele zu finden, welche gratis heruntergeladen werden können.
Hier sind die Vorteile der Programmierumgebung kurz summarisch angegeben:

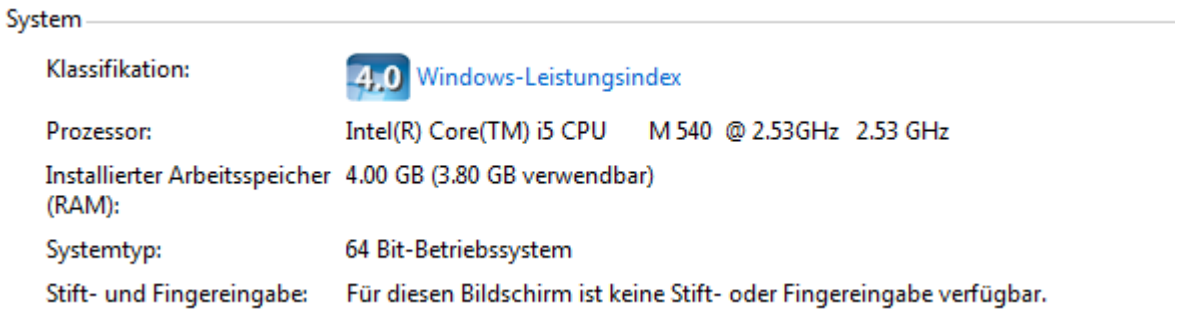
+ Einfache Installation
+ Einfache Bedienung
+ Erstellen von Programmen zur Ausführung durch Dritte ist möglich
+ Java-Syntax

Die Nachteile im Gegensatz zu einer „realen“ Java Umgebung sind:

– Kein Debugger, der bei der Suche nach Fehlern hilft
– Keine Autovervollständigung
– Verwalten von Projekten und mehreren Dateien ist schwierig
– Import von weiteren Paketen ist schwierig

1.2 Installation von Processing

Auf www.processing.org finden Sie unter "download processing" die verschiedenen verfügbaren Varianten. Zur Überprüfung der Bit-Version gehen Sie wie folgt vor:
Systemsteuerung > System
Dort können Sie den Systemtyp ablesen.

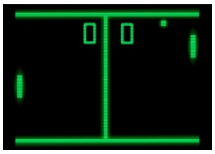


Processing muss nicht „installiert“ werden. Es startet auf einem Computer mit einer Java Version als Standalone (keine Administratorenrechte nötig).

Es empfiehlt sich, dass Sie den Download und die Installation von Processing (entsprechen dem Systemtyp) vorgängig selber vornehmen. Wenn Sie in einem Computerraum ca. 20 Installationen parallel vornehmen können, benötigen Sie dazu 1-2 Stunden. Wenn dann zum Unterrichtsbeginn alle PC's laufen und Processing bereits geöffnet ist, kann der Unterricht ohne Verzögerung starten. Bei allen weiteren Lektionen können Sie von dieser Vorarbeit profitieren.

1.3 Grundsätzlich Wissenswertes zur (Spiele)Programmierung

Mit einer Programmiersprache können dem Computer Befehle angegeben werden, welche er dann nacheinander abarbeitet. Dem Computer müssen dazu einerseits die richtigen Befehle (Syntax) und die



richtige Befehlsfolge (Semantik) übergeben werden, damit das Programm auch das Richtige tut. Ähnlich wie bei einer Sprache wie Deutsch. Die Befehle entsprechen hier den Worten, und die Befehlsfolge einem ganzen Satz.

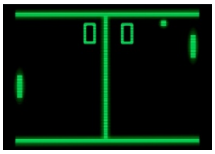
Beispiel:
Der Hnud sieplt mit dm Blal. *Falsche Syntax/ Falsche Befehle*
Der Ball spielt mit dem Hund. *Falsche Semantik/ Falsche Befehlsfolge obwohl die Syntax korrekt ist.*

Es ist eine anspruchsvolle Aufgabe, dem Computer die richtigen Befehle und die richtige Befehlsfolge zu übergeben, so dass das Gewünschte passiert. An diese Aufgabe sollen die SuS schrittweise herangeführt werden. Die Unterrichtseinheit ist wie folgt gegliedert:

1 **Einführung in das Gebiet der Spieleprogrammierung**. Die SuS sollen erkennen, dass alle Spiele, vom einfachen "Pong" bis hin zu den heutigen Spielen, aus einem Hintergrund und beweglichen Elementen (Spielfiguren) bestehen.
2 **Befehle in Zusammenhang mit Formen**: Szenen und Spielfiguren bestehen aus geometrischen Formen. Es werden hier einfache geometrische Formen und Kombinationen von Formen auf den Bildschirm gezeichnet. Die Formen können sich überlagern, was bedingt, dass man die Befehle in der richtigen Reihenfolge gibt. So können beliebige Objekte gezeichnet werden.
3 **Befehle in Zusammenhang mit Farben**: Im Gegensatz zum "Pong"-Spiel sind die Bildschirme heute farbig. Die geometrischen Formen können also auch unterschiedliche Farben haben.
4 **Zwischenspeicherung von Daten (Variablen):** Um sich in einem Spiel die Position einer Spielfigur (im Spiel) zu merken, braucht es eine Möglichkeit die zugehörigen Daten zwischen zu speichern.
5 **Reaktion auf Benutzerinput:** Damit man ein Spiel spielen kann, muss es auf den Spieler (der Mensch am Computer) reagieren können. Die Interaktion kann über die Maus oder die Tastatur geschehen.
6 **Bedingte Programmausführung:** Je nachdem, wo sich die Spielfiguren befinden, soll etwas anderes geschehen. Trifft der Schläger denn Ball nicht, soll das Spiel abbrechen, ansonsten soll der Ball zurückgespielt werden.
7 **Spieleprogrammierung**

1.4 Übersicht über die Programmierbefehle

Für eine Übersicht sind alle in der Unterrichtseinheit verwendeten Befehle hier vorab aufgeführt und beschrieben (alles entspricht grundsätzlich der Java-Syntax). Diejenigen Befehle, welche nicht alle SuS am Ende beherrschen sollten, sind grau hinterlegt. In den Aufgaben 1-7 werden diese Befehle schrittweise eingeführt und eingeübt. Möglicherweise ist es für Sie als Lehrperson sinnvoll, diese Übersicht griffbereit zu halten, ev. auch für die Lernenden projizieren zu können.



Generelle Befehle:

Befehl	Bedeutung	Beispiel
`size(x,y);`	Breite (x) und Höhe (y) des Fensters setzen	`size(200,400);`

`background(r,g,b);`	Setzt die Hintergrundfarbe auf die Werte r für Rot, g für Grün und b für Blau (alle Werte liegen zwischen 0 und 255)	`//Blau`
		`background(0,0,255);`
		`//Rot`
		`background(255,0,0);`

Die Methoden von Processing, in denen der Code steht:

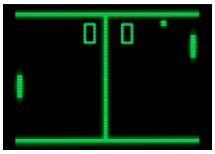
Methode	Bedeutung

`void setup(){`
`  ...`
`}`
	In dieser Methode stehen alle Befehle, welche nur einmal ausgeführt werden sollen. Das sind typischerweise:
	• Setzen der Fenstergrösse
	• Setzten der Hintergrundfarbe
	• Zeichnen des Hintergrunds

`void draw(){`
`  ...`
`}`
	In dieser Methode stehen alle Befehle, welche immer wieder ausgeführt werden sollen. Das sind:
	• Überprüfen von Programmzuständen wie: Maus gedrückt und Taste gedrückt
	• Zeichnen von Objekten an der Mausposition

Zum Zeichnen von Figuren:

Befehl	Bedeutung	Beispiel
`point(x,y)`	Zeichnet einen Punkt an der Position x,y	`point(30,20);`
`line(x1,y1,x2,y2)`	Zeichnet eine Line vom Punkt x1,y1 zum Punkt x2,y2	`line(10,20,90,10);`
`rect(x,y,b,h)`	Zeichnet ausgehend vom Punkt x,y ein Rechteck mit der Breite b und der Höhe h	`rect(20,10,300,200);`
`ellipse(x,y,b,h)`	Zeichnet eine Ellipse um den Punkt x,y mit der Höhe h und der Breite b.	`ellipse(50,70,30,30);`



Zum Setzten von Farben:

Befehl	Bedeutung	Beispiel

`background(r,g,b);`	Setzt die Hintergrundfarbe auf die Werte r für Rot, g für Grün und b für Blau (alle Werte liegen zwischen 0 und 255)	`//Blau`
		`background(0,0,255);`
		`//Rot`
		`background(255,0,0);`

`fill(r,g,b);`	Setzt die Füllfarbe für die Objekte auf die Werte r für Rot, g für Grün und b für Blau (alle Werte liegen zwischen 0 und 255)	`//Schwarz`
		`fill(255,255,255)`
		`//Weiss`
		`fill(0,0,0);`

Variablen aus Processing

Befehl	Bedeutung
`height`	Höhe des Fensters
`width`	Breite des Fensters
`mouseX`	Position der Maus auf der x-Achse
`mouseY`	Position der Maus auf der y-Achse
`mousePressed`	True wenn die Maus gedrückt ist und false, wenn sie nicht gedrückt ist.
`key`	Hier ist der Buchstabe gespeichert, der als letztes auf der Tastatur gedrückt wurde.

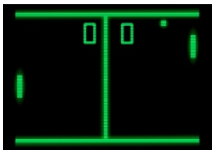
Eigene Variablen definieren (für das Pong Spiel)

Befehl	Bedeutung
`int x=0;`	Erstellt eine ganzzahlige Variable x, und weist ihr den Wert 0 zu.
`x=x+20`	Nimmt den Wert von x, addiert 20 und speichert den Wert wieder in x. Vergleichbar mit einem Dokument das geöffnet, verändert und dann wieder unter dem gleichen Namen gespeichert wird.

Verzweigung / Bedingte Programmausführung

Befehl	Bedeutung

`if ( Bedingung ){`
`. . .`
`}`
	Testet, ob die Bedingung zutrifft. Wenn dies der Fall ist, werden die Befehle in den Klammern ausgeführt. Ansonsten passiert nichts.

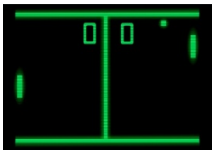


Operatoren zum Vergleichen von Werten

Symbol	Beschreibung	Beispiel
<	Testet, ob das links kleiner ist als das rechts	`mouseX<200`
>	Testet, ob das links grösser ist als das rechts	`mouseX>200`
<=	Testet, ob das links kleiner oder gleich ist wie das rechts	`mouseX<=200`
>=	Testet, ob das links grösser oder gleich ist wie das rechts	`mouseX<=200`
==	Testet, ob das links gleich dem rechts ist	`mouseX==200`

Operatoren zum Verknüpfen von Bedingungen

Symbol	Beschreibung	Beispiel
&	Testet, ob das links und das rechts zutrifft	`(mousePressed) & (mouseX<10)`
|	Testet, ob das links oder das rechts zutrifft	`(mousePressed) | (mouseX<10)`



2. Die Unterrichtssequenz und ihre Aufgaben

2.1 Einführung (Aufgabenblatt 1)

Bei der Einführung geht es darum herauszufinden, was ein Computerspiel ausmacht. In der Diskussion und mit Hilfe geeigneter Rückfragen der Lehrperson soll die Komplexität der heutigen Computerspiele auf drei einfache Elemente heruntergebrochen werden:

- **Szene**: Als Szene kann der Hintergrund des Spiels bezeichnet werden. Das ist diejenige Welt, welche sich nicht verändert und einfach den Rahmen des Spiels bildet.
- **Spielfiguren**: Das sind die beweglichen, durch den Benutzer steuerbaren Elemente des Spiels, wie die eigene Spielfigur und sowie alle Gegner.
- **Interaktion**: Die Interaktion mit dem Spiel kann über die Maus, die Tastatur oder eine beliebige andere Steuerungseinheit gemacht werden.

Als Einstieg eignet sich das Video http://www.youtube.com/watch?v=4_cBBoZW3JI (die ersten 6:05 Minuten). Hier wird kurz historisch auf das Aufkommen von Computer-Spielen und ihre kulturelle Bedeutung eingegangen.

Aufgabenblatt 1: Eigenschaften von Video-/Computerspielen

Sie haben sicher schon einmal ein Computerspiel oder ein Videospiel gespielt. Heute gibt es ganz unterschiedliche Spiele. Hier sollen Sie sich überlegen, was denn ein Spiel ausmacht.
Aufgabe 1: Was für Spiele kennen Sie. Geben Sie hier möglichst unterschiedliche Spiele an.

Aufgabe 2: Wie kann mit den Spielfiguren interagiert werden (in Kontakt getreten werden)?

- Maus
- Tastatur
- . . .

Aufgabe 3: Was haben unterschiedliche Spiele gemeinsam? Worin unterscheiden sie sich? Aus was für Grundelementen bestehen die Spiele?

- Hintergrund (Spielumgebung)
- Bewegliche Elemente: Spielfiguren, Gegner

2.2 Geometrische Figuren auf dem Bildschirm zeichnen (Aufgabenblatt 2)

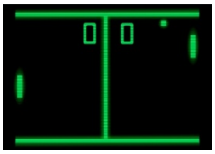
Im zweiten Teil geht es um das Darstellen von geometrischen Figuren auf dem Bildschirm. Aus diesen geometrischen Figuren kann alles auf dem Bildschirm dargestellt werden. Im „schlimmsten" Fall muss die Figur aus lauter Punkten (Pixeln) zusammengesetzt werden. In dieser Aufgabe geht es darum ein Verständnis für das Koordinatensystem des Computers zu entwickeln.

HINWEISE:

- Die zu zeichnenden Figuren überlagern sich (siehe Bild 1 von Aufgabenblatt 2). Somit spielt die Reihenfolge in der gezeichnet wird eine grosse Rolle.
- Die Anweisungen zum Zeichnen der einzelnen Elemente wie Kreise, Rechtecke oder Linien sollten gemäss Darstellungen in der Übersichtstabelle von Aufgabenblatt 2 an der, vorzugsweise karierten, Tafel erklärt werden.

Aufgabenblatt 2: geometrische Figuren

Ein Merkmal von Computerspielen ist, dass sie in einer bestimmten Welt integriert sind. Dabei kann es sich um ein Spielbrett, wie man es auch von „realen" Spielen kennt oder eine virtuelle Umgebung han-



deln. In dieser Aufgabe sollen Sie Elemente kennen lernen, mit denen solche Umgebungen und Spielfiguren auf dem Computer gezeichnet werden können.

Aufgabe 1: Bilder aus Einzelteilen erzeugen

Überlegen Sie sich, wie Sie die folgenden Bilder erstellen können. Dabei dürfen Sie nur die folgenden geometrischen Objekte verwenden:

- Punkte
- Linien
- Rechtecke
- Ellipsen (Kreise)

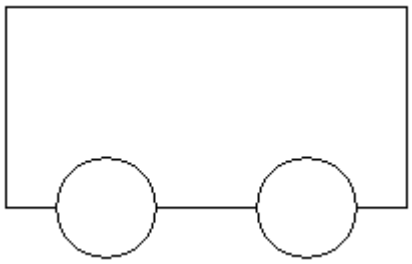


Bild 1

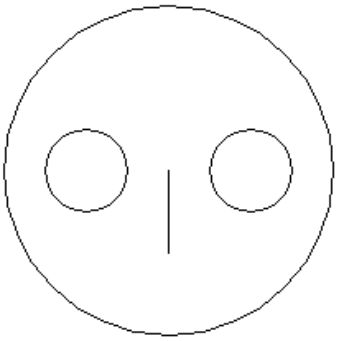


Bild 2

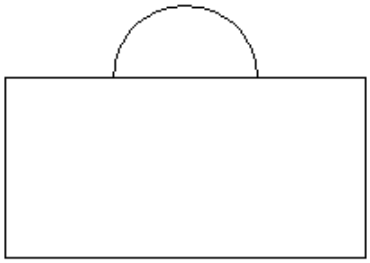


Bild 3

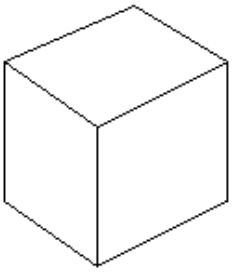
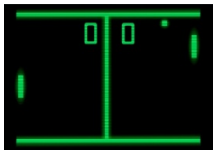
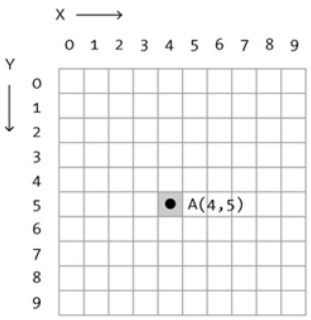
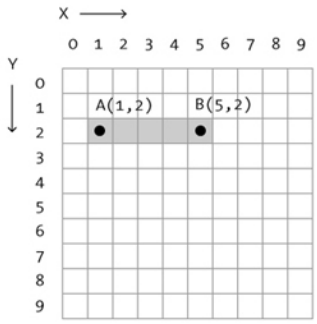
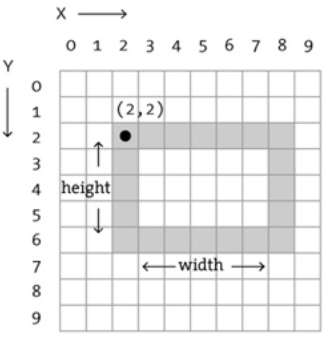
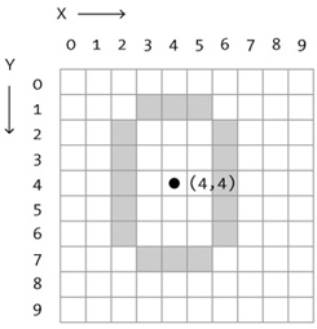
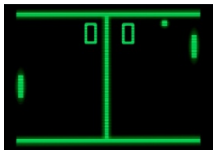


Bild 4

Die Anweisungen für den Computer zum Zeichnen von geometrischen Objekten sind in der folgenden Tabelle zusammengefasst.



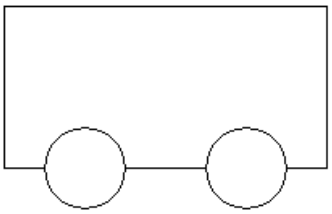
	Beispiel
Ein **Punkt** braucht zwei Angaben, eine x- und eine y-Position. Ausgehend von der oberen linken Ecke wird dann genau der Pixel ausgefüllt, der an dieser Position liegt. In Programmiersprache heisst das `point(4,5)`	
Eine **Linie** besteht aus einem Anfangs und einem Endpunkt. Diese Punkte haben jeweils eine x- und eine y-Koordinate. Im Vergleich zum Punkt braucht es also 4 Werte, damit der Computer weiss, wie die Line aussieht. Im Beispiel haben wir: `line(1,2,5,2)` Die roten Werte bezeichnen die Koordinaten des ersten, und die grünen Werte die Koordinaten des zweiten Punktes.	
Ein **Rechteck** hat eine Position (obere/linke Ecke) und eine Breite (engl. width) und eine Höhe (engl. height). Auch um ein Rechteck zu zeichnen braucht der Computer vier Werte. Zeichnen des Beispiels: `rect(2,2,7,5)` Die roten Werte bezeichnen die obere/linke Ecke. Der grüne Wert gibt an, wie viele Pixel das Rechteck breit ist, und der blaue wie viele Pixel das Rechteck hoch ist. Merke: Sind Höhe und Breite gleich, ist es ein **Quadrat**.	
Eine **Ellipse** hat einen Mittelpunkt und eine Breite (engl. width) und eine Höhe (engl. height). Zeichnen des Beispiels: `ellipse(4,4,7,5)` *Die roten Werte bezeichnen den Mittelpunkt. Der grüne Wert gibt an, wie viele Pixel die Ellipse breit ist, und der blaue wie viele Pixel die Ellipse hoch ist.* *Merke: Sind Höhe und Breite gleich, ist es ein **Kreis**.*	

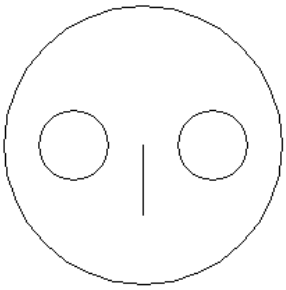


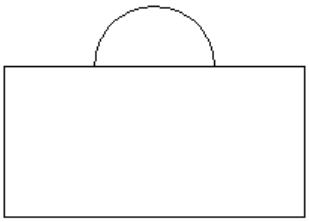
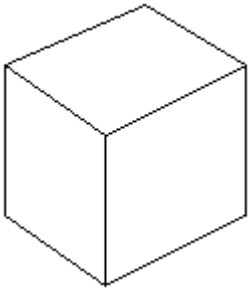
Aufgabe 2: In Computersprache übersetzen

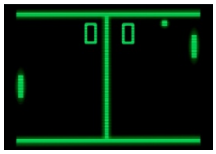
Überlegen Sie sich, wie Sie die Objekte aus Aufgabe 2.1 so formulieren können, dass es der Computer auch versteht.

Bild	Code
	`ellipse(100,100,150,50);`

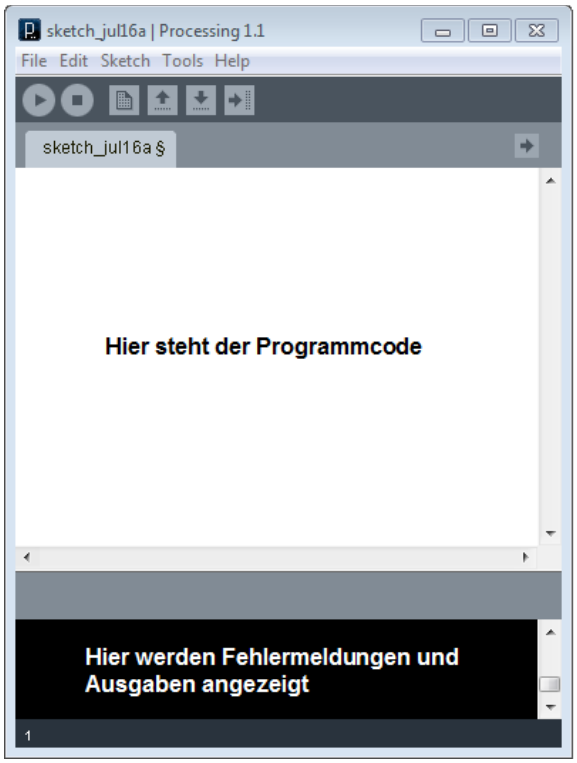
	`size(400,400);`
	`background(255);`
	`rect(100,100,200,100);`
	`ellipse(150,200,50,50);`
	`ellipse(250,200,50,50);`

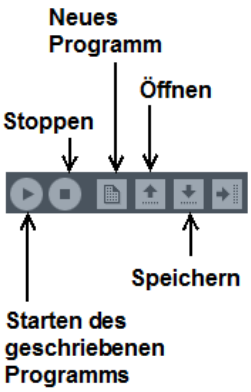
	`size(400,400);`
	`background(255);`
	`ellipse(200,200,200,200);`
	`ellipse(150,200,50,50);`
	`ellipse(250,200,50,50);`
	`line(200,210,200,260);`

	*`size(400,400);`* *`ellipse(200,100,80,80);`* *`rect(100,100,200,100);`* *Bemerkung: Wenn zuerst das rect und dann der Kreis gezeichnet werden, dann stimmt das Bild nicht.*
	*`size(400,400);`* *`line(100,130,170,100);`* *`line(100,130,100,205);`* *`line(100,130,150,165);`* *`line(220,130,170,100);`* *`line(220,130,150,165);`* *`line(220,130,220,205);`* *`line(150,240,100,205);`* *`line(150,240,150,165);`* *`line(150,240,220,205);`*



Nun wollen wir unser erstes Programm schreiben. Das Programm mit dem wir dies tun, heisst Processing. Wenn das Programm nicht bereits geöffnet ist, wird es einfach durch Doppelklick auf processing.exe gestartet.





Aufgabe 3: Erstes Programm schreiben

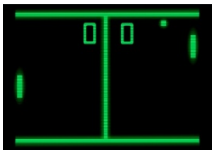
Geben Sie im Feld, in dem der Programmcode stehen soll folgendes ein und starten Sie das Programm mit dem Pfeilsymbol:

```
void setup(){
  size(400,400);
  rect(20,40,100,50);
}
```

Zwischen den geschweiften Klammern { } stehen die Anweisungen an den Computer welche unsere „Szene“ zeichnen sollen. Als erstes wird hier jeweils noch angegeben, wie gross die Spielfläche (size) sein soll. In diesem Fall ist es eine Fläche der Grösse 400 Pixel auf 400 Pixel.

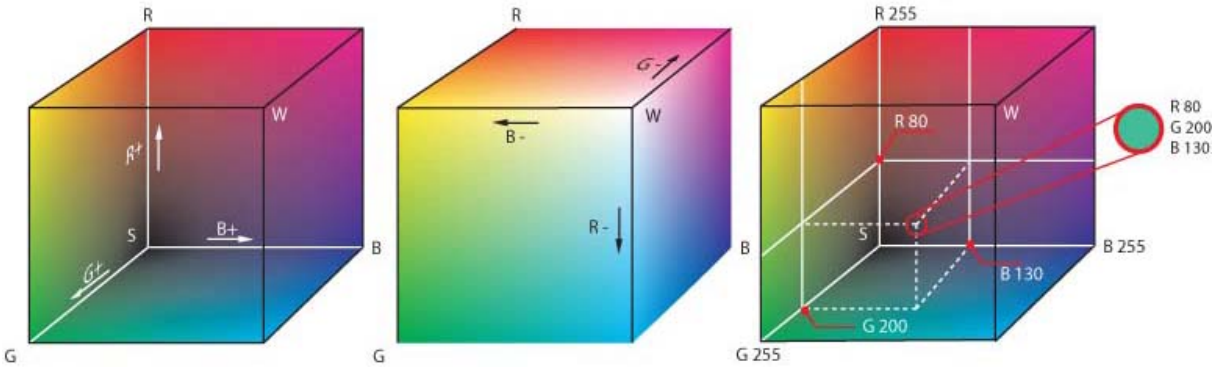
Aufgabe 4: Eigene Objekte zeichnen

Ersetzen Sie nun den Befehl rect(20,40,100,50) durch die Anweisungen, welche Sie in Aufgabe 2 aufgeschrieben haben.



2.3 Farbdarstellung mit dem Computer (Aufgabenblatt 3)

Die Elemente sind jetzt erst schwarz und weiss. Der Computer kann aber offensichtlich auch mit Farben arbeiten. Im Aufgabenblatt 3 geht es deshalb um die Verwendung von Farben. Die Farben basieren auf dem RGB-Farbraum (http://de.wikipedia.org/wiki/RGB-Farbraum).



Die Farbe wird hierbei aus den Grundfarben **R**ot **G**rün und **B**lau zusammengesetzt. Die Abwesenheit von Farbe entspricht den Werten 0,0,0 und ist somit Schwarz. Alle Farben zusammen in der maximalen Intensität entsprechen 255,255,255 und sind somit Weiss.
Die einzelnen Farbkomponenten können 256 verschiedene Werte annehmen (0 bis 255). Dies entspricht 8 Bit Speicherplatz, denn ($256 = 2^8$). Somit braucht eine ganze RGB-Farbe 24 Bit.

HINWEISE:

- Falls Sie sich mit Bitoperationen etc. auskennen, können Sie hier auch dieses Thema noch ein bisschen vertiefen.
- Zur Aufgabe 2: Verlangt werden soll das zeichnen der ersten Figur. Sobald alle SuS damit fertig sind, können Sie weiterfahren. Die zweite Abbildung ist zur Beschäftigung der schnelleren Schülerinnen und Schüler gedacht. Je nach Interessenlage und Schulstufe kann das Zeichnen farbiger Objekte oder komplizierterer Muster durchaus weitergeführt werden. Das ist vom konzeptionellen Verständnis her weniger anspruchsvoll als später folgende Aufgaben, hat aber bezüglich Ästhetik und Kreativität durchaus seinen Reiz.
- Der Befehl fill(x,y,z) steht immer **vor** dem Objekt, welches entsprechend gefüllt werden soll.

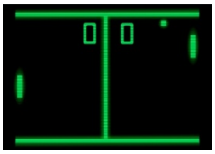
Aufgabenblatt 3: Farben

Bis hier waren unsere Bilder nur aus den Farben Grau, Weiss und Schwarz. In dieser Aufgabe soll nun Farbe in unsere Szene kommen.

Aufgabe 1: Die Hintergrundfarbe

Die Hintergrundfarbe kann mit dem Befehl `background()` gesetzt werden. Diese Anweisung benötigt drei Werte. Der erste Wert gibt an, wie gross der Rot-Anteil ist, der zweite Wert gibt an wie gross der Grün-Anteil ist und der dritte Wert gibt an, wie gross der Blau-Anteil ist. Diese Farben werden auch RGB-Farben genannt (Rot-Grün-Blau-Farben). Die Werte liegen bei allen drei Farben zwischen 0 (nicht vorhanden) und 255 (so viel wie möglich).
Nehmen Sie das folgende Programm als Ausgangslage:

```
void setup(){
  size(400,400);
  background(0,0,0);
}
```



Füllen Sie nun die folgende Tabelle aus. Das heisst, geben Sie an, welche Hintergrundfarbe bei den gegebenen Farben erscheint. Ersetzen Sie dazu die drei 0 Werte im Befehl `background` mit den Werten der Tabelle.

Rot	Grün	Blau	Farbe ?
0	0	0	*schwarz*
255	255	255	*weiss*
255	0	0	*rot*
0	255	0	*grün*
0	0	255	*blau*
255	255	0	*gelb*
0	255	255	*Hellblau/türkis*
255	0	255	*violett*
120	0	0	*dunkelrot*
100	255	255	*hellblau*
255	150	150	*hellrosa*

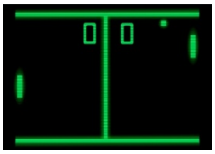
Natürlich können auch geometrische Formen farbig gezeichnet werden.
Mit der Anweisung fill() kann die Farbe geändert werden, mit der die gezeichneten Objekte ausgefüllt werden. Auch hier werden wieder die drei Werte für Rot, Grün und Blau angegeben.
Beispiel:

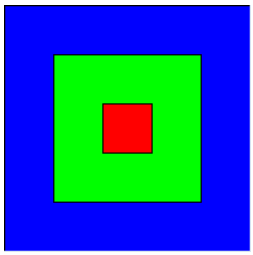
```
void setup(){
   size(400,400);
   fill(0,0,0);
   rect(100,100,200,200);
}
```

Welche Farbe hat das gezeichnete Quadrat? schwarz

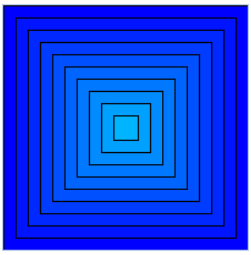
Aufgabe 2: Farbige Figuren Zeichnen

Zeichnen Sie die folgenden zwei Figuren und schreiben Sie den Code in die Tabelle:





```
void setup(){
   size(300,300);
   fill(0,0,255);
   rect(0,0,300,300);
   fill(0,255,0);
   rect(50,50,200,200);
   fill(255,0,0);
   rect(100,100,100,100);
}
```



```
  size(300,300);
  fill(0,0,255);
  rect(0,0,300,300);
// Die Rot und Grün Anteile werden um 30 erhöht
// Das Quadrat hat um 40 kürzere Seiten, und der
// Startpunkt verschiebt sich um 20.
  fill(30,30,255);
  rect(20,20,260,260);
  fill(60,60,255);
  rect(40,40,220,220);
  fill(90,90,255);
  rect(60,60,180,180);
  fill(120,120,255);
  rect(80,80,140,140);
  fill(150,150,255);
  rect(100,100,100,100);
  fill(180,180,255);
  rect(120,120,60,60);
  fill(200,200,255);
  rect(140,140,20,20);
```

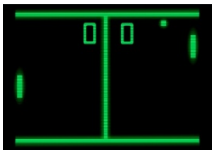
Hinweis (freiwillig): mit dem Befehl noStroke() werden die schwarzen Ränder der Rechtecke nicht mehr gezeichnet. Mit dem Befehl stroke() wird analog zum Befehl fill() die Farbe für die Ränder der Objekte gesetzt.

2.4 Variablen: Speichern von Daten zur Wiederverwendung (Aufgabenblatt 4)

Variablen sind wichtige Bestandteile von Programmen. Sie dienen dazu, Zustände (Daten) des Programms zu speichern und später wieder zu verwenden. Ein Programm kann beschrieben werden als:

Programm= Algorithmus + Daten

Die Daten sind dabei das, was durch den Algorithmus (eine Art Prozessablauf) bearbeitet wird und schlussendlich zu einem Resultat führen. Beispiele für Daten und Algorithmen sind:



Daten	Algorithmus	Anwendungsbereich
Zahlen, Buchstaben, Wörter	Sortieralgorithmus, Suchalgorithmus	Excel, Word
Kartenmaterial	Kürzeste Wege	SBB-Fahrplan, Navigationsprogramm

Variablen zu verstehen ist für einige Schülerinnen und Schüler ganz einfach, andere haben Mühe mit dem Konzept. Vor allem, wenn eine Variable mit einem neuen Wert überschrieben wird, wobei der neue Wert aus dem alten Wert berechnet wird.

HINWEISE:

- Am besten greifen Sie als Lehrperson an dieser Stelle strukturierend ein. Allerdings nicht indem Sie längere Erklärungen abgeben, sondern indem Sie sicherstellen, dass alle Lernenden die nächsten 1-2 Schritte gleichzeitig ausführen und dann gemeinsam besprechen.
- Lassen Sie die Schülerinnen und Schüler ohne grosse Vorbemerkungen das simple Programm mit dem Befehl `line(0,0,width, height)` zeichnen und erklären Sie ihnen die deutsche Bedeutung der Worte width und height. Lassen Sie die Lernenden das Ergebnis erklären und begründen.
- Gehen Sie dann mit der Programmzeile `ellipse(width/2, height/2, 2,50,50)` analog vor.
- Die meisten Lernenden verstehen dann die Bedeutung eines Wertes bzw. einer Variable sehr schnell.

Aufgabenblatt 4: Variablen

In allen Computerspielen können sich die Zustände (Werte) des Hintergrunds, der Spielfiguren oder des Spielers verändern. Diese Zustände müssen irgendwo gespeichert werden. Z.B. möchte man sich merken, wo sich der Spielball, der Gegner oder das Ziel gerade befindet. Dazu werden Variablen benötigt.

Werte in Processing

In Processing gibt es einige Schlüsselwörter, welche für Werte stehen. Einige Beispiele dafür sind:

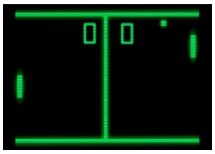
- height: Höhe des Fensters
- width: Breite des Fensters
- mouseX: x-Koordinate der Maus
- mouseY: y Koordinate der Maus

Soll nun eine Linie von links-oben nach rechts-unten gezeichnet werden, kann das auch geschrieben werden als:

```
line(0,0,width,height);
```

Mit diesen Werten kann auch gerechnet werden:

Code	Bedeutung
+	Addition
-	Subtraktion
/	Division (ganzzahlig)
*	Multiplikation

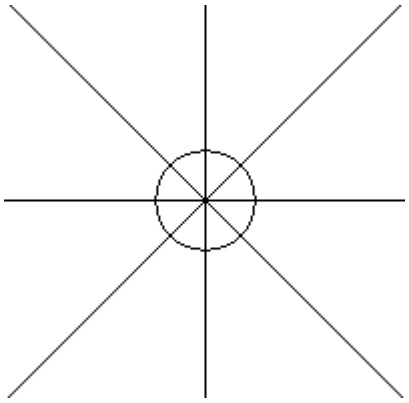


Soll also ein Kreis in der Mitte des Fensters gezeichnet werden, sieht das so aus:

```
ellipse(width/2,height/2,50,50);
```

Aufgabe 1:

Zeichnen Sie mit Hilfe von Rechenoperationen und den genannten Schlüsselwörtern das folgende Bild.



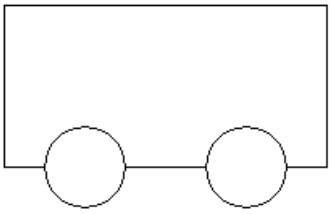
```
size(. . .,. . .);
 background(255);
 ellipse(width/2, height/2, width/4, height/4);
 line(0,0,width, height);
 line(width, 0, 0, height);
 line(width/2,0, width/2, height);
 line(0,height/2,width, height/2);
```

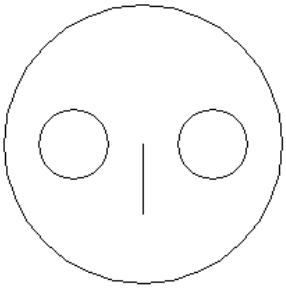
Das Bild soll immer gleich gezeichnet werden, auch wenn die Werte in size() verändert werden.

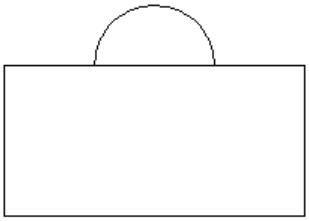
Aufgabe 2:

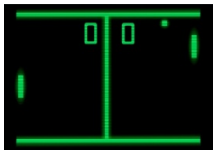
Zeichnen Sie mindestens eines der Bilder aus Aufgabenblatt 2 so, dass sie immer in der Mitte des Fensters sind, egal was für eine Grösse für das Fenster am Anfang gewählt wird.

Bild	Code

	`rect(width/2-100,height/2-50,200,100);`
	`ellipse(width/2-50,height/2+50,50,50);`
	`ellipse(width/2+50,height/2+50,50,50);`

	`size(400,400);`
	`background(255);`
	`ellipse(width/2,height/2,200,200);`
	`ellipse(width/2-50,height/2,50,50);`
	`ellipse(width/2+50,height/2,50,50);`
	`line(width/2,height/2+10,width/2,height/2+60);`

	`size(400,400);`
	`ellipse(width/2,height/2-50,80,80);`
	`rect(width/2-100,height/2-50,200,100);`



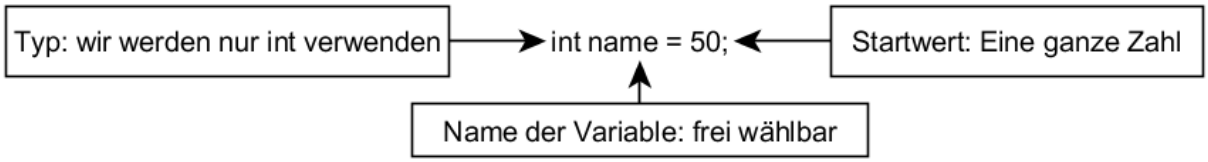
Testen der Zentrierung

Nun können Sie Ihren Code testen indem Sie den Code für das Zeichnen im folgenden Programm an der markierten Stelle einfügen.

```
void setup(){
  size(400,400);
// bedeutet, dass das Fenster eine veränderbare Grösse hat.
  frame.setResizable(true);
}
void draw(){
  // Zeichencode
}
```

Eigene Variablen (für die etwas schnelleren SuS, wird für "Pong" gebraucht)

Deklaration von Variablen:



Aufgabe 3: Erste Variable

In dieser Aufgabe werden wir eine Variable definieren und deren Wert anschliessend zum Zeichnen verwenden. Die Definition der Variable muss ganz am Anfang des Programms stehen. Die Variable mit dem Namen pos wird auf den Startwert 100 gesetzt. Anschliessend wird dieser Wert an vier Stellen im Programm verwendet. Bei der Verwendung kann dem Wert auch etwas hinzugezählt oder abgezogen werden.

```
int pos=100;
void setup(){
  size(400,400);
  background(255,255,255);
  ellipse(pos, 100, 100, 100);
  ellipse(pos+10, 100, 100, 100);
  ellipse(pos+20, 100, 100, 100);
  ellipse(pos+30, 100, 100, 100);
}
```

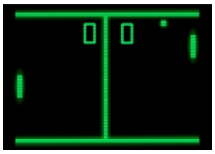
Kopieren Sie den obigen Code, testen Sie ihn und machen Sie anschliessend folgende Änderungen:

a. Ändern Sie die Variable pos so, dass der Mittelpunkt des ersten Kreises nicht an der Position 100/100 ist, sondern an der Position 50/100.

```
int pos=50;
```

b. Ändern Sie das Programm so ab, dass sich die Kreise weniger überlappen.

```
ellipse(pos, 100, 100, 100);
ellipse(pos+50, 100, 100, 100);
ellipse(pos+100, 100, 100, 100);
ellipse(pos+150, 100, 100, 100);
```



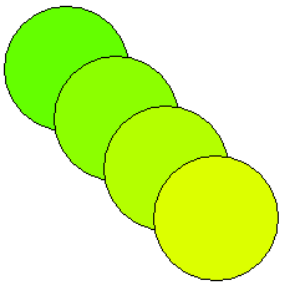
c. Verwenden Sie die Variable pos auch noch für das Setzen bzw. Veränderung der Farbe.

```
fill(0,pos,0);
ellipse(pos, pos, 100, 100);
fill(0,pos+30,0);
ellipse(pos+50, 100, 100, 100);
fill(0,pos+60,0);
ellipse(pos+100, 100, 100, 100);
fill(0,pos+90,0);
ellipse(pos+150, 100, 100, 100);
```

d. Der Kreis soll diagonal wandern. Ändern Sie den Code entsprechend.

```
fill(0,pos,0);
ellipse(pos, pos, 100, 100);
fill(0,pos+30,0);
ellipse(pos+50, pos+50, 100, 100);
fill(0,pos+60,0);
ellipse(pos+100, pos+100, 100, 100);
fill(0,pos+90,0);
ellipse(pos+150, pos+150, 100, 100);
```

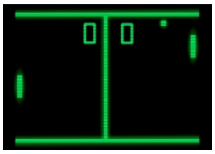
Das Resultat könnte am Ende z.B. so aussehen:



Aufgabe 4: Variable verändern

In der Aufgabe 3 wurde der am Anfang in der Variablen gespeicherte Wert immer wieder verwendet, aber nicht verändert. Der folgende Code macht genau das gleiche wie der gegebene Code aus Aufgabe 3. Nun wird der Wert von pos aber jeweils verändert.

```
int pos= 100;
void setup(){
  size(400,400);
  background(255,255,255);
  ellipse(pos, 100, 100, 100);   <- Wert von pos = 100
  pos=pos+10;
  ellipse(pos, 100, 100, 100);   <- Wert von pos = 110
  pos=pos+10;
  ellipse(pos, 100, 100, 100);   <- Wert von pos = 120
  pos=pos+10;
  ellipse(pos, 100, 100, 100);   <- Wert von pos = 130
}
```



Kopieren Sie den obigen Code, testen Sie ihn und machen Sie anschliessend wieder folgende Änderungen:

a. Ändern Sie die Variable pos so, dass der Mittelpunkt des ersten Kreises nicht an der Position 100/100 ist, sondern an der Position 50/100.
   Wie oben
b. Ändern Sie das Programm so ab, dass sich die Kreise weniger überlappen.
   pos=pos+10; wird zu pos=pos+50;
c. Verwenden Sie die Variable pos auch noch für das Setzen bzw. Veränderung der Farbe.

```
pos=pos+10;
fill(0,pos,0) ;
ellipse(pos, 100, 100, 100);
```

d. Der Kreis soll diagonal wandern. Ändern Sie den Code entsprechend.

```
pos=pos+10;
fill(0,pos,0) ;
ellipse(pos, pos, 100, 100);
```

Welche Variante bevorzugen Sie? Die Version von Aufgabe 3 oder von Aufgabe 4?
Variante 4 hat den Vorteil, dass man die gleichen drei Zeilen endlos aneinanderreihen kann und das Gewünschte passiert. Man muss nicht selber rechnen.

2.5. Interaktion: Reagieren auf Benutzerinput (Arbeitsblatt 5)

Bis jetzt haben die Lernenden unbewegliche Bilder gezeichnet. Processing bietet jedoch eine ganz einfache (viel einfachere als in anderen Programmierumgebungen) Möglichkeit an, auf Input des Benutzers zu reagieren. Dazu bietet Processing die internen Variablen `mouseX` und `mouseY` an. Diese besitzen den Wert der x bzw. y-Koordinate der Maus. Wenn sich die Maus auf dem Programmfeld bewegt, werden diese Werte intern immer aktualisiert. Damit kann schon mit wenig Programmcode ein kleines Zeichenprogramm geschrieben werden.

Aufgabenblatt 5: Interaktion

Bis jetzt haben wir einfach statische Objekte gezeichnet. In einem Computerspiel gibt es aber immer auch Elemente, welche sich bewegen, seien dies Spielfiguren oder Gegner. In diesem Aufgabenblatt geht es nun um die Interaktion mit dem Programm. Das Programm soll einerseits auf die Position der Maus,andererseits auf Tasten reagieren.

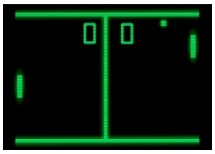
Aufgabe 1: Verändern der Hintergrundfarbe abhängig von der Mausposition

Wir wollen mit einem neuen Basisprogramm starten:

```
void setup(){
  size(400,400);
}
void draw(){
  background(255,255,255);
}
```

a) Übernehmen Sie dieses Programm und starten Sie es.
b) Ändern Sie das Programm nun wie folgt ab:

```
void setup(){
```



```
  size(400,400);
}
void draw(){
  background(mouseX,mouseY,255);
}
```

Starten Sie das Programm erneut und beobachten Sie was passiert. Wie können Sie sich das erklären?

Die Hintergrundfarbe ändert sich wenn man mit der Maus über das Fenster fährt. Erklärung: Für den rot-Wert wird die x-Koordinate der Maus verwendet, für den grün-Wert die y-Koordinate?

Aufgabe 2: Zeichnen einer Figur an der Mausposition

Übernehmen Sie nun das Programm:

```
void setup(){
  size(400,400);
  background(255,255,255);
}
void draw(){
  fill(0,0,0);
  ellipse(mouseX,mouseY,10,10);
}
```

Alles was fett markiert ist, wird nun von Processing immer wieder aufgerufen. Die Variablen mouseX und mouseY sind wie height und width Variablen von Processing. Diese beiden bezeichnen die Position der Maus auf dem Bildschirm.

a) Ändern Sie das Programm so ab, dass anstelle der Ellipse ein Rechteck an der Position der Maus gezeichnet wird.

```
ellipse(mouseX,mouseY,10,10); ersetzten durch
rect(mouseX,mouseY,10,10);
```

b) Ändern Sie das Programm so ab, dass anstelle der Ellipse ein Punkt gezeichnet wird.

```
ellipse(mouseX,mouseY,10,10);
ersetzten durch
point(mouseX,mouseY);
```

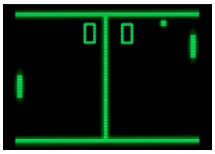
Zusatzaufgaben (freiwillig)

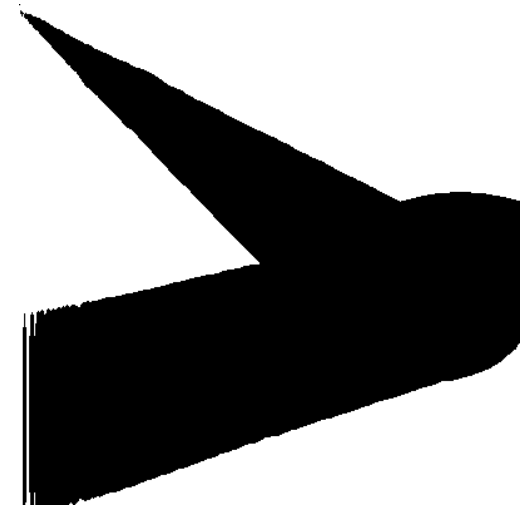
c) Ändern Sie das Programm so ab, dass zwischen der Mausposition und der oberen linken Ecke eine Linie gezeichnet wird.

```
ellipse(mouseX,mouseY,10,10);
ersetzten durch
line(0,0,mouseX,mouseY);
```

d) Überlegen Sie, wie Sie die Grösse des zu zeichnenden Objekts von der Position der Maus abhängig machen können. Erstellen Sie dazu ein Programm.

```
ellipse(mouseX,mouseY,10,10);
ersetzten durch
ellipse(mouseX, mouseY, mouseX/2, mouseY/2);
```





```
rect(mouseX, mouseY, mouseX/4, mouseX/4);
```

2.6.Verzweigungen: Bedingte Programmausführung (Aufgabenblatt 6)

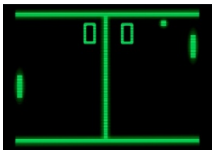
Um kleine Spiele programmieren zu können braucht es als letztes Programmierelement noch die sogenannte Verzweigung. Bei der bedingten Programmausführung wird mit if-Verzweigungen gearbeitet.
Beispiel:
Wenn … dann …
Wenn es regnet, dann gehe ich ins Museum.

Aufgabenblatt 6: Verzweigungen (Bedingte Programmausführung)

Aufgabe 1: Nur zeichnen, wenn die Maus gedrückt ist

Wir starten mit dem letzten Programm aus Aufgabenblatt 5.

```
void setup(){
  size(400,400);
  background(255,255,255);
}
void draw(){
  fill(0,0,0);
  ellipse(mouseX,mouseY,10,10);
}
```



Der Kreis soll hier nur gezeichnet werden, wenn die Maus gedrückt ist. Zum Testen, ob die Maus gedrückt ist, wird die Befehlsfolge `if (test){…}` verwendet.
Ersetzen Sie nun die Zeile `ellipse(mouseX, mouseY, 10,10)` durch die Zeilen:

```
if (mousePressed){
  ellipse(mouseX, mouseY, 10,10);
}
```

Starten Sie Ihr Programm.
Das Programm zeichnet jetzt nur, wenn die Maus gedrückt wird (engl. pressed = gedrückt).

Aufgabe 2: Nur im oberen Teil des Feldes zeichnen

Man kann natürlich auch mehr testen, als nur, ob die Maus gedrückt ist oder nicht. Hier sollen Sie nur noch einen Teil der Fläche für das Zeichnen zulassen.

```
void setup(){
  size(400,400);
  background(255,255,255);
}
void draw(){
  if (mouseX > 200){
     ellipse(mouseX, mouseY, 10,10);
  }
}
```

Ersetzen Sie den mousPressed durch mouseX>200. Was macht das Programm?

- Ändern Sie das Programm so ab, dass nur im unteren Bereich des Feldes gezeichnet werden kann.
  if (mouseY > height/2) {…}
- Ändern Sie das Programm so ab, dass nur links oder rechts gezeichnet werden kann.
  if (mouseX > width/2) {…}
  if (mouseX < width/2) {…}

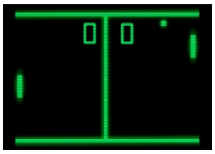
Verwenden Sie dazu die folgenden Operationen:

Symbol	Beschreibung	Beispiel
<	Testet, ob das links kleiner ist als das rechts	mouseX<200
>	Testet, ob das links grösser ist als das rechts	mouseX>200
<=	Testet, ob das links kleiner oder gleich ist wie das rechts	mouseX<=200
>=	Testet, ob das links grösser oder gleich ist wie das rechts	mouseX<=200

Aufgabe 3 (für die etwas schnelleren): Zwei Begrenzungen einführen

Wir wollen nun nicht nur eine Bedingung, sondern mehrere auf einmal testen. Dafür können wir die einzelnen Tests zusammenhängen.

Symbol	Beschreibung	Beispiel
&	Testet, ob das links und das rechts zutrifft	(mousePressed) & (mouseX<10)
|	Testet, ob das links oder das rechts zutrifft	(mousePressed) | (mouseX<10)



Ändern Sie Ihr Programm so ab, dass nur gezeichnet wird, wenn:

- die Maus gedrückt wird **und** man sich im oberen Bereich befindet.

```
if ((mouseY < height/2)& (mousePressed)) {. . . }
```

- die Maus gedrückt wird **oder** man sich im oberen Bereich befindet.

```
if ((mouseY < height /2) | (mousePressed)) {. . . }
```

- man sich im oberen rechten Bereich befindet

```
if ((mouseY < height /2) & (mouseX > width/2)) {. . . }
```

- man sich im unteren linken Bereich befindet

```
if ((mouseY > height /2) & (mouseX < width/2)) {. . . }
```

Sie können natürlich auch wieder mit Farben arbeiten.
Ändern Sie Ihr Programm so ab, dass

- Im oberen Bereich rot gezeichnet wird und im unteren blau

```
void setup(){
  size(400,400);
  background(255,255,255);
}
void draw(){
  if (mouseY < height/2){
     fill(255,0,0);
  }
  if (mouseY > height/2){
     fill(0,0,255);
  }
  ellipse(mouseX, mouseY, 10,10);
}
```

Speichern oder notieren Sie sich jeweils den Programmcode.

2.7 Programmierung von Spielen (Arbeitsblatt 7)

Der letzte Teil kann den Fortschritten der Klasse, bzw. der einzelnen Schülerinnen und Schülerx angepasst werden. Wer eine Herausforderung sucht, kann das "Pong"-Spiel programmieren, die anderen können kleinere Spiele programmieren, welche nicht so komplex sind.

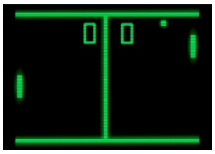
2.7.Variante A: Kleinere Spiele

Z.B. Geheimbotschaften und einfache Verschlüsselungen.

Aufgabenblatt 7a: Geheimbotschaft entschlüsseln

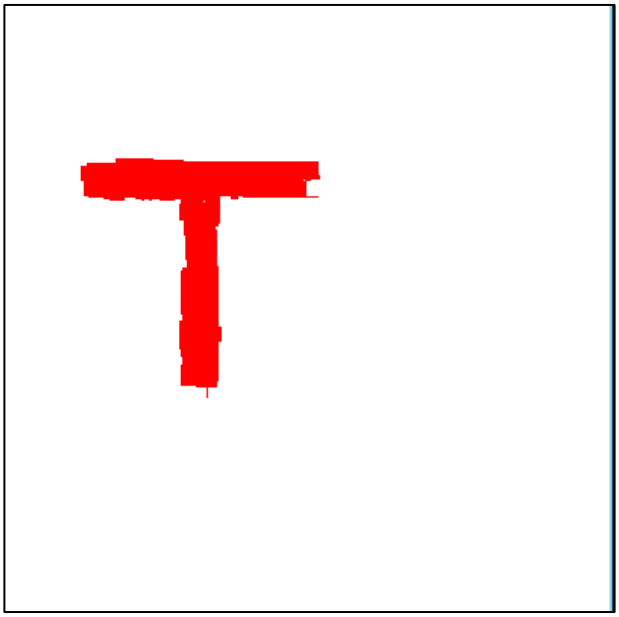
In dieser Aufgabe sollen Sie eine geheime Botschaft in einem Programm verstecken. Der Benutzer des Programms findet die Botschaft indem er sie durch „Zeichnen“ aufdeckt. Kopieren Sie als Beispiel den folgenden Code.

```
void setup(){
```



```
  size(400,400);
  background(255);
  noStroke();
}

void draw(){
  if (mousePressed){
    rect(mouseX,mouseY,10,10);
    if (mouseX<200){
      if (mouseX>50){
        if (mouseY<120){
          if (mouseY>100){
            fill(255,0,0);
            rect(mouseX,mouseY,10,10);
            fill(255,255,255);
          }
        }
      }
    }
    if (mouseX<50+85){
      if (mouseX>50+65){
        if (mouseY<250){
          if (mouseY>120){
            fill(255,0,0);
            rect(mouseX,mouseY,10,10);
            fill(255,255,255);
          }
        }
      }
    }
  }
}
```



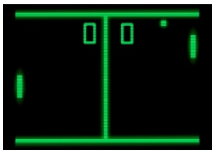
Was ist die geheime Botschaft des Programms?

Verschlüsseln Sie nun Ihre eigene Geheimbotschaft, nach dem gegebenen Beispiel. Folgende Varianten können Sie dabei ausprobieren:

- Geheimbotschaft erscheint nur, wenn der geheime Schlüssel gedrückt wird (zum Beispiel die Taste g)
- Die verschiedenen Teile erscheinen mit unterschiedlichen Vorbedingungen. Ein Teil erscheint, wenn die Maus gedrückt ist, ein anderer, wenn eine bestimmte Taste gedrückt wird.

2.7.Variante B: Pong

Das hier programmierte Spiel ist eine Vereinfachung des Originals. Es wird dabei nur ein Schläger simuliert mit dem gegen eine Wand gespielt wird. Die Flugbahn des Balles wird auch nicht durch den Schläger beeinflusst.



Das Spiel Pong wird in mehreren geführten Schritten programmiert. Als erstes wird die Bewegung des Schlägers, anschliessend der Ball programmiert.

HINWEISE:

- Die Programmierung des Spiels ist eine Herausforderung, die aber zu bewältigen ist. In einer Testklasse (8. Schuljahr, progymnasiale Stufe), konnten ca. 1/3 der Jugendlichen die Aufgabe ganz ohne Angaben zur Lösung bewältigen.
- Die Schwierigkeit besteht nicht darin, dass ganz neue Programmierelemente dazukommen. Alle notwendigen Aspekte sind in den Aufgabenblättern 1-6 eingeführt worden. Die Schwierigkeit besteht in der korrekten Zusammensetzung und richtigen Verknüpfung aller Teilaspekte.
- Lernende, die keine Lust auf Pong haben, können sich in dieser Zeit sehr gut mit den einfacheren Spielen oder weiterhin mit Aufgaben aus vorangegangenen Aufgabenblättern beschäftigen und diese auch kreativ weiterentwickeln.

Aufgabenblatt 7b: Einfache Pong Variante

Nun haben wir fast alles beisammen, um unser kleines Pong-Spiel zu programmieren. Bisher haben wir folgendes gesehen:

- Zeichnen von Figuren
- Setzen von Farben
- Erstellen und Verwenden von eigenen Variablen
- Verwenden von Processing-Variablen
- Bedingte Programmausführung

Wir werden unser Pong-Spiel in zwei Teilen programmieren:

1. Der Balken, der rauf und runter fährt (Der Schläger).
2. Der Ball, der an den Seiten, bzw. dem Balken abprallt.

Der Schläger

Als erstes soll der Schläger simuliert werden. Dafür werden wir ein Rechteck verwenden, welches am linken Rand von oben nach unten wandern kann. Für die Steuerung werden wir die Tasten ‚w' (für aufwärts) ‚y' (für abwärts) verwenden.

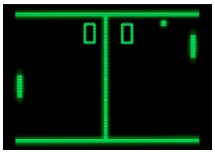
Aufgabe 1: setup-Methode

In der Methode setup soll folgendes gemacht werden:

- Die Grösse des Fensters wird auf 400x400 Pixel gesetzt
- Der Hintergrund wird auf Weiss gesetzt
- Die Füllfarbe wird auf Schwarz gesetzt.
- Der Schläger wird mit folgenden Eigenschaften gezeichnet: Abstand vom linken Rand sind 10 Pixel, ungefähr in der Mitte des Feldes mit der Breite 5 Pixel und der Höhe 40 Pixel.

Aufgabe 2: Position des Schlägers speichern

Was wir uns nun noch merken wollen, ist die Position des Schlägers. Dies machen wir mit einer Variablen pos. Definieren Sie diese Variable ganz am Anfang des Programms und setzten Sie den Wert auf die y-Koordinate des Schlägers.



Aufgabe 3: Bewegung des Schlägers

Nun soll sich der Schläger natürlich noch bewegen. Schreiben Sie die folgende Methode:

```
void draw(){
  if(keyPressed){
    if (key=='w'){
      pos=pos-3;
    }
    if (key=='y'){
      pos=pos+3;
    }
  }
  rect(10,pos,5,40);
}
```

Testen Sie nun dieses Programm. Macht es das, was Sie von ihm erwarten würden?

Aufgabe 4: Löschen des Hintergrunds

Anscheinend muss das vorher gezeichnete wieder gelöscht werden. Dies wird so gelöst, dass zu Beginn der Methode draw der Hintergrund wieder auf Weiss gesetzt wird. Fügen Sie diese Anweisung hinzu.

Aufgabe 5: Schläger im Spielfeld behalten

Nun haben wir noch das Problem, dass der Schläger aus dem Spielfeld gesteuert werden kann. Das wollen wir natürlich verhindern.
Bevor der Wert von pos um 3 reduziert wird, soll vorher geprüft werden, ob der Wert denn noch über 0 liegt. Das wird wie folgt gemacht:

```
if (pos>0) {
      pos=pos-3;
}
```

Schreiben Sie nun einen analogen Test, der prüft, ob der Schläger schon unten am Bildschirm angekommen ist. Denken Sie dabei daran, dass die Position die obere linke Ecke des Schlägers bezeichnet. Der Schläger hat aber noch eine bestimmte Länge.

Der Ball

Und nun wollen wir noch den Ball simulieren, zuerst einmal in einem eigenen kleinen Programm.
Wir brauchen:

- Zwei Variablen für die Position des Balles(x=200 und y=200 Position)
- Zwei Variablen für die Geschwindigkeit des Balles. (speedx=2 und speedy=1)

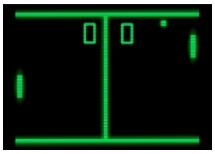
Aufgabe 6: Die Variablen

Für den Ball brauchen wir vier Variablen. Zwei Variablen, um uns die Position (x- und y-Koordinate) zu merken. Deklarieren Sie diese zwei Variablen.
Zwei weitere Variablen speichern die Geschwindigkeit des Balles in der x- bzw. y-Achse. Deklarieren Sie diese zwei Variablen (speedx und speedy) und setzen Sie den Wert von speedx auf 2, sowie den von speedy auf 1.

Aufgabe 7: setup-Methode

In der Methode setup soll das analoge gemacht werden wie beim Schläger:

- Die Grösse des Fensters wird auf 400x400 gesetzt.



- Der Hintergrund wird auf Weiss gesetzt.
- Die Füllfarbe wird auf Schwarz gesetzt.
- Der Ball wird in der Mitte des Fensters gezeichnet (an der Position x/y). Der Durchmesser des Balles beträgt 10 Pixel.

Aufgabe 8: Bewegung des Balls

In der Methode draw wird als erstes die Position des Balles verändert. Dazu wird zur x Koordinate der Wert von speedx hinzugezählt. Zur y Koordinate wird der Wert von speedy hinzugezählt.
Anschliessend soll der Ball an der neuen Position wieder gezeichnet werden.
Testen Sie Ihr Programm.

Aufgabe 9: Abprallen des Balles

Mit dieser Lösung wird der Ball einfach aus dem Spielfeld rausfliegen. Wir möchten aber, dass er abprallt. Fügen Sie also die folgenden Zeilen zwischen dem Verändern der Position und dem Zeichnen des Balles ein:

```
if(x>390){
  speedx=speedx*-1;
}
if(x<25){
  speedx=speedx*-1;
}
if(y<5){
  speedy=speedy*-1;
}
if(y>395){
  speedy=speedy*-1;
}
```

Aufgabe 10: Zusammenfügen

Fügen Sie nun die beiden Programme zusammen.

Aufgabe 11: Testen, ob der Ball den Schläger trifft

Diese Aufgabe ist komplex und ist die Abschlussaufgabe des Moduls. Die folgenden Zeilen sollen so geändert werden, dass nur dann die Anweisung speedx=speedx*-1 ausgeführt wird, wenn sich die Position des Schlägers auf der Höhe des Balles befindet:

```
if(x<25){
  speedx=speedx*-1;
}
```