Programmieren lernen mit processing

Ein Arbeitsheft

Autor: Ingo Bartling

Inhaltsverzeichnis

KAPITEL 1: Allgemeines

Was ist Programmieren?

„Programmieren bedeutet ein Problem zu zerlegen, Algorithmen und Abläufe zu definieren und in eine Computersprache zu übersetzten."

Was ist ein Algorithmus?

Ein Algorithmus ist eine Vorschrift, die folgende Eigenschaften erfüllt:

1. **Eindeutigkeit**
   Ein Algorithmus muss in der Beschreibung eindeutig sein.

2. **Ausführbarkeit**
   Jeder Einzelschritt muss ausführbar sein.

3. **Finitheit (Endlichkeit)**
   Der Algorithmus muss in endlicher Zeit ausgeschrieben werden (und damit auch endlich lang sein)

4. **Terminierung**
   Nach endlich vielen Schritten liefert der Algorithmus immer ein Ergebnis

5. **Determiniertheit**
   Bei gleichen Startwerten kommt immer das gleiche Ergebnis heraus

6. **Determinismus**
   Zu jedem Zeitpunkt gibt es nur genaue 1 Möglichkeit, wie fortgesetzt werden kann

Aufgabe 1.1.1: Rezept

Schreibe einen Rezept aus mindestens 5 Schritten auf (oder drucke es aus und klebe es in dein Heft ein). Überprüfe nachvollziehbar anhand der obigen 6 Eigenschaften, ob ein Algorithmus vorliegt.

Was ist eine Computersprache?

Computersprache gibt es in verschiedenen Abstraktionsstufen („01010100101110“ bis Scratch) und dienen der Kommunikation mit einem Computer. Anders als sogenannte natürliche Sprachen wie Englisch, Französisch oder Deutsch sind diese Sprache vor allem eindeutig. Der Schlüsselwort „class“ oder „for“ in der Computersprache bedeutet immer das gleiche. Im Deutschen kann das durchaus anders sein. So kann „Mutter“ mal Mama bedeuten oder auch das Gegenstück bei einer Schraube sein. (Mehr dazu in der 12. Klasse)

Unsere Programmiersprache wird Java sein. Man könnte auch eine andere Sprache nehmen, da es fast egal ist mit welcher Programmiersprache man anfängt. Da aber das bayerische Abitur an Java angelehnt ist, ist dies für Schüler am sinnvollsten.

Was ist Lernen?

Etwas Neues zu lernen erzwingt in der Regel zwei Schritte:

1. Neues integrieren
   Zunächst muss das neue Wissen in Form von Fakten gelernt werden
2. Neues festigen
   Wiederholen, wiederholen, wiederholen

Wenn ich Gitarre lernen möchte, dann muss ich zunächst die Griffe und Anschlag- oder Zupfmuster lernen. Im zweiten Schritt muss das schnelle, automatische Spielen durch viel Üben trainiert werden.

Möchte ich Portraits zeichnen, so muss ich erst die Proportionen genau lernen. Dann übe ich durch viele Wiederholungen.

Beim Programmieren lernen ist dies ähnlich. Zunächst vermittelt der Lehrer die Fakten bevor der Student oder Schüler, angeleitet durch die Lehrperson, selbstständig übt.

Was ist processing?

Mit Java lassen sich die unterschiedlichsten Arten von Software entwickeln: Büroanwendungen, Apps für Android-Handys, Spezial-Programme für Physik, Bank-Software, etc. Für den unerfahrenen Programmierer macht es meiner Erfahrung nach aber am meisten Freude, wenn er Interaktionen, Spiele und Grafik-Effekte erzeugen kann. Hier erzeugen selbst kleinste Anpassungen am selbst geschriebenen Quelltext sicht- und erlebbare Veränderungen.

Weitere Informationen finden sich bei processing.org.

KAPITEL 1.2: Einfache Datentypen und -strukturen

Ein Programm macht nur Sinn, wenn das Programm Daten hat, die verarbeitet werden. Dies können über vielfältige Arten ein Programm zugeführt werden. Die vorliegenden Daten, aber auch das Programm selbst, muss im Speicher des Computers liegen. Damit der Computer weiß, wie viel Speicher er zur Verfügung stellen muss, muss in den meisten Hochsprachen wie Java der Datentyp einer Variablen angegeben werden.

1.2.1 Primitive Datentypen

Im Themenbereich Datenbanken hast du bereits verschiedene Datentypen kennengelernt. Auch in Java gibt es sogenannten primitive Datentypen, also Datenstrukturen, die aus keinen anderen bestehen.

Aufgabe 1.2.1: Datentypen bei Datenbanken

Erstelle eine tabellarische Auflistung von mindestens 4 verschiedene Datentypen, die du beim Thema Datenbanken kennengelernt hast, gib jeweils ein Beispiel an und erkläre knapp die Datentypen.

Aufgabe 1.2.2: Datentypen in Java

Erstelle eine tabellarische Auflistung von der für uns wichtigsten primitiven Datentypen in Java. Übertrage hierzu die Tabelle in dein Heft und ergänze - mit Bleistift.

Datentyp	Beispiel	Erklärung	Wertebereich / Beispiel
float	true		
		Kommazahlen	
	-2		
		Ein einzelnes Zeichen	
String		Mehrere Zeichen	

Auch wenn es kein primitiver Datentyp ist, so ist der Datentyp „String“ dennoch so wichtig, dass ich ihn als grundlegend erachte und an dieser Stelle hier einführen möchte.

Aufgabe 1.2.3: Superhelden-Klasse
Ziehe eine der Superhelden-Karten und wähle für jede Eigenschaft einen passenden Datentyp. Definiere sodann Variablen für die Superhelden-Eigenschaften und gib Variablenwerte wieder aus.

Aufgabe 1.2.4: Schüler-Klasse
Gib Eigenschaften von dir selbst so an, dass jeder wichtiger primitiver Datentyp (boolean, int, float/double, String) mindestens 1 mal benutzt wird und gib alle Werte wieder aus.

1.2.2 Referenz-Datentypen

Aufgabe 1.2.5: Klasse-Definition
Strukturiere die Superhelden-Attribute so, dass eine Klasse „Superheld" entsteht.

Aufgabe 1.2.6: Methode definieren
Definiere eine Methode „ausgeben`()`", die alle Attribute der Klasse Superheld ausgibt.

Aufgabe 1.2.7: Standardkonstruktor
Definiere einen Standardkonstruktor und einen Konstruktor mit Parametern, so dass alle Attribute der Klasse „Superheld" mit Startwert belegt werden können. Achte auf eine ordentliche Strukturierung.

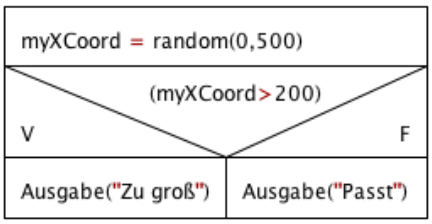
Aufgabe 1.2.8: Methodenkopf & -rumpf
Markiere bei folgenden Methoden den Methodenkopf in Blau und den Methodenrumpf in Grün.

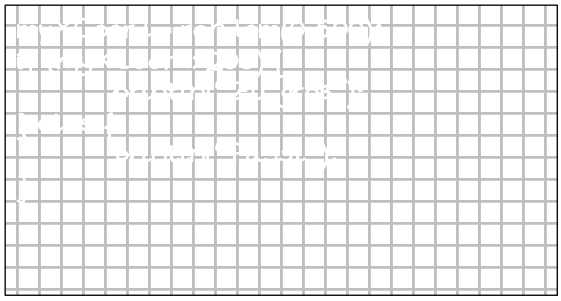
```
void draw() {
   rect(10,10,100,200);
}
```

```
void print(String x) {
     println(x);
}
```

Aufgabe 1.3.1: Struktogramme

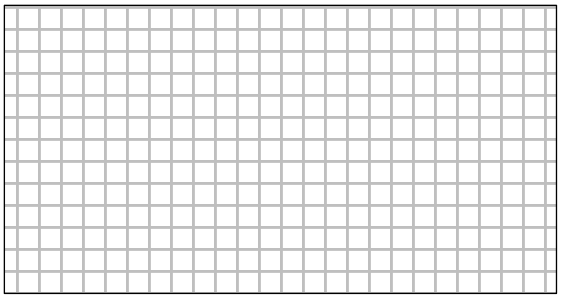
a) Übersetze das folgende Struktogramm in Java!





b) Zeichne das Struktogramm zu folgendem Java-Quelltext!

```
if (newXCoord<0) {
   xCoord = 10;
} else {
   xCoord = newXCoord;
}
```



Aufgabe 1.3.2: Klasse mit Nicht-Standardkonstruktor

Definiere einen Klasse für Rechtecke: xCoord, yCoord, laenge, breite inkl. 2 Konstruktoren und ausgeben()-Methode.

Aufgabe 1.3.3: if-else

Definiere eine Klasse für Rechtecke: xCoord, yCoord, laenge, breite inkl. 2 Konstruktoren und ausgeben()-Methode.

Ergänze den Nicht-Standardkonstruktor um if-else-Kontrollstrukturen.

Aufgabe 1.3.4: Die Klasse Kreis

a) Definiere eine Klasse für Kreise, wobei die Klasse folgenden Attributen und Konstruktoren umfasst: xCoord, yCoord, durchmesser, inkl. 2 Konstruktoren und ausgeben()-Methode.

b) Ergänze den nicht-Standardkonstruktor um if-else-Kontrollstrukturen.

c) Verändere den Standardkonstruktor so, dass die Startwerte per Zufall belegt werden. Bediene dich dabei der processing-Funktion

```
random(float unterWert, float obererWert)
```

Kapitel 1.4: Was du bis jetzt können musst

Allgemeines zu Java

- ☐ Den Unterschied zwischen *.java und *.class-Dateien erklären können.

  _____

- ☐ Den Begriff kompilieren erklären können.

  _____

- ☐ Erläutern können, warum ein Java-Programm kompiliert werden muss.

  _____

- ☐ Erläutern können, warum nicht auf jedem Computer Java-Programme ablaufen.

  _____

- ☐ Das Programm processing und processing.org kennen.
- ☐ Ein- und mehrzeilige Kommentare einfügen können und wissen, was und wie kommentiert werden muss.

  _____
  _____
  _____
  _____
  _____
  _____
  _____
  _____
  _____
  _____
  _____
  _____
  _____

- ☐ Das Wort „implementieren“ erläutern können.

  _____

Die Grundbegriffe des Programmierens[1]

☐ Eine Klasse deklarieren können

☐ Attribute deklarieren können

☐ Eine Variable deklarieren können

☐ Alle wichtigen, primitive Datentypen angeben und erklären können

☐ Eine Variable mit eine Startwert initialisieren können.

☐ Zeichenketten (mit Variablen-/Attributwerten) verknüpfen können

[1] in Java

☐ Welche Eigenschaft wird durch „public“ und „private“[2] definiert und was bedeuten sie?

☐ Grundlegende Programmierrichtlinien kennen (Einrückungen, Groß-/Kleinschreibungen, KamelHöckerschreibweise)

☐ Die Aufgabe eines Konstruktors angeben können

☐ Standardkonstruktoren .....

☐ ..... und Konstruktoren mit Parametern definieren können

☐ Die Werte der Eingangsparameter in den Attributen speichern können

[2] „protected“ spielt im Anfangsunterricht aus meiner Erfahrung heraus keine Rolle.

☐ Wissen, was ein Methodenkopf ...

☐ .... usnd was ein –rumpf ist.

☐ Die Bestandteile eines Methodenkopfes in der richtigen Reihenfolge angeben können.

☐ Den Unterschied zwischen Prozedur und Funktion benennen können

☐ Eine Ausgabe in das Editorfenster schreiben können

☐ Ein Objekt im Quelltext erzeugen können

☐ Ein Methode eines Objektes im Quelltext aufrufen können.

Kontrollstrukturen

☐ Eine bedingte ein/zweiseitige Verzweigung angeben können.

☐ Du kennst alle Vergleichsoperatoren

☐ Alle booleschen Operatoren an einem Beispiel erklären können

KAPITEL 1.4: Graphik in Processing

Aufgabe 1.4.1: Farben

In welcher Einheit wird der Bildschirm unterteilt? _____

Erläutere die Bedeutung des Befehls strokeWeight(x) und fill(x). Aus welchen Wertebereich darf x gewählt werden? _____

_____

_____

Zur Darstellung von Farben wird häufig das RGB-Modell genommen. Erläutere dies am Beispiel „fill(255,0,255)“. _____

_____

_____

Aufgabe 1.4.2: setup() und draw()

Viele processing-Projekte werden vor allem mit Hilfe zweier Methoden gesteuert. Erläutere!

Befehl	Fragen	Deine Erläuterung

void setup() {
 	
}
	Wie oft wird diese Methode aufgerufen?	

void draw() {
 	
}
	Wie oft wird diese Methode aufgerufen?
	_____	

Mit Hilfe welchen Befehls kann ein mehrfaches Wiederholen der Methode draw() verhindert werden?_____

Aufgabe 1.4.3: Das Koordinatensystem

Erläutere das Koordinatensystem (Ursprung, Achsen) in processing!

_____

_____

Aufgabe 1.4.4: Grundlagen des Zeichnens

Zur Darstellung von Objekten in processing musst du ein paar wichtige Methoden und Funktionen kennen. Ergänze die Tabelle jeweils um eine knappe Erklärung oder gib den Befehl an. Beantworte auch die Fragen. Informationen findest du auf processing.org.

Befehl	Frage	Erläuterung
size(600,400)		
		Das Zeichenfenster soll genauso groß sein wie der Bildschirm.
frameRate(60)	Wie hoch ist der Standardwert? _____	
background(0)		
	Auf welche Ecke des Objekts bezieht sich die Positionsangabe?	Es wird ein Rechteck an der Position (100|200) mit der Breite 300 und Höhe 50 gezeichnet.
ellipse(20,50,100,100)		
		Es soll eine Linie vom Punkt (100|200) zum Punkt (300|400) gezogen werden
mouseX mouseY		
height width		

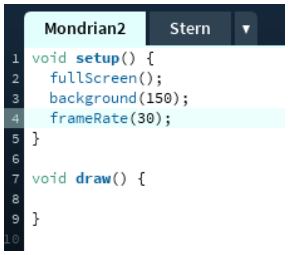
Kapitel 1.5: Komplexere Programmieraufgaben

1. Lese die ganze Aufgabenstellung.
2. Lies die Aufgabenstellung nochmals und markiere alle Substantive, Adjektive und Verben unter folgendem Aspekt:

   Substantiv – mögliche Klasse
   Adjektiv – mögliches Attribut einer Klasse
   Verb – mögliche Methode einer Klasse
3. Lege ein neues processing-Projekt an und definiere die Methoden setup() und draw() im ersten Reiter des Projekts.
4. Ähnlich zum Kochen erfolgt nun das sogenannte "Mise-en-place". Ergänze die setup()-Methode so, dass die äußeren Rahmenbedingungen für dein Projekt gegeben sind:

   - Fenstergröße
   - (Hintergrund)-Farbe(n)
   - frameRate(30) etc.

   Lege für jeden Referenz-Datentyp eine neue, aber noch leere Klasse in einem eigenen Reiter an.



5. Beginne nun die eigentliche Aufgabe zu lösen, in dem du möglichst einfach beginnst und schrittweise dein Programm erweiterst, verallgemeinerst und um weitere Funktionen ergänzt.

   Beachte: - Wechsle erst in eine neue Zeile wechselst, wenn die aktuelle Zeile korrekt ist.
   - Achte darauf, dass dein Programm immer funktioniert.

Beispiel an der Aufgabe „Mondrian 2"

1. Definiere die Methode setup() mit Methodenrumpf und draw(). Lege eine Klasse „Kreuz" an.
2. Erzeuge innerhalb der Methode draw() ein Kreuz an einer speziellen Stelle z.B. (100|100)
3. Lass dieses Kreuz an der Stelle eines Mausklicks erzeugen.
4. Passe nun die Klasse Kreuz so an, dass mit Hilfe des Standardkonstruktors „Kreuz()" ein Kreuz an der Stelle (100|100) erzeugt wird. Ergänze hierzu die Klasse um die benötigten Attribute, den Standardkonstruktor und eine Methode show(). show() ist dabei nahezu identisch zu dem Programmcode aus Punkt 2.
5. Passe nun die draw()-Methode an:

```
void draw(){
    Kreuz k1 = new Kreuz();
    k1.show();
}
```

Aufgabe 1.5.1: Abschlussaufgabe 1 (Strichcode)

a) Lege ein neues processing-Projekt mit dem Namen „Strichcode“ an.

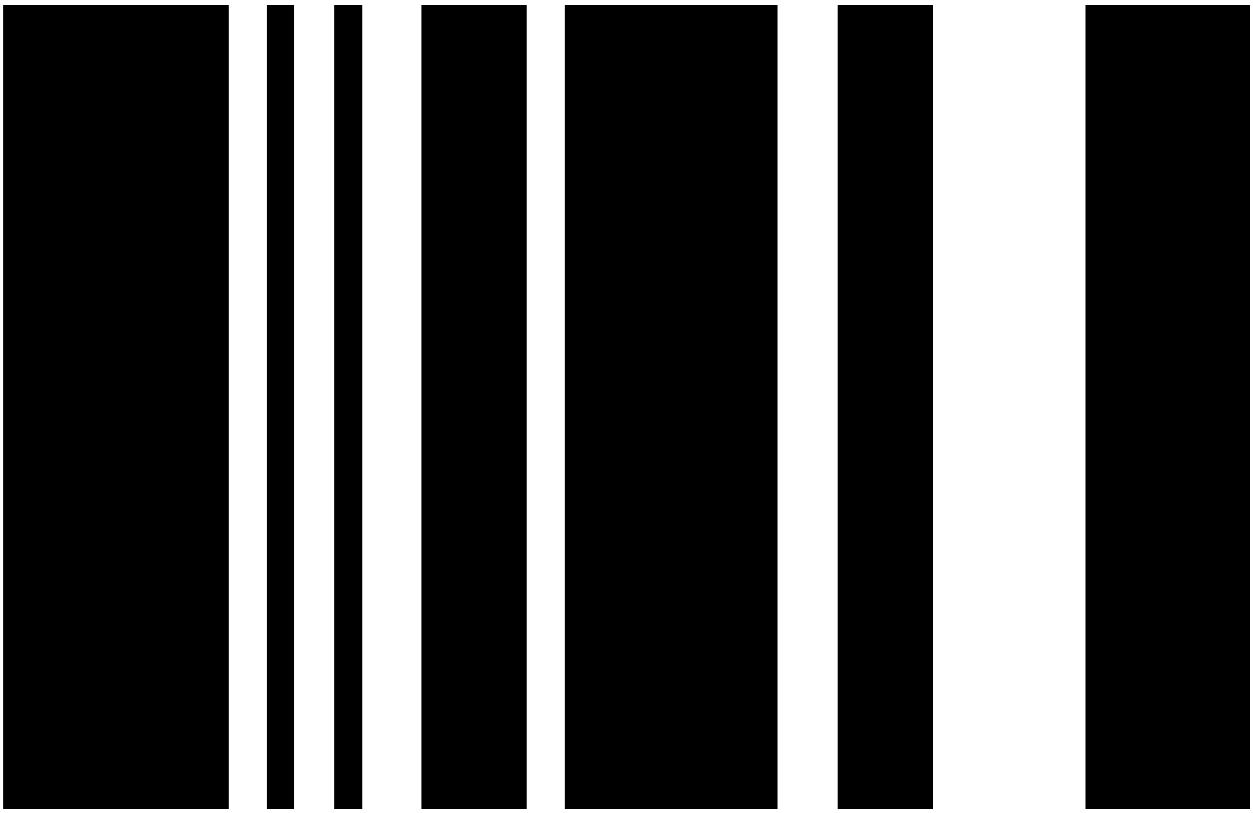
b) Erzeuge ein Fenster in Bildschirmgröße mit schwarzem Hintergrund.

c) Zeichen ein weißes Rechteck über die gesamte Bildschirmhöhe, das 100px breit ist und einen 10px breiten schwarzen Rand besitzt sowie die x-Position 50% von der Bildschirmbreite besitzt.
   Warum ist das Rechteck dennoch nicht genau in der Mitte des Bildschirms?

d) Reduziere die frameRate auf 5 und lasse bei jedem neuen Bildaufbau ein bildschirmhohes, zufällig x-positioniertes, weißes Rechteck zeichnen.

e) Erhöhe die frameRate auf 10 und verändere das Programm so, dass das die Rechtecke mit Hilfe der Maus positioniert werden können, aber immer noch Bildschirm hoch sind.

f) Verändere das Programm so, dass mit 80%iger Wahrscheinlichkeit ein weißes, sonst aber ein schwarzes Rechteck gezeichnet wird.
   Tipp: random(1,10)<9



Aufgabe 1.5.2: Abschlussaufgabe 2 (Mondrian 1)

a) Lege ein neues processing-Projekt mit dem Namen „Mondrian1“ an.

b) Erstelle ausschließlich mit den Methoden setup() und draw() ein Programm mit dem Bilder ähnlich zu Mondrians Bildern erzeugt werden können. Dabei soll ein Fenster der Größe 800x400 mit Rechtecken gefüllt werden, die eine Zufallsfüllfarbe und einen Zufallsbreite sowie –höhe haben. Die Rechtecke besitzen einen 10px breiten, schwarzen Rand.

c) Die Rechtecke sollen automatisch erzeugt werden, wobei ungefähr pro Sekunde nur 1 Rechteck erzeugt wird.

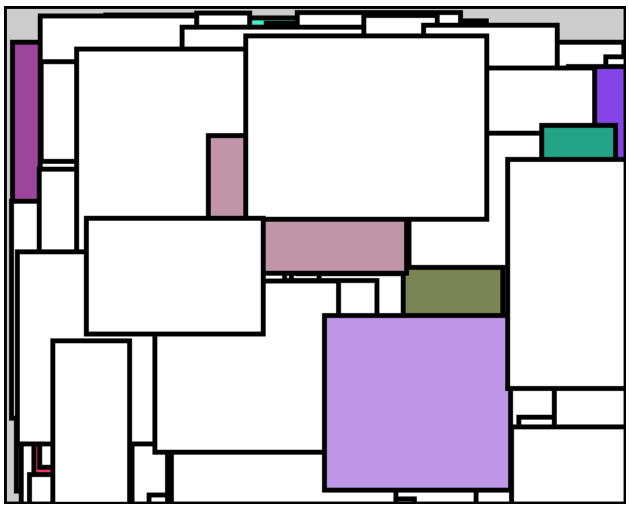
d) Durch Linksklick mit der Maus soll das Bild gelöscht werden.

e) Durch Klicken mit der linken Maustaste werden Rechtecke an der Mausposition erzeugt.

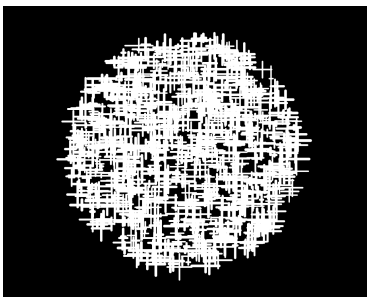
f) Statt Rechtecken werden Quadrate mit einer Zufallsgröße erzeugt.

Sternchenaufgaben dienen dazu dein Problemlösungsdenken zu testen und zu schulen. Diese Aufgaben haben ihre Schwierigkeit oftmals in der Frage: In welchen Schritten löse ich das Problem am einfachsten? Oftmals hilft es sich zu vorzustellen, wie man in der Realität das Problem lösen könnte: „Erst dann Rand malen und dann die Rechtecke. Oder erst die Rechtecke und dann zum Schluss den Rand übermalen.“

g*) Die Position der Rechtecke muss so eingeschränkt werden, dass am Rand ein 10px-breiter schwarzer Rand bleibt.



Aufgabe 1.5.3: Abschlussaufgabe 3 (Mondrian2)



Überlege zunächst welche Information du zum Zeichnen eines Kreuzes benötigst. Programmieren beginnt sollte immer mit Papier und Bleistift beginnen und nicht am Computer.

a) Lege ein neues processing-Projekt mit dem Namen „Mondrian2“ an.

b) Lege eine neue Klasse „Kreuz“ an. Jedes Kreuz soll als Attribute unter anderem xPos und yPos haben.
Jedes Kreuz besteht aus 10px-breiten schwarzen Linien, deren Länge (horizontal wie vertikal) zwischen 10 und 50 Pixeln liegt.

c) Implementiere einen Standardkonstruktor und einen Nicht-Standardkonstruktor.

d) Implementiere auch eine Methode gibAus(), die alle Attribute-Werte in der Konsole ausgibt.

e) Ebenso wird eine Methode show() benötigt, die das Kreuz auf Basis der Attribute-Werte zeichnet.

f) Lege in der Projekt-Klasse die Methode setup() und draw() an. Bei Klick mit der linken Maustaste soll ein Kreuz gezeichnet werden.

g**) Die Position der Kreuze muss so eingeschränkt werden, dass ähnlich zu nebenstehendem Bild ein Kreis gebildet wird.

h*) Mache die Kreuze transparent. Je weiter sie von Mittelpunkt des Fensters entfernt sind, desto durchsichtiger sollen die Kreuze erscheinen. Benutze hierfür den sogenannten Alpha-Kanal einer Farbe, also z.B. stroke(255,100). 100 bestimmt in diesem Fall die Transparenz. Der Wert geht von 0 bis 255.

i**) Verändere die Transparenz oder Farbe mit der Anzahl der durchlaufenen Frames.

Benutze die processing-Funktion „line(x1,y1,x2,y2)“ mit der eine Linie vom Punkt (x1|y1) zum Punkt (x2|y2) gezogen wird.

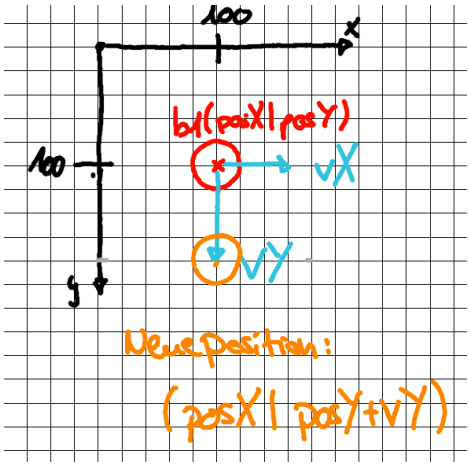
Für die Teilaufgabe g**) kann die Funktion „dist(x1,y1,x2,y2)“ benutzt werden. Diese berechnet den Abstand zwischen dem Punkt (x1|y1) und dem Punkt (x2|y2).
Für die Teilaufgabe h*) kann zusätzlich die Funktion map() benutzt werden.
Bei der Teilaufgabe i**) kann mit dem Modulooperator % gearbeitet werden, der den Rest einer Division zurückliefert: zum Beispiel:15%256 = 15, 255%256 = 255, 256%256 = 0.

Kapitel 2.1: Bewegung mit konstanter Geschwindigkeit

Nach den „statischen“ Effekten werden nun einzelnen Figuren wie Kreise und Linien animiert. In einem späteren Kapitel werden dann passend zu Spielen Grafiken animiert.

Anders als in der Physik, wird Bewegung damit nicht in m/s angegeben sondern in „Schrittweite in Pixel“ pro Frame. Die Bewegungsgeschwindigkeit lässt sich zum einen über die „frames per second“ steuern, aber auch über die Schrittweite pro Frame. Das Problem im ersten Fall ist, dass die framerate nicht zwingend gewährleistet werden kann, da diese auch von der Hardware abhängt und nicht beliebig nach oben geregelt werden kann. Wenn nichts anderes angegeben wurde, ist eine framerate von 30 eine gute Wahl. Die Bewegung ist flüssig und lässt sich ganz gut auf m/s umrechnen.

Im Gegensatz zur Realität ist die Bewegung in processing ruckartig: Pro Frame wird beispielsweise ein Kreis um 10px bewegt. Ist ein Kollisionsobjekt nur 1 Pixel entfernt, überlagern sich die Objekte. Wird also die „Schrittweite in Pixel“ zu hoch eingestellt, so wird diese sogenannte „collision detection“ immer komplexer und eine zugehörige immer aufwändiger. In der Regel hat sich ein Wert aus [-5;5] bis maximal |10| bewährt.



Basis der Bewegungsberechnungen bildet die Physik der (beschleunigten) Bewegung.

Im einfachsten Fall ist die Beschleunigung a=0.
Damit ergibt sich die neue y-Position des Balls aus der Berechnung:

„Alte y-Position + Geschwindigkeit“ ergibt die neue „y-Position“.

Übersetzt in Programmcode würde dies lauten:

_____ oder in Kurzform: _____

und analog für die x-Koordinate _____ .

Aufgabe 2.1.1: Klasse Ball

a) Lege ein neues processing-Projekt mit dem Namen „Ballphysik“ an. Lege einen weiteren Reiter „Ball“ in diesem Projekt an. Implementiere die Klasse „Ball“ mit einem Standardkonstruktor, so dass oben abgebildete Situation dargestellt wird: Der Ball hat einen Radius von 20px und ist an der Position (100|100).

b) Ergänze die Klasse „Ball“ um die Attribute „float vX;“ und „float vY“. Die Startwerte für die Geschwindigkeitsattribute sollen zunächst auf „vX=0“ und „vY=2;“ festgelegt sein, um eine eindeutige Ausgangssituation zu definieren.

Aufgabe 2.1.2: Bewegung mit konstanter Geschwindigkeit

a) Implementiere in der Klasse „Ball“ eine Methode „void update()“, welche die neue Position des Balls berechnet.
Auf der Karte „Ballphysik“ soll nebenstehender Inhalt dabei umgesetzt.

```
Ball b1 = new Ball();
void setup() {
  background(0);
  framerate(30);
}
void draw() {
  b1.update();
  b1.show();
}
```

b) Erläutere die Zeile

```
Ball b1 = new Ball();
```

b) Erläutere weiter, warum die Zeile „Ball b1 = new Ball();“ außerhalb jeglicher Methode stehen kann.

_____

_____

f) Welchen Effekt hätte es, wenn die Zeile „Ball b1 = new Ball();“ innerhalb der Methode draw() stehen würde.

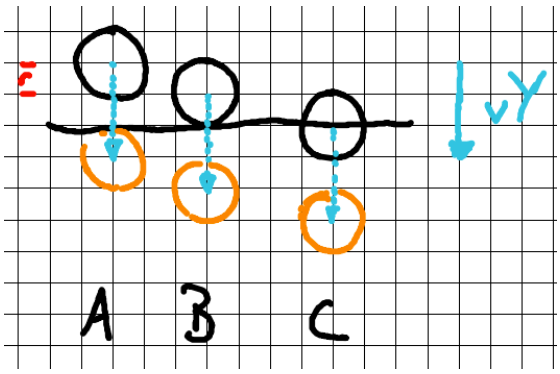
_____

_____

_____

g)

Damit der Ball am unteren Bildschirmrand abprallt, müssen die nebenstehenden drei Situation A, B, C erfasst werden. Hierbei wird ausgehend von der y-Position zunächst der Radius r (rot) hinzuaddiert, um den untersten Punkt des Balls zu berechnen.



Nun wird geprüft, ob beim nächsten Animationsschritt, der unterste Punkt des Balls den Bildschirmrand berührt oder sogar darüber hinzugeht.

Aufgabe 2.1.3: Abprallen am Rand

a) Setze die Formulierung des letzten Absatzes in Programmcode um:

_____

Wenn die Bedingung für das Abprallen erfüllt ist, wird der Wert von vY einfach mit „-1“ multipliziert. Anstatt also 100+2 und damit 102 als neue y-Koordinate zu erhalten, wird 100+(-2) = 98 berechnet. Der Ball bewegt sich damit wieder nach oben.

b) Gib die vollständige bedingte Verzweigung für ein korrektes Abprall-Verhalten am unteren Bildschirmrand an.

_____

_____

_____

c) Gib analog zur Teilaufgabe b) den Programmcode für das Abprallen am linken und rechten Bildschirmrand an.

d) Wie du vielleicht gemerkt hast, hast du zwei Mal fast den gleichen Programmcode eingegeben. Nur die Bedingung hat sich geändert. Umgangssprachlich würde man sagen: „Wenn der Ball am linken Rand ODER der Ball am rechten Rand ist, tue...“. Dieses „ODER“ gibt es auch beim Programmieren: _____ . Das logische UND würde lauten: _____. Fülle nun die folgenden Tabellen aus:

ODER	true	false
true		
false		

UND	true	false
true		
false		

e) Fasse die Verzweigungen nun zusammen. Eine für die Vertikal-Bewegung und eine für die Horizontal-Bewegung.

_____

_____

_____

Kapitel 2.2: Bewegung mit konstanter Beschleunigung

Aufgabe 2.2.1: Bewegung mit konstanter Beschleunigung

Definiere in der Projekt-Klasse „Ballphysik“ eine Variable „g“als Simulation für die Erdbeschleunigung. Initialisiere g in der Methode setup() auf 0.1 . Passe nun die Geschwindigkeit innerhalb der update()-Methode an. Dabei soll vY solange um g erhöht werden, bis der maximale Wert 10 erreicht wurde, denn Körper können aufgrund der Luftreibung nicht beliebig schnell fallen.
Achte in der Methode update() darauf, dass die Schritte Beschleunigen, Abprallen, neue Position in dieser Reihenfolge abgearbeitet werden.

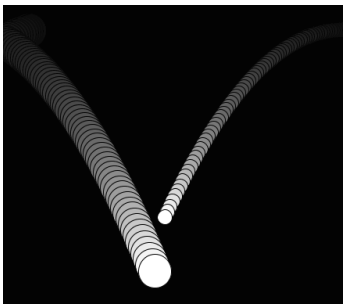
Aufgabe 2.2.2: Energieverlust

Um ein realistischeres Verhalten des Balls zu erhalten (u.a. eine abnehmende Sprunghöhe), soll noch ein „Energieverlust“ simuliert werden. Hierzu wird vX (Gleit- bzw. Rollreibung) und vY (Verformungsarbeit) bei Kollision mit dem Boden auf 90% verringert werden.
Gib die zugehörigen Programmierzeilen an:

_____

_____

Aufgabe 2.2.3: Schlieren-Effekt



Ein schöner Effekt ist der nebenstehende Effekt, den ich Schlieren-Effekt nenne. Dieser wird dadurch erreicht, dass bei jedem Durchgang von draw() das gesamte Bild leicht transparent übermalt wird:

```
//Dunkles Grau (12) das fast vollständig durchsichtig ist (10)
fill(12,10);
//Fenstergroßes Rechteck
rect(0,0,width,height);
//Füllfarbe wieder zurücksetzen
fill(255);
```

Aufgabe 2.2.4: Bildschirmschoner

Lege ein ein neues Projekt „BildschirmSchoner“ an. Erstelle zwei neue Dateien „Linie“ und „Punkt“. Der Anfangs- und Endpunkt der Linie wird durch einen „Punkt“ definiert.

a) Übertrage die Animation aus der Aufgabe 2.1.3 auf die Punkte, so dass die Punkte sich über den Bildschirm bewegen und abprallen würden.

b) Animiere nun die Linie. Benutze hierzu das beistehende Grundkonstrukt für die Klasse Linie.

```
class Linie {
  Punkt anfangsPunkt;
  Punkt endPunkt;
  float r,g,b;
  float thickness;

  void update() {
    anfangsPunkt.update();
    endPunkt.update();
  }

  void show() {
    strokeWeight(thickness);
    stroke(r,g,b);
    line(anfangsPunkt.x,anfangsPunkt.y,endPunkt.x,endPunkt.y);
  }
}
```

*Aufgabe 2.2.5 *: Fliegender Text*

Ersetze die Linie aus Aufgabe 2.1.7 durch einen Text, der sich über den Bildschirm bewegt. Informiere dich dazu auf der Internetseite processing.org über Funktionen mit denen du die Darstellung und Bewegung des Textes steuern und beeinflussen kannst.
Hinweis: text(...), textSize(...), textWidth(...)

Kapitel 3.1: for-Schleife und Arrays

In den allermeisten Spielen und Animationen gibt meistens nicht nur 1 oder 2 Objekte, sondern viele: Mehrere Feinde oder mehrere Hindernisse. Hierfür kann man sich der Datenstruktur Feld (engl. Array) bedienen. Um effizient mit der Datenstruktur zu arbeiten, bedient man sich häufig einer Zählwiederholung oder „for-Schleife".

Da es absolut notwendig ist, dass die Syntax der Zählschleife auswendig sitzt, folgen zunächst ein paar Fingerübungen.

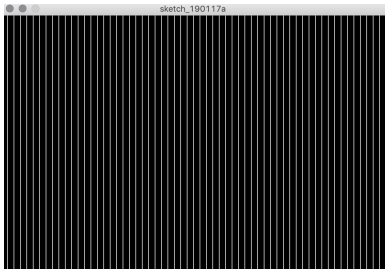
Aufgabe 3.1.1: Fingerübungen für Zählwiederholungen

Tippe den Quelltext ab und löse die Aufgaben, die als Kommentare in den Methoden stehen.

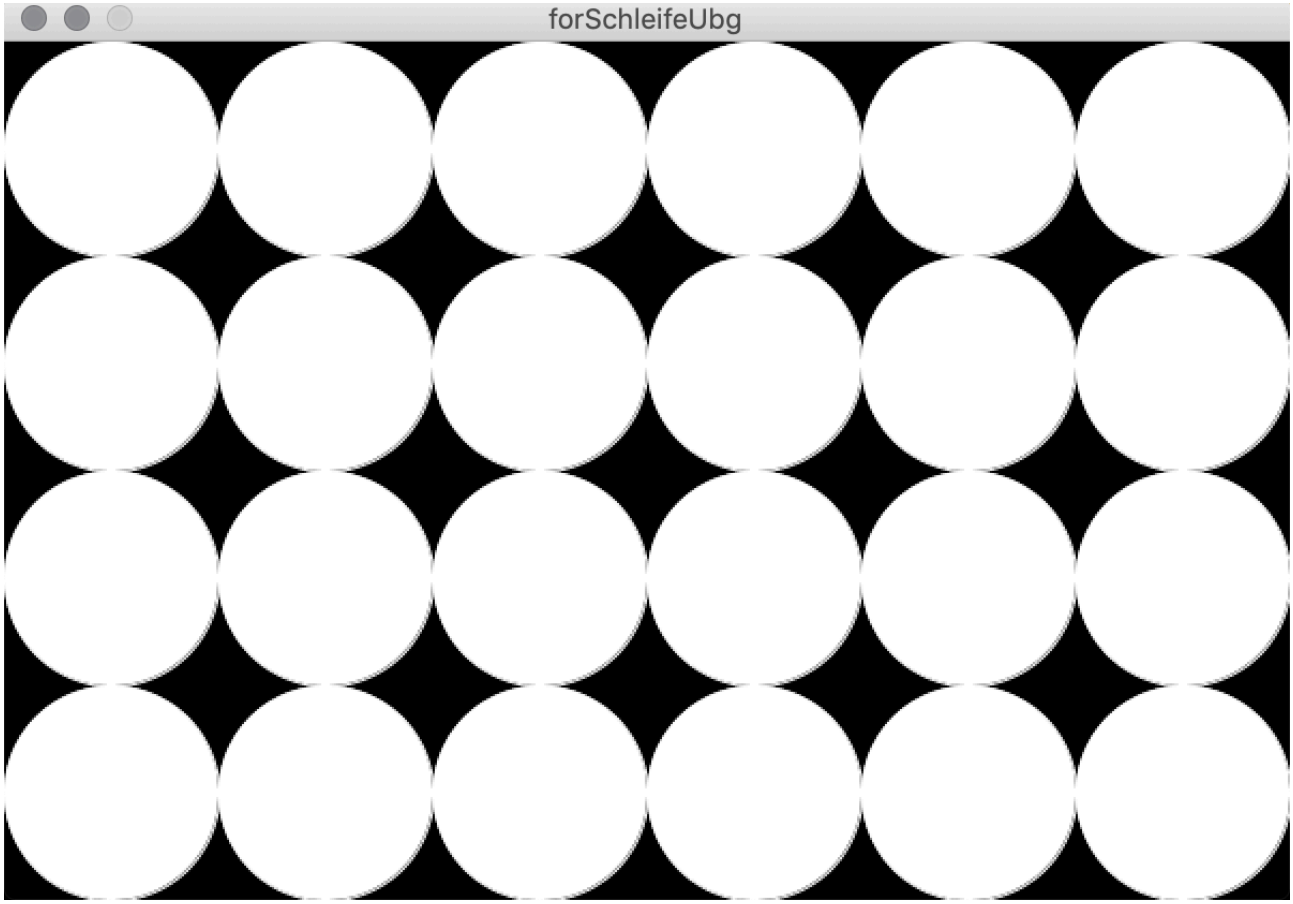
```
void setup() {
  size(600,400);
  background(0);
  stroke(255);
}

void draw() {
  aufgabe3c(50); //Hier wird die jeweilige Aufgaben angegeben
  noLoop(); //Abbruch von draw();
}

void aufgabe1() {
  //Zeichne senkrechte Linien im Abstand 10px
}

void aufgabe1b(int abstand) {
  //Zeichne senkrechte Linien in einem Abstand.
  //der mit Hilfe eines Parameters "abstand" übergeben wurde
}

void aufgabe2() {
  //Zeichne "kariertes" Papier mit 10px Abstand
}

void aufgabe2b(int abstand) {
  //Zeichne "kariertes" Papier mit "abstand" Abstand
}
void aufgabe3() {
```

Grundwissen

```
for (int idx=0; idx<100; idx+=1) {
  line(x1,y1,x2,y2);
}
```



```
  //Zeichne 1 Reihe von Kreisen mit r=20px,
  //die sich berühren
}

void aufgabe3b(int radius) {
  //Zeichne 1 Reihe von Kreisen mit r=radius,
  //die sich berühren
}

void aufgabe3c(int radius) {
  //Fülle das Fenster mit Kreisen mit r=radius,
  //die sich berühren
  //Tipp: benutze verschachtelte for-Schleifen
}
```



Aufgabe 3.1.2: Bildmanipulation (Geschachtelte for-Schleifen)

Da die Pixel eines Bildes in einem Array verwaltet werden können einfach Filter, wie sie bei vielen Apps eingebaut sind, selbst programmiert werden.

a.) Tippe zunächst den beistehenden Code ab.

```
PImage img;  //Ein processing Bild-Objekt names img

void setup() {
  size(265, 265);  //Größe des Bildes
  //Das Bild muss im gleichen Projekt-Ordner
  //oder im Ordner data im Projektordner liegen
  img = loadImage("flower.jpg");
}

void draw() {
  loadPixels(); //Alle Pixel des Bildes in das Feld pixels laden
  img.loadPixels();
  //----- Bildmanipulation ----------

  //---------------------------------
  updatePixels(); //Das Feld pixels wieder in das Bild zurückschreiben
}

void mouseReleased() {
  //Erzeugen eines eindeutigen Namen
  //Das tif-Format speichert alle Pixel
  String fileName = "myPic"+year()+month()+day()+hour()+minute()+second()+".tif";

  //Speichern des aktuellen Bildes
  save(fileName);
}
```

b.) Lege ein Blumen-Bilddatei in den Projektordner und passe die setup()-Methode so an, dass das Bild angepasst wird.
Durch Klicken bzw. Loslassen der Maus wird ein zusätzliches Bild in deinem Projektordner gespeichert. Kontrolliere dies.

c.) Mit Hilfe der Funktion color(r,g,b) kann jedem Pixel ein neuer Rot, Grün oder Blau-Wert zugeordnet werden. Ergänze die draw() so, dass dein Blumenbild nur in Rot angezeigt wird.
Tipp: Die Codezeile „pixels[idx] = color(red(img.pixels[idx]),0,0);“ weist dem idx-ten Pixel nur einen Rot-Wert zu. Grün und Blau werden auf 0 gesetzt. Die Funktion red() gibt den Rotanteil mit Werten aus [0,255] zurück. Hierzu wird auf das idx-te Pixel des Original-Bildes zurückgegriffen: img.pixels[idx].

d.) Verändere deine draw()-Methode nacheinander so, dass die drei Varianten deines Bildes im „Warhol"-Stil erzeugt werden.

e.) Mit der Funktion brightness(img.pixels[idx]) wird die Helligkeit eines Pixels mit Werten aus [0,255] abgerufen. Damit lassen sich Schwarz-Weiß-Bilder erzeugen.

Benutze hierfür entweder das Codefragment `color(brightness(img.pixels[idx]))` oder `color((r+g+b)/3)`, wobei die drei Variablen r,g,b die jeweiligen Rot, Grün, Blau-Werte des Pixels sind:

```
color pixelColor = img.get(x,y);
float r = red(pixelColor);
float g = green(pixelColor);
float b = blue(pixelColor);
```

f.) Um die Helligkeit besser anpassen zu können, soll die Helligkeit mit der Maus gesteuert werden. Hierzu wird der x-Werte der Maus [0,width] auf das Intervall [-255,255] projiziert. Gilt mouseX = width/2, die Maus ist also in der senkrechten Mitte der Bildes, so bekommst du das Bild aus e.) . Ist die Maus ganz links, so sollte ein ganz schwarzes, ganz Rechts ein weißes Bild erzeugt werden.

Tipp:

```
float curBrightness = brightness(img.pixels[idx]);
float curMouseX = mouseX;
float curSummand = map(curMouseX,0,width,-255,255);
pixels[idx] =  color(curBrightness+curSummand);
```

g.) 

Erstelle einen Filter, der ein Schwarz-Weiß-Bild erzeugt. Ist die Helligkeit eines Pixels größer als 100, so soll das Pixel weiß werden sonst schwarz.

h.) Wandle dein Bild im Stile des Pointilismus um. Male hierfür an jedem 10. Pixel einen Kreis (r=10) mit der Farbe des Pixels.

Tipp: Hierfür muss das Bild nicht mit loadPixels geladen werden. Mit Hilfe der Funktion get(x,y), kann man auch direkt die Farbe eines Pixels abrufen: color pixelColor = img.get(x,y);
Benutze zwei ineinander geschachtelte for-Schleifen, um das ganze Bild umzuwandeln. Versuche auch Quadrate aus.

i.) Eine Abwandlung von e.) sind Bilder im Sepia-Stil. Verwende hierzu folgenden Berechnungsansatz:

```
float newR = 0.393*r + 0.769*g + 0.189*b;
float newG = 0.349*r + 0.686*g + 0.168*b;
float newB = 0.272*r + 0.534*g + 0.131*b;
//Wertebereich einschränken auf [0,255]
newR = constrain(newR,0,255);
```

j.) 

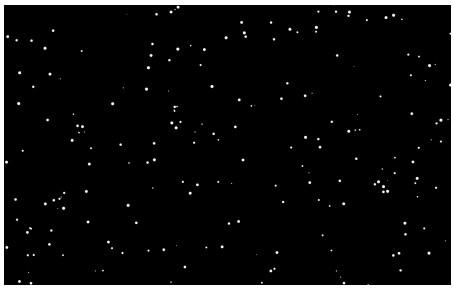
Bei vielen Foto-Apps gibt es aber nicht nur Farbeffekte, sondern auch Maskierungen. Hierbei wird jeder Pixel in Abhängigkeit vom Abstand zum Bildmittelpunkts in seiner Helligkeit verändert.
Der Radius des hellen Bereichs sollte durch mouseX veränderbar sein (vgl. Teilaufgabe f.)).

Aufgabe 3.1.3: Bildmanipulation

Verändere ein Bild so, dass du mindestens zwei der obigen Effekte benutzt und mit einander kombinierst. Ergänze eventuell noch andere Effekte: Miniatur, Unschärfe. Nimm deine Foto-App oder ein anderes Fotobearbeitungsprogramm als Anregung.

Kapitel 3.2 Eigene Felder

Aufgabe 3.2.1: Schneefall



a.) Erstelle einen Bildschirmschoner, der Schneefall simuliert. Definiere hierfür eine Klasse „Snowflake“ in einem neuen Projekt „Snowfall“. Die Schneeflocken sollen weiße Kreise mit einem Durchmesser d zwischen 2 und 10 sein. Die Geschwindigkeit vY soll umgekehrt proportional zum Durchmesser sein, d.h. je größer die Flocke, desto langsamer fällt sie. Lass die Flocken dabei nicht geradlinig fallen, sondern variiere die Bewegung durch leichtes hin und her bewegen (variieren von vX). Ist eine Schneeflocke außerhalb des unteren Bildschirmrandes, so soll sie wieder von oben starten, aber neu initialisiert werden, d.h. sie bekommt eine neue Größe und neue Geschwindigkeiten.
Hinweis: Achte darauf, dass die Schneeflocken von Beginn an über den ganzen Bildschirm verteilt sind.

b.) Ergänze deine Schneeflocken um das Attribut „lifetime“. Bei jedem frame-Durchlauf soll dabei dieses Attribut um 1 erniedrigt werden. Ist lifetime==0, dann ist die Lebenszeit der Schneeflocke abgelaufen (geschmolzen); sie wir daraufhin neu initialisiert.

c.) Ergänze Wind. Definiere hierfür eine globale Variable windVX, die mit der mouseX-Position gesteuert werden kann: Steht die Maus in der Mitte des Bildschirms, so ist windVX=0. Ist die mouseX==0, so bläst der Wind maximal von links; für windVX==width kommt der Wind maximal von rechts. Spätestens jetzt muss ergänzt werden, dass die Schneeflocken bei horizontalem Verlassen des Bildschirms neu positioniert werden. Eventuell einfach auf die andere Seite bringen durch ändern der x-Koordinate.

d.)* Ergänze einen sichelförmigen Mond. Passe die „Helligkeit“ der Schneeflocken umgekehrt proportional zum Abstand einer Schneeflocke zum Mondmittelpunkt an.

Aufgabe 3.2.2: Sortieren mit BubbleSort

Ziel dieser Aufgabe ist neben dem weiteren Üben der Zählwiederholung zusammen mit Feldern das Anwenden des BubbleSort-Sortier-Algorithmus.

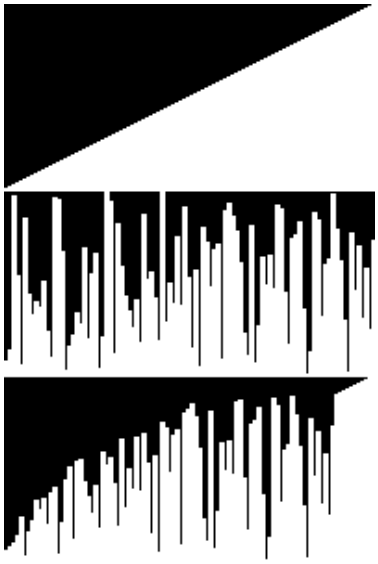
a.) Gegeben sei ein Feld „zahlen“ der Länge 100, das mit lauter verschiedenen natürlichen Zahlen gefüllt ist. Benutze hierfür eine Methode init(). Diese Methode init() belegt das Feld mit den Werten 1 bis einschließlich 100.
Erzeuge dann innerhalb der setup()-Methode ein Fenster der Größe size(200,300) und lass Rechtecke so in das Fenster zeichnen, dass das obere Drittel mit Rechtecken der Breite 2px gezeichnet wird. Die Höhe des Rechtecks entspricht den Einträgen im „zahlen“-Feld.
Beachte, dass die Rechtecke von „unten nach oben“ gezeichnet werden:

```
rect(idx*2,100,2,-lines[idx]);
```

b.) Damit später etwas zu sortieren ist, muss das Feld zunächst gemischt werden. Hierfür lässt sich ebenfalls der BubbleSort-Algorithmus benutzten.
Notiere den Mischalgorithmus im linken Bereich in Form eines Struktogramms und im rechten Bereich in Java.

Struktogramm	Java

c.) Lass das gemischte Feld im mittleren Bereich anzeigen in dem du die setup()-Methode entsprechend ergänzt.



d.) Das Sortieren des Feldes wird nun im unteren Drittel animiert dargestellt. Implementiere hierzu folgende Schritte in der draw()-Methode:

- Übermalen des unteren Drittels mit einem schwarzen Rechteck.
- Durchführung eines Sortiervorgang über das ganze Feld.
- Zeichnen der Rechtecke.

Eventuell muss die frameRate auf 10 gesenkt werden, um die Animation besser sichtbar zu machen.

e.) Notiere hier den BubbleSort-Algorithmus mit Hilfe zweier verschachtelter Zählwiederholungen-Schleifen in Java:

Der BubbleSort-Algorithmus

Kapitel 3.3 Komplexe Aufgaben

Programmierhinweise

Einige wichtige Hinweise vorneweg, um sowohl dir das Programmieren zu erleichtern, als auch deinem Lehrer es einfacher zu machen, eventuelle Fehler zu finden:

1. **Ordentliches Einrücken des Quelltextes**
   Achte darauf, dass du alle „Rümpfe", also alles, was in geschweiften Klammern steht, eingerückt wird.

2. **Position der schließende Klammer**
   Die schließende Klammer steht exakt unterhalb des ersten Zeichens in derjenige Zeile, in der die korrespondierende öffnende Klammer steht.

3. **Benutze sprechende Namen für Variablen**
   Hierdurch wird der Programmiercode einfacher lesbar: Für dich, deinen Nachbarn oder auch deinen Lehrer.

4. **Lerne die Grundbegriffe**
   Alles, was bis jetzt gemacht wurde, sollte bekannt sein. Dir sollten dabei nicht nur Programmierstrukturen (Definition von Klassen, Variablen, Wiederholungen, ...) bekannt sein, sondern auch die bisher benutzten Befehle von processing (mouseX, mouseY, ellipse(), dist(), ...).
   Wenn du dir noch unsicher bist, so schreibe dir im Rahmen einer Hausaufgabe eine Zusammenfassung.

5. **Für eine „Best-of"-Datei oder -Buch**
   So, wie du immer wieder als Schreiner mit den gleichen Werkzeugen arbeitest und dennoch ganz unterschiedliche Dinge herstellen kannst, wirst du beim Programmieren auch immer wieder die gleichen Dinge benutze und ganz unterschiedliche Programme erstellen. Halte daher wichtige Programmfragmente fest. Wenn du das in einer Datei machst, so kannst du sie per Copy-and-Paste leicht übernehmen – hast sie aber vielleicht nicht immer dabei.
   Wenn du nicht weißt, was du genau aufschreiben solltest, so frage deinen Lehrer.

Aufgabe 3.3.1: Ameisen-Simulation

Es soll ein Programm zur Simulation von „Ameisen“ geschrieben werden, die sich auf „Futter“ zu bewegen.

a.) Erstelle ein neues Projekt „AntSimulation“. Das processing-Fenster soll Bildschirmgröße haben und schwarz sein. Die Framerate soll 30 betragen.

Es wird wieder vom Spezialfall (nur 1 Ameise) zur Verallgemeinerung (n Ameisen) programmiert. So lassen sich Fehler am ehesten vermeiden.

b.) Erstelle in einem neuen Reiter „Ant“ des Projekts die Klasse „Ant“ mit den Attributen x,y,v und d für die Position (x|y), die Geschwindigkeit v (Zufallswert zwischen 0,1% und 3%). Der Durchmesser d soll 5 betragen, um einen eindeutigen Mittelpunkt zu haben. Die Position soll zufällig im Fenster gewählt werden.
Ergänze die Methode show(): An der Position (x|y) soll eine weiße Kreisfläche mit dem Durchmesser d gezeichnet werden.
Definiere eine Ameise mit dem Namen „ant1“ im Reiter „AntSimulation“ und lass die Ameise innerhalb der Methode draw() zeichnen: „ant1.show()“ .

c.) Ergänze in der Klasse „Ant“ eine Methode „update(float targetX, float targetY)“. Hier bei soll die Ameise den Abstand zum Ziel (engl. target) für die horizontale und vertikale Richtung berechnen. Die Aktualisierung der x- bzw. y-Position soll dann die aktuelle Position plus den Bruchteil v der jeweiligen Richtung sein:

```
x += (targetX-x)*v;
y += (targetY-y)*v;
```

Ergänze in der Methode draw() den Aufruf „ant1.update(mouseX, mouseY)“. Beschreibe das Verhalten der Ameise.

d.) Passe dein Programm nun so an, dass 200 Ameisen zufällig im Fenster erzeugt und gleichzeitig animiert werden.

e.) Anstelle des Mauszeigers soll nun Futter das Ziel der Ameisen sein. Definiere analog zu den Ameisen hierfür eine Klasse „Target“ mit den Attributen x,y und d=20. Die Position des Futters soll beim Programmstart zufällig gewählt sein.

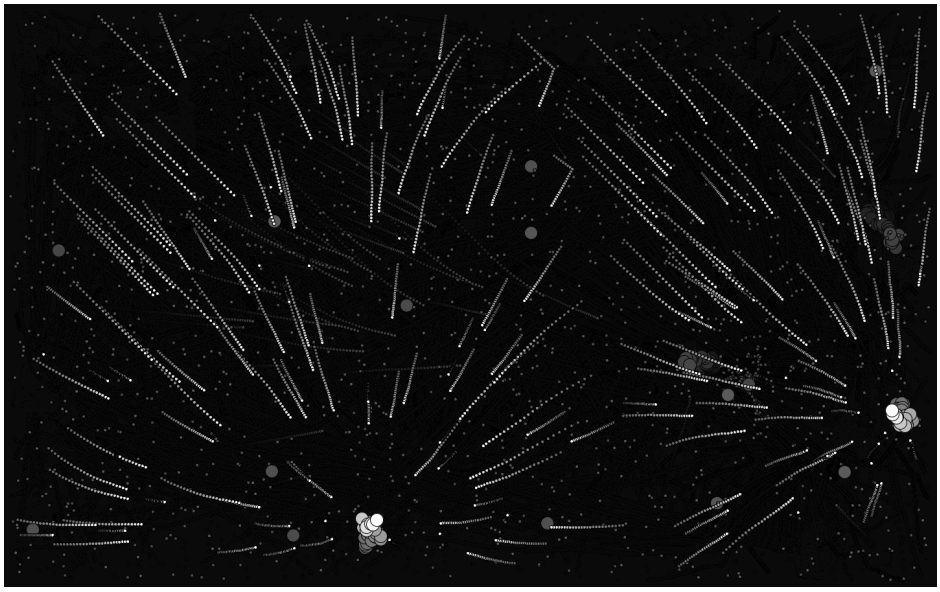
f.) Wenn das Futter kein Futter ist, sondern Gift, dann sterben die Ameisen, wenn Sie das Gift berühren. Passe dein Programm entsprechend an:

Immer, wenn die Ameisen das Ziel erreichen, werden sie zerstört. Dafür taucht aber eine neue Ameise auf.
Tipp: Statt das Objekt wirklich zu zerstören, ist es einfacher bei Berührung mit dem Gift (benutze die bereits bekannte Funktion dist(x1,y1,x2,y2)) die x-, y-, und v-Attribute neu zu initialisieren.

g.) Statt einem Futter sollen nun mindestens 2 Futtertröge bzw. Giftköder ausgelegt werden. Passe dein Programm entsprechend an.

Ab hier kommen noch Ideen für Weiterentwicklungen:

h.)* Animiere die „Targets" so, dass sie sich ein wenig bewegen bzw. zappeln.



i.)** Definiere einen Jäger durch ein Klasse „Hunter", der sich über das Fenster bewegt und immer versucht, die nächstgelegene Ameise zu fressen. Gib dem Jäger eine andere Farbe, damit besser unterschieden werden kann.

j.)** Versuche das Verhalten der Ameisen und Jäger sinnvoller zu programmieren.

- Die Ameisen versuchen dem Jäger auszuweichen.
- Der Jäger wird mit jeder gefressenen Ameise größer und langsamer.
- Das Ameisen laufen nicht geradlinig sondern „zappeln" leicht.
- usw.

Aufgabe 3.3.2: „Agar.io“

Das Online-Spiel „agar.io“ ist den meisten Schülern im Jahr 2019 bekannt. Eine vereinfachte Version soll nun programmiert werden.[3]

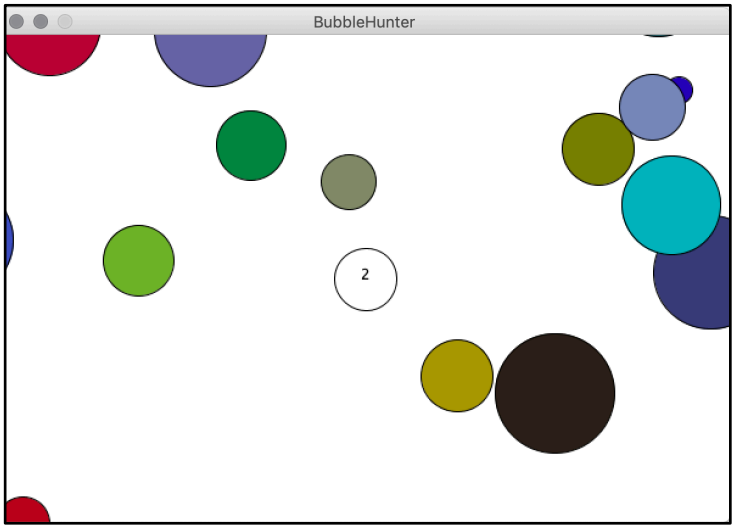
a.) Erstelle ein neues Projekt „BubbleHunter“. Hintergrundfarbe des bildschirmfüllenden Spiels soll diesmal Weiß sein.
Erstelle eine Klasse „Bubble“ analog zu den Ameisen mit den bekannten Attributen x,y,d sowie colorR, colorG,colorB, da die Bubbles bunt werden sollen. Ergänze auch die benötgte Methode show().
Initialisiere die Bubbles durch den Konstruktor so, dass auch außerhalb des sichtbaren Bereichs Bubbles erzeugt werden.

```
x = random(-2*width,2*width);
y = random(-2*height,2*height);
```

Zudem soll der Durchmesser zwischen 10 und 100 liegen.
Hinweis: Es muss nicht dafür gesorgt werden, dass die Bubbles vom Rand abprallen, da die Animation in diesem Spiel anders ablaufen wird.

b.) Implementiere eine Klasse „Player“. Die Spielfigur „player“ ist ein Kreis in der Mitte des Bildschirms mit dem Anfangsdurchmesser 50px und einer weißen Farbe. Innerhalb des Kreises soll der Punktestand des Spielers, welcher in einem weiteren Attribut „points“ mitgeführt wird, angezeigt werden.

c.) Erzeuge und verwalte 100 Bubbles mit Hilfe eines entsprechenden Arrays.



[3] Einige Aspekte des Spiels wurden bereits bei der Ameisen-Simulation benutzt und können übernommen werden. Müssen aber angepasst werden.

Eine bei Spielen häufig vorzufindende Animationstechnik, wie sie vor allem bei vielen Jump-and-Run Spielen vorzufinden ist, ist die Animation des Hintergrundes anstelle der Spielfigur. Dies wird erreicht, in dem in der draw()-Methode folgende Zeile ergänzt wird:

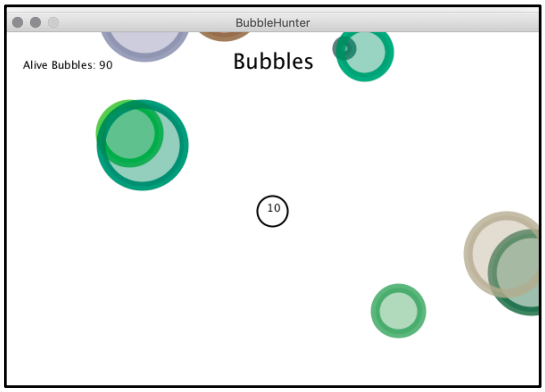
```
translate(width/2-player.x, height/2-player.y);
```

Der Ursprung bzw. der Hintergrund wird auf den Mittelpunkt der Spielfigur „player" verschoben (engl.: translate).
Allein das Player-Objekt bekommt eine update()-Methode mit nebenstehendem Inhalt.

```
void update() {
  float dx = (mouseX-width/2)*0.01;
  float dy = (mouseY-height/2)*0.01;
  x+=dx;
  y+=dy;
  d*=0.9995;
}
```

Um das Spiel interessanter zu machen, verringert sich der Durchmesser der Spielfigur pro Frame um 0,05%. So wird der Spieler gezwungen zu handeln.

d.) Ergänze in der show()-Methode der Bubble-Klasse das Kollisionsverhalten:



- Trifft die Spielerfigur auf einen Bubble mit kleinerem Durchmesser, so verschwindet die Blase und die Fläche der Spielfigur wächst um den Anteil der zerplatzten Blase.
- Um das Zerplatzen der Blase zu realisieren, bekommt jede Bubble ein Attribut dead. Gilt dead==true, so wird die Blase nicht mehr gezeichnet.

e.)* Weiter Ergänzungen für das Spiel könnten wir folgt aussehen:

- Ergänze abschließend noch eine Spielstand und einen Titel.
- Ist der Durchmesser des Spielers zu klein wird ein „Du bist verhungert" oder ähnliches eingeblendet.
- Klickt der Benutzer mit der Maus, so wird das Spiel neu gestartet.
- Ergänze mindestens einen Jäger, der immer versucht den Spieler zu fangen. Das Verhalten soll dabei identisch zum Spieler sein in Bezug auf die Kollisionen.

KAPITEL 4: Diagramme

Nachdem nun die grundlegenden Programmierstrukturen kennengelernt wurden und an zusehends komplexen Programmen geübt wurden, ist es nun an der Zeit Programme strukturiert anzugehen. Das bedeutet, dass jetzt vor dem eigentlichen Programmieren das Modellieren und das Erstellen eines Programmentwurfs steht.
Modelliert wird dabei mit Hilfe unterschiedlichster Diagrammarten. Jede Diagrammart stellt dabei einen speziellen Aspekt in den Vordergrund:

	Diagrammart	**Ziel des Diagramms**
1.	Struktogramm (Programmablaufplan[4])	Sprachunabhängige Darstellung von Algorithmen bzw. Programmcodes
2.	Klassendiagramm	Darstellung des Zusammenhangs und der Art der zu verwalteten Programmelemente
3.	Objektdiagramm	Darstellen eines aktuellen Zustands eines Programms
4.	Zustandsdiagramm	Darstellung aller möglichen Zustände eines Programms mit ihren auslösenden Ereignissen und eventuellen Nebeneffekten
5.	Sequenzdiagramm	Darstellung eines zeitlichen Ablaufs

Die ersten drei Diagramme sollten aus den vorangegangenen Jahrgangsstufe bekannt sein.

[4] Nicht in jedem Lehrplan enthalten. Eines von beiden sollte jedoch bekannt sein.