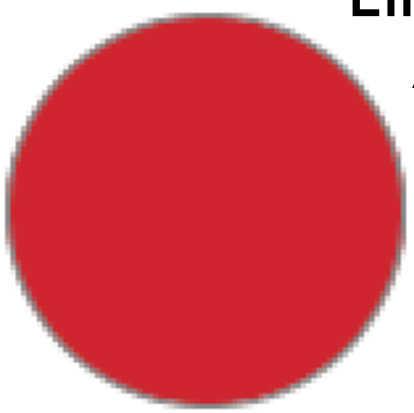
Programmieren lernen mit processing



Ein Arbeitsheft

Autor: Ingo Bartling

CC BY-NC-SA 4.0

Inhaltsverzeichnis

Vorwort zur Benutzung **4**

KAPITEL 1: Allgemeines **5**

Was ist Programmieren? **5**

Was ist ein Algorithmus? **5**

Aufgabe 1.1.1: Rezept 5

Was ist eine Computersprache? **6**

Was ist Lernen? **6**

Was ist processing? **7**

KAPITEL 1.2: Einfache Datentypen und -strukturen **8**

1.2.1 Primitive Datentypen **8**

Aufgabe 1.2.1: Datentypen bei Datenbanken 8

Aufgabe 1.2.2: Datentypen in Java 8

Aufgabe 1.2.3: Superhelden-Klasse 9

Aufgabe 1.2.4: Schüler-Klasse 9

1.2.2 Referenz-Datentypen **9**

Aufgabe 1.2.5: Klasse-Definition 9

Aufgabe 1.2.6: Methode definieren 9

Aufgabe 1.2.7: Standardkonstruktor 9

Aufgabe 1.2.8: Methodenkopf & -rumpf 9

KAPITEL 1.3: if-else **10**

Aufgabe 1.3.1: Struktogramme 10

Aufgabe 1.3.2: Klasse mit Nicht-Standardkonstruktor 10

Aufgabe 1.3.3: if-else 10

Aufgabe 1.3.4: Die Klasse Kreis 10

KAPITAL 1.4: Was du bis jetzt können musst **11**

Allgemeines zu Java 11

Die Grundbegriffe des Programmierens 12

Kontrollstrukturen 15

KAPITEL 1.5: Graphik in Processing **16**

Aufgabe 1.5.1: Farben 16

Aufgabe 1.5.2: setup() und draw() 16

Aufgabe 1.5.3: Das Koordinatensystem 16

Aufgabe 1.5.4: Grundlagen des Zeichnens 17

Kapitel 1.6: Komplexere Programmieraufgaben **18**

Beispiel an der Aufgabe „Mondrian 2“ 18

Aufgabe 1.6.1: Abschlussaufgabe 1 (Strichcode) 19

Aufgabe 1.6.2: Abschlussaufgabe 2 (Mondrian 1) 20

Aufgabe 1.6.3: Abschlussaufgabe 3 (Mondrian2) 21

Aufgabe 1.6.4: Was du bis jetzt kennen solltest 22

KAPITEL 2: Animationen **23**

Kapitel 2.1: Bewegung mit konstanter Geschwindigkeit **23**

Aufgabe 2.1.1: Klasse Ball 24

Aufgabe 2.1.2: Bewegung mit konstanter Geschwindigkeit 24

Aufgabe 2.1.3: Abprallen am Rand 25

Kapitel 2.2: Bewegung mit konstanter Beschleunigung **27**

Aufgabe 2.2.1: Bewegung mit konstanter Beschleunigung 27

Aufgabe 2.2.2: Energieverlust 27

Aufgabe 2.2.3: Schlieren-Effekt 27

Aufgabe 2.2.4: Bildschirmschoner 28

Aufgabe 2.2.5*: Fliegender Text 28

KAPITEL 3: Komplexere Strukturen 29
Kapitel 3.1: for-Schleife und Arrays 29
Aufgabe 3.1.1: Fingerübungen für Zählwiederholungen 29
Aufgabe 3.1.2: Statische Bildmanipulation 31
Aufgabe 3.1.3: Dynamische Bildmanipulation 35
Aufgabe 3.1.3: Maskierung 37
Aufgabe 3.1.4: „Verpixler“ 37
Aufgabe 3.1.5: Bildmanipulation (Abschlussaufgabe) 37
Kapitel 3.2 Eigene Felder 38
Aufgabe 3.2.1: Schneefall 38
Aufgabe 3.2.2: Sortieren mit BubbleSort 39
Aufgabe 3.2.3: Andere Sortieralgorithmen 41
Aufgabe 3.2.4: Kleine Anwendungsaufgaben 42
Aufgabe 3.2.5: Anwendungsaufgaben mit eigenen Klassen 43
Aufgabe 3.2.6: Spielereien 44
Kapitel 3.3 Komplexe Aufgaben 45
Programmierhinweise 45
Aufgabe 3.3.1: Ameisen-Simulation 46
Aufgabe 3.3.2: „Agar.io“ 48

KAPITEL 4: Spiele 50
Das Spiel „Bubbles“ 51
`Die Hauptdatei` 51
`Anzeige` 51
`Anzeige` 51
`StartScreen` 52
`PlayingScreen` 52
`Button` 52
`EndScreen` 53
`GameLogic` 53
`Bubble` 54

KAPITEL 4.1: Zustands-Diagramme 55
Aufgabe 4.1.1: Definition 55
Aufgabe 4.1.2: Smartphone 55
Aufgabe 4.1.3: Getränkeautomat 56
Aufgabe 4.1.4: Software 57

KAPITEL 4.2: Sequenz-Diagramme 58

Vorwort zur Benutzung

Dieses Arbeitsheft wurde parallel zum Unterricht im Schuljahr 2018/19 in zwei Klassen der zehnten Jahrgangsstufe entwickelt und getestet. Und es wird sich auch in den folgenden Jahren bestimmt noch weiterentwickeln.

Es ist kein Buch zum Selbstlernen. Aus meiner eigenen Erfahrung aus dem Unterricht heraus, ist es von Vorteil, wenn einem ein Lehrer die Elemente erklärt. Was den Schülern abgeht, sind dann die passenden Übungen. Frei nach der Melodie „Ich würde gerne programmieren, ich weiß aber nicht was."

Dieses Heft ist daher als ***Begleitlektüre*** zum normalen Unterricht zu verstehen und soll keinen Unterricht ersetzten, sondern begleiten. Daher gibt es auch keine Musterlösungen. Die Versuchung, einfach nur die Musterlösung zu präsentieren, wäre vielleicht doch einfach groß.

Ach ja, warum processing. Deswegen:

1. Es kostet nichts: processing.org.
2. Mehrere Programmiersprachen sind möglich (JAVA, Python, JavaScript) inklusiver mehrerer Plattformen.
3. Das „Hello world"-Programm ist ein 1-Zeiler: println(„Hello World);
4. Die Orientierung an Grafik ermöglicht motivierende Ergebnisse.
5. Spiele, die immer sehr motivierend sind, sind gut umzusetzen.

Gerade die letzten beiden Punkte sind, denke ich, sehr reizvoll für die Mehrheit der Schüler. Zumindest habe ich das immer so erlebt.

Ingo Bartling (2019)

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
boolean	true	Wahrheitswert	true, false
float	3.141f	Kommazahl	+/-1,4E-45 bis +/-3,4E+38
double	3.14	Kommazahlen	+/-4,9E-324 bis +/-1,7E+308
int	-2	ganze Zahlen	-2.147.483.648 bis 2.147.483.647
char	'a'	Ein einzelnes Zeichen	-
String	"Hallo"	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

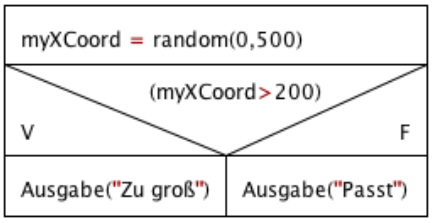
Markiere bei den folgenden Methoden den Methodenkopf in Blau und den Methodenrumpf in Grün.

```
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!




```
myXCoord = random(0,500);
if (myXCoord>200) {
        println("Zu grop");
} else {
        println("Passt");
}
```

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)
```

KAPITAL 1.4: Was du bis jetzt können musst

Allgemeines zu Java

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

.java-Dateien sind das Drehbuch, .class-Dateien der Film

☐ Den Begriff kompilieren erklären können.

Durch das Kompilieren wird aus dem Drehbuch die „Filmrolle" erstellt.

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

Nur der Film kann später von anderen auch angeschaut bzw. benutzt werden.

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

Da es nicht für jedes Betriebssystem einen Projektor gibt.

☐ Das Programm processing und processing.org kennen.

☐ Ein- und mehrzeilige Kommentare einfügen können und wissen, was und wie kommentiert werden muss.

Ein einzeiliger Kommentar beginnt mit //... , ein mehrzeiliger wird „geklammert" durch /*...*/ .

Es wird das Ziel jeder Methode angegeben, außer diese erschließt sich vollständig aus dem Methodennamen.

Es wird jedes Attribut und jede Variable erklärt bei der Deklaration erläutert, außer wenn sich dies vollständig aus dem Namen erschließt.

Der Methodenrumpf wird durch einzeilige Kommentare, die als Zwischenüberschriften dienen, gegliedert.

☐ Das Wort „implementieren" erläutern können.

„umsetzen", hier im Sinne von „programmieren". Wichtig für schriftliche Tests.

Die Grundbegriffe des Programmierens[1]

☐ Eine Klasse deklarieren können

class – EigenerName – { – }

☐ Attribute deklarieren können

Datentyp[1] – eigenerName ;

☐ Eine Variable deklarieren können

Datentyp – eigenerName ;

☐ Alle wichtigen, primitive Datentypen angeben und erklären können

int (Ganze Zahlen), double (Kommazahlen: 3.14), String (Zeichenketten),

char (1 Zeichen), boolean (Wahrheitswert: true, false)

String ist genau genommen kein primitiver Datentyp, da er aus char gebildet

wird. Daher auch das große S bei String.

Es gibt noch mehr primitiven Datentypen, aber diese genügen am Anfang.

☐ Eine Variable mit eine Startwert initialisieren können.

```
eigenerName - = - Wert ;
```

Beispiel: `alter = 18; name = „Max“;`

Variablen, die nur innerhalb von Methoden auftauchen sollten bei der

Definition bereits mit einen Startwert initialisiert werden, um Fehler zu

vermeiden.

Beispiel: `Datentyp – eigenerName - = - Wert ;`

☐ Zeichenketten (mit Variablen-/Attributwerten) verknüpfen können

Beispiel: `String antwort = „Ich heisse: “ + name;`

[1] in Java

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

Sichtbarkeit oder Zugriffsrecht: Bei public darf jedes Objekt auf das Attribut / Methode zugreifen. Bei private nur das Objekt selbst.

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

Alles, was Teil von etwas anderem ist, wird um 1 Tabulator eingerückt.

Klassennamen beginnen immer mit einem Großbuchstaben, Methoden (Funktionen und Prozeduren) und Attribute sowie Variablen immer mit einem Kleinbuchstaben.

Da Klassen- und Methodennamen keine Leerzeichen (keine Sonderzeichen) enthalten dürfen, aber sprechende Namen wichtig sind, beginnt jedes innere Wort eines Namens mit einem Großbuchstaben: alleDatenAusgeben()

☐ Die Aufgabe eines Konstruktors angeben können

Ein Konstruktor erzeugt gemäß des Bauplans (Klasse mit Attributen, Methoden) ein Objekt der Klasse und sollte alle Attribute initialisieren.

☐ Standardkonstruktoren .....

Ein Standardkonstruktor hat keine Parameterliste und initialisiert alle Attribute mit fest vorgegebenen Startwerten.

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

Diese Konstruktoren initialisieren ebenfalls alle Attribute, allerdings benutzen sie dafür die Wert aus der Parameterliste.

☐ Die Werte der Eingangsparameter in den Attributen speichern können

this[1].attributName = parameterWert ;

Beispiel: this.alter = neuesAlter;

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

☐ Wissen, was ein Methodenkopf ...

☐ .... und was ein –rumpf ist.

```
public void alterAusgeben() {

  System.out.println(alter)
}
```

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

Datentyp – methodenName (ev. Parameterliste)

☐ Den Unterschied zwischen Prozedur und Funktion benennen können

Eine Prozedur liefert keine Antwort an den Aufrufenden zurück, daher ist der Rückgabewert void. Alltagsbeispiel: vonDerTafelAbschreiben() ist eine Prozedur von Schüler, da sie keine Antwort an den auffordernden Lehrer zurückgeben. Aber Sie produzieren einen Hefteintrag.

Eine Funktion gibt dem Aufrufenden eine Antwort. Die Art der Antwort (Datentyp) muss dabei vorher festgelegt werden. Alltagsbeispiel wäre bei einem Schüler die Funktion entschuldigungGeben() bei der der Schüler in Form einer Zeichenkette (String) eine Entschuldigung an den Lehrer zurückgibt.

☐ Eine Ausgabe in das Editorfenster schreiben können

Beispiel: println(„Hallo Welt“);

Ohne Zeilenumbruch am Ende: print(„Hallo Welt“);

☐ Ein Objekt im Quelltext erzeugen können

new – Konstruktorname – (– Werteliste –)

Beispiel:

Hausaufgabe neueHA = new Hausaufgabe(„M“, „20161101“, „B: S 12/1“);

☐ Ein Methode eines Objektes im Quelltext aufrufen können.

wer . was (Werteliste);

Beispiel: String alleInfos = neueHA.getInfos();

neueHA.setDate(„20161101“);

Kontrollstrukturen

☐ Eine bedingte ein/zweiseitige Verzweigung angeben können.

```
if – ( – Ja/Nein-Frage – ) – {
        //Ja-Fall
} else {
        //Nein-Fall; der nicht kommen muss
}
```

☐ Du kennst alle Vergleichsoperatoren

>, <, <=, >=, !=, ==

☐ Alle booleschen Operatoren an einem Beispiel erklären können

Und && bzw. oder || oder !

Beispiel

```
if (alter >= 16 && alter <18) {
        println(„Bier ist erlaubt.“);
}
```

KAPITEL 1.5: Graphik in Processing

Aufgabe 1.5.1: Farben

In welcher Einheit wird der Bildschirm unterteilt? Pixel

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

strokeWeight(x) ist die Liniendicke; x>0

fill(x) ist die Füllfarbe; 0<=x<=255 in Grauwerten (Schwarz bis Weiß)

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

Farbaddition der Farben Rot, Grün, Blau mit Werten zw. 0 und 255 (max)

fill(255,0,255) entspricht Lila

Aufgabe 1.5.2: 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?
	1
		Diese Methode wird vor dem Anzeigen des Fensters einmal aufgerufen und dient der Vorbereitung des Programms.

void draw() {
 	
}
	Wie oft wird diese Methode aufgerufen?
	Unendlich
		Diese Methode wird nach dem Anzeigen des Fensters unendlich oft aufgerufen. Die Durchlaufrate pro Sekunde wird durch frameRate(x) bestimmt

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

Aufgabe 1.5.3: Das Koordinatensystem

Erläutere das Koordinatensystem (Ursprung, Achsen) in processing!

Der Ursprung liegt oben links. Die x-Achse verläuft nach rechts.

Die y-Achse nach unten.

Aufgabe 1.5.4: Grundlagen des Zeichnens

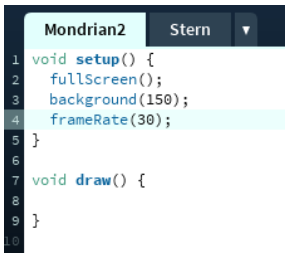
Zur Darstellung von Objekten in processing musst du ein paar wichtige Methoden und Funktionen kennen. Ergänze die Tabelle jeweils um eine knappe Erklärung oder gib den Befehl an. Beantworte auch die Fragen. Informationen findest du auf processing.org.

Befehl	Frage	Erläuterung
size(600,400)		Ein Fenster der Größe 600x400 öffnet sich.
fullScreen()		Das Zeichenfenster soll genauso groß sein wie der Bildschirm.
frameRate(60)	Wie hoch ist der Standardwert? 60	60 frames pro Sekunden, wenn möglich
background(0)		Der Bildschirm wird schwarz übermalt.
rect(100,200,300,50)	Auf welche Ecke des Objekts bezieht sich die Positionsangabe? linke, obere Ecke	Es wird ein Rechteck an der Position (100|200) mit der Breite 300 und Höhe 50 gezeichnet.
ellipse(20,50,100,100)		Ein Kreis mit M(20|50) und r=100px wird gezeichnet.
line(100,200,300,400)		Es soll eine Linie vom Punkt (100|200) zum Punkt (300|400) gezogen werden
mouseX mouseY		x-Koordinate, y-Koordinate des Mausklicks
height width		Höhe und Breite des Fensters

Kapitel 1.6: 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 1.6.1: Abschlussaufgabe 1 (Strichcode)

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

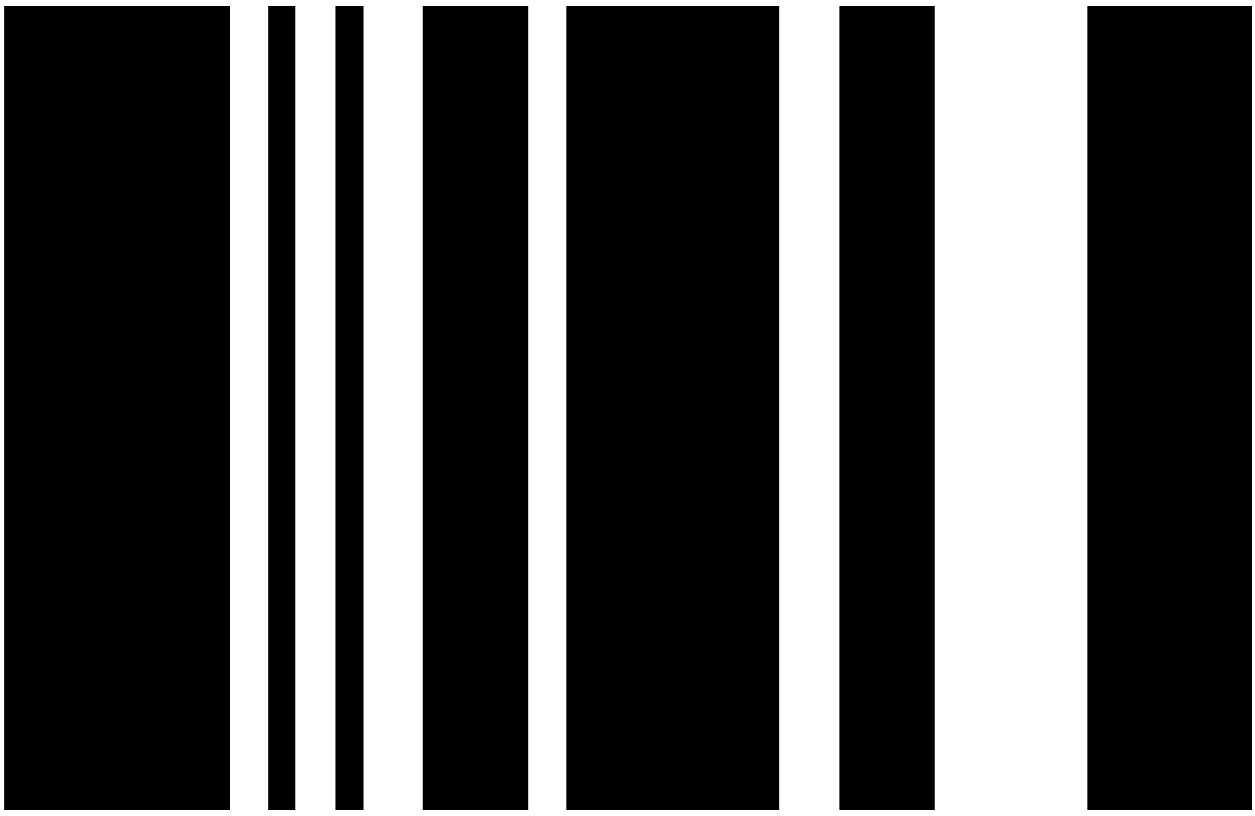
b) Erzeuge ein Fenster in Bildschirmgröße mit schwarzem Hintergrund.

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)<8*
random(1,10) liefert einen Wert aus dem Intervall [1,10[



Aufgabe 1.6.2: 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.

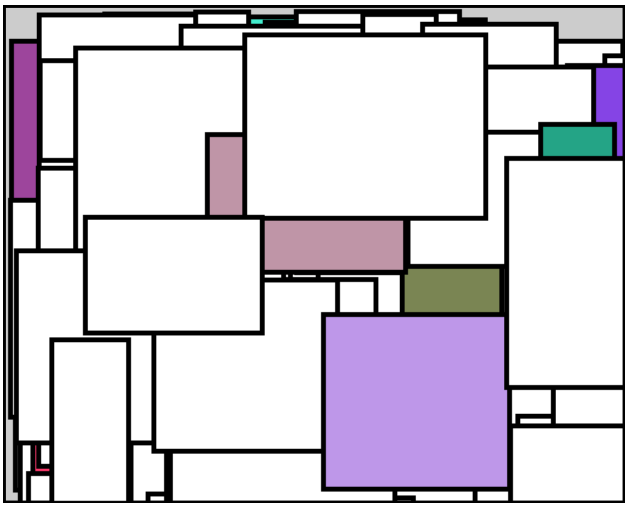
d) Durch Linksklick mit der Maus soll das Bild gelöscht werden.

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

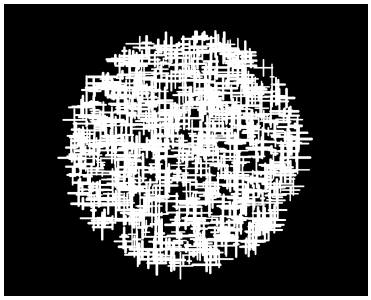
f) Statt Rechtecken werden Quadrate mit einer Zufallsgröße erzeugt.

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 1.6.3: 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.

Aufgabe 1.6.4: Was du bis jetzt kennen solltest

Bei den vorangegangenen Aufgaben hast du einige neue processing-Befehle kennengelernt. Fülle die Tabelle aus, um eine Art Nachschlagewerk zu erhalten. Hilfe und weitere Informationen findest du auf processing.org.

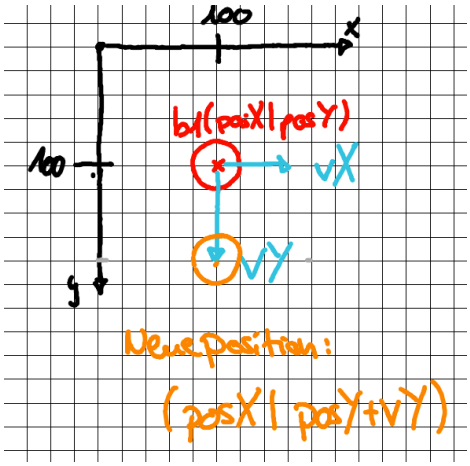
Befehl	Frage	Erläuterung
random(10)	Welchen Datentyp liefert die Funktion zurück? float	Eine Zufallszahl aus dem Bereich [0;10[
random(10,21)		Eine Zufallszahl zwischen [10,20[
int()		Wandelt einen einfachen Datentyp in eine ganze Zahl um.
int(random(1,7))	Zeichne das Datenflussdiagramm:	Es wird eine Zufallszahl zwischen 1 und 6 zurückgegeben (Würfeln)
map(mouseX,0,width,1,100)		Der Wertebereich von mouseX wird auf [1,100] abgebildet werden.
dist(x1,y1,x2,y2)		Der Abstand zwischen zwei Punkten (x1,y1) und (x2|y2)

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:

_posY = posY + vY;_____ oder in Kurzform ____posY += vY;_____

und analog für die x-Koordinate _posX += vX;_____ .

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

Zuweisung

Ball	b1	=	new	Ball();
Datentyp	Objektname	„	Neues Objekt“	Konstruktor des Datentyps

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

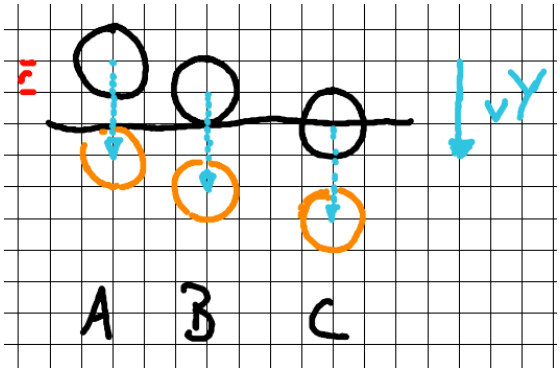
Es liegt eigentlich eine Klasse „Ballphysik“ vor. Die Zeile „class Ballphysik {}“ wird nur nicht angezeigt.

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

Das Objekt b1 wird bei jedem Durchgang neu angelegt anstatt das bereits bestehende Objekt neu zu definieren(). Eine spätere Animation wäre dann nicht möglich.

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:

```
if (posY + r +vY >= height) {}
```

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.

```
if (posY + r +vY >= height) {
        vY *= -1;
}
```

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	true	true
false	true	false

UND	true	false
true	true	false
false	false	false

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

```
if (posX + r + vY >= width || posX - r + vY <= 0) {
        vX *= -1;
}
```

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.

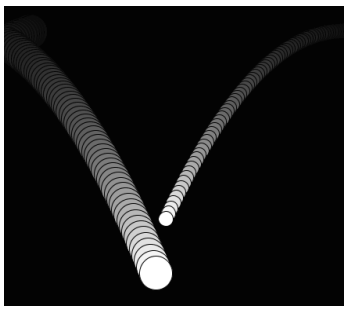
Aufgabe 2.2.2: Energieverlust

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:

vX = vX * 0.9;

vY *= 0.9;

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.

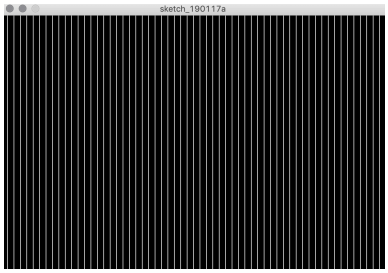
Aufgabe 3.1.1: Fingerübungen für Zählwiederholungen

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

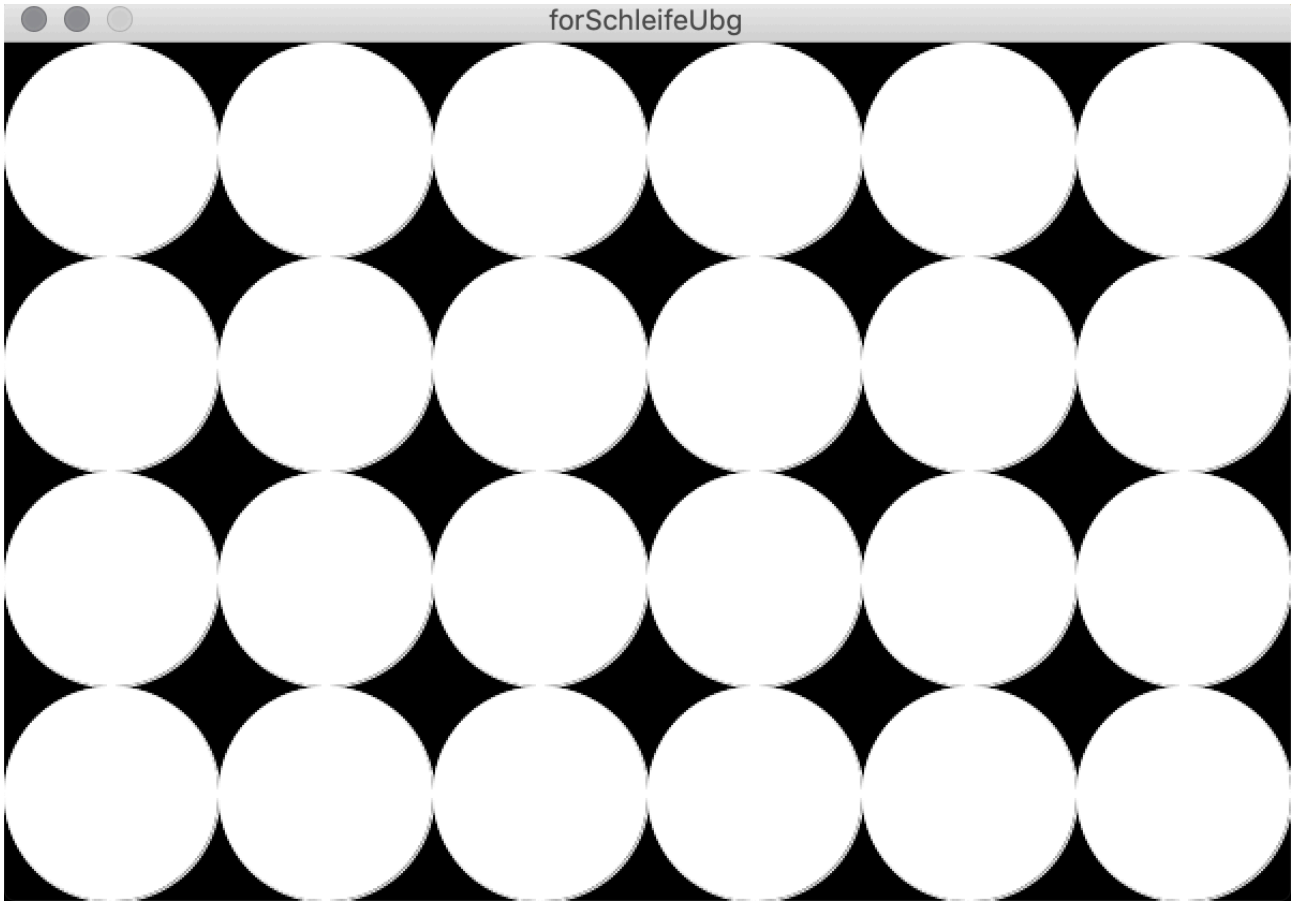
```
void setup() {
  size(600,400);
  background(0);
  stroke(255);
}

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: Statische Bildmanipulation

Da die Pixel eines Bildes in einem Array verwaltet werden, können einfach Filter, wie sie bei vielen Apps eingebaut sind, selbst programmiert werden. Die zu manipulierenden Bilder müssen im Format jpg, gif, png und tga vorliegen.

Für processing ist der angezeigte Fensterinhalt nur ein Pixel-Bild, das die ganze Zeit verändert wird. Möchte man die angezeigten Pixel im Fenster von Hand verändern, so muss zunächst ein vorgegebenes Array „pixels" initialisiert werden. Die Länge dieses Arrays ist Breite x Höhe des Bildes. Bevor mit einer Bildmanipulation angefangen werden kann, muss dieses Array durch Aufruf der Methode loadPixels() initialisiert werden. Die gemachten Veränderungen wirken sich erst aus, wenn der Inhalt des Arrays „pixels" durch Aufruf der Methode „updatePixels()" wieder in das Fenster geschrieben wird.

a.) Tippe zunächst folgenden Code ab:

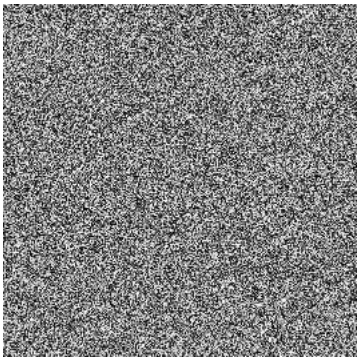
```
void setup() {
  size(600, 400);  //Größe des Bildes bzw. Fensters
  loadPixels();    //Alle Pixel des Bildes in das Feld pixels laden
  //----- Bildmanipulation ----------

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

void draw() {}

//Speichern des neuen Bildes auf der Festplatte im gleichen Projektordner
void mouseReleased() {
  //Erzeugen eines eindeutigen Namen
  //Zusammensetzen eines eindeutigen Dateinamens
  String fileName = "myPic"+day()+minute()+second()+".jpg";

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

Die Methode draw() ist im Moment nur nötig, damit die Methode mouseReleased() überhaupt ausgeführt wird.

Ergänze nun eine for-Schleife, die über die Länge des pixels-Arrays iteriert und jedes Pixel mit einem zufälligen Grauwert belegt:

```
pixels[idx] = color(random(255));
```

Kontrolliere, ob durch Klick auf das erzeugte Bild eine Datei in deinem Projektordner erzeugt wird.



In den folgenden Teilaufgaben wird ein Bild auf die unterschiedlichsten Weisen manipuliert. Suche, unter Berücksichtigung der entsprechenden Bildrechte, ein möglichst buntes Bild, achte auf deutliche Rot-, Grün-,Blau-Bereiche und speichere dies im gleichen Ordner in dem auch deine Projektdatei liegt.

b.) Passe den Quelltext entsprechend an.

```
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");

  //Bild speichern
  image(img,0,0);

  loadPixels();    //Alle Pixel des Bildes in das Feld pixels laden
  //----- Bildmanipulation ----------
  //--------------------------------
  updatePixels(); //Das Feld pixels wieder in das Bild zurückschreiben
}
void draw() {}

void mouseReleased() {
  String fileName = "myPic"+day()+minute()+second()+".jpg";
  save(fileName);
}
```

Um ein gespeichertes Bild in das interne Pixel-Array „pixels" zu übertragen, wird es zunächst mit Hilfe des Befehls „image(img,0,0);" im Fenster angezeigt. Im zweiten Schritt wird durch Aufruf von loadPixels() das angezeigte Bild im processing-internen Array „pixels" gespeichert.

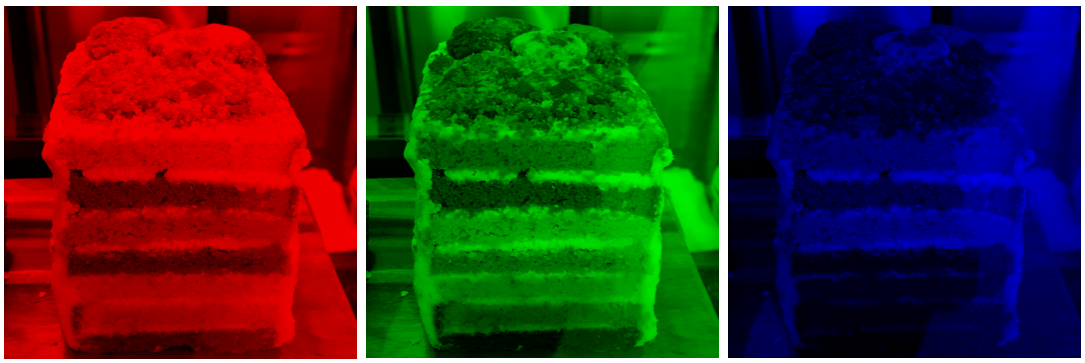
c.) Lege ein Blumen-Bilddatei in den Projektordner und passe die setup()-Methode so an, dass das Bild angezeigt wird.

d.) Mit der Funktion brightness(pixels[idx]) wird die Helligkeit eines Pixels mit Werten aus [0,255] abgerufen. Damit lassen sich Graustufen-Bilder erzeugen. factor definiert die Helligkeit des Graustufenbildes. Versuche die Werte 0.5, 1 und1.5 . Speichere drei Varianten.

```
pixels[idx] = color(factor*brightness(pixels[idx]));
```

e.) Ist die Helligkeit eines Pixels größer als der Schwellwert 100, so soll das entsprechende Pixel weiß, andernfalls schwarz werden. Es entsteht ein reines Schwarz-Weiß-Bild. Probiere Grenzwerte aus dem Intervall [0,255].

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



Benutze hierfür die Funktionen red(pixels[idx]), green(pixels[idx]) oder blue(pixels[idx]), um die einzelnen Farbinformationen abzufragen und entsprechende das Bild einzufärben:

```
pixels[idx] = color(red(pixels[idx]),0,0);
```

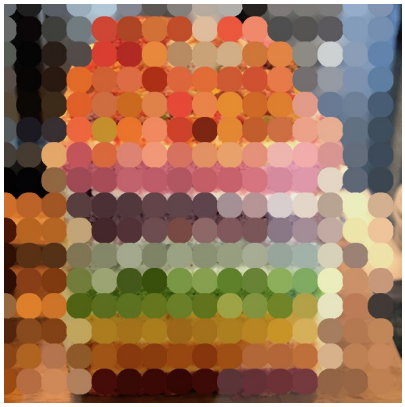
Speichere jeweils eine blaue, eine rote und eine grüne Version deines Bildes.

Hinweis: Da additive Farbmischung vorliegt, alle Farben zusammen ergeben die Farbe Weiß, kannst du auch leicht ein gelbes Bild mischen. Speichere dies.

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

```
//Farbinformationen des Pixels idx
float r = red(pixels[idx]); //etc.

float newR = 0.4*r + 0.75*g + 0.2*b;
float newG = 0.35*r + 0.7*g + 0.15*b;
float newB = 0.25*r + 0.5*g + 0.15*b;

//Für bessere Ergebnisse sollte der
//Wertebereich eingeschränkt werden: [0,255]
newR = constrain(newR,0,255);
newG = constrain(newR,0,255);
newB = constrain(newR,0,255);
```

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



Tipp: Hierfür muss das Bild nicht mit loadPixels() erst in das interne Array „pixels" geladen werden. Mit Hilfe der Funktion get(x,y), kann direkt die Farbe eines Pixels des Objekts img abgerufen werden:

```
color pixelColor = img.get(x,y);
```

Benutze zwei ineinander geschachtelte for-Schleifen, um das ganze Bild umzuwandeln. Versuche auch andere geometrische Figuren.

Aufgabe 3.1.3: Dynamische Bildmanipulation

Wie dir bei der Aufgabe 3.1.2 h) vielleicht aufgefallen ist, liegt im Hintergrund immer noch das alte Bild. Das ist weniger schön und soll im Folgenden anders sein.

Zudem ist es unhandlich, dass verschiedene Grenzwerte nicht per Maus manipuliert werden können. Bei der Aufgabe 3.1.2 e) wäre es eleganter gewesen, wenn der Schwellwert mit der Maus verändert werden könnte und die Veränderung sofort sichtbar gewesen wäre-

a.) Pass zunächst deinen Quellcode so an, dass die Bildmanipulation nun im Rumpf der Methode draw() steht.

```
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");
  loadPixels(); //Einmaliges Initialiseren des internen pixels-Array
}

void draw() {
  //----- Bildmanipulation ----------

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

void mouseReleased() {
  String fileName = "myPic"+day()+minute()+second()+".jpg";
  save(fileName);
}
```

Beachte, da sich das angezeigte Bild die ganze Zeit verändert, nicht mehr auf das angezeigte Bild und seine Pixel zugegriffen werden kann. Es muss immer auf das ursprüngliche „img"-Objekt und sein pixel-Array „pixels" zurückgegriffen werden.

b.) Übertrage deine Lösung aus Teilaufgabe 3.1.2 e) in die draw()-Methode.
Tipp: Falls dein Bild sich nur langsam ändert, so kann das an folgenden Punkten liegen:

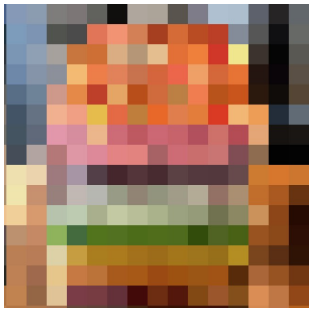
- *Dein Bild ist zu groß (Benutze maximal 1000px auf 600px)*
- *Du hast noch eine println-Ausgabe des pixel-Arrays im Quelltext. Die Ausgabe dauert sehr lange.*

b.) Um die Helligkeit leichter anpassen zu können, soll diese mit der Maus gesteuert werden können. Hierzu wird der x-Werte der Maus [0,width] auf das Intervall [-255,255] projiziert. Gilt mouseX == width/2, so bekommst du das Bild aus 3.1.2 d.). 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 curSummand = map(mouseX,0,width,-255,255);
pixels[idx] =  color(curBrightness+curSummand);
```

c.) Passe den Quelltext aus b.) so an, dass der Schwellwert, der bei der Aufgabe 3.1.2 e) fix auf 100 stand, nun mit dem Wert von mouseX korreliert. Versuche eigene, sinnvolle Intervalle für die Funktion map() zu finden.

d.) Anders als in 3.1.2 h) soll das Bild nun „verpixelt“ werden. Je weiter die Maus im rechten Bildbereich ist, desto stärker soll „verpixelt“ werden.



Aufgabe 3.1.3: Maskierung

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 3.1.4: „Verpixler"

Möchtest du Bilder veröffentlichen, so muss du eventuell Gesichter, Kennzeichen und anderes unscharf machen, damit nicht genau erkannt wird, wer oder was auf deinem Foto ist.

a.) Recherchiere im Internet, was alles aus Datenschutzgründen verpixelt werden muss.

Gesichter, Kennzeichen, Firmennamen wg Werbung

b.) Suche nach Apps, so dass du Fotos auf deinem Smartphone verpixeln kannst.

c.) Schreibe ein neues Projekt „Pixelator" bei dem du mit der Maus nur Bereiche deines Bildes verpixeln kannst.
Um das Speichern und das Verpixeln zu trennen, muss die mouseReleased()-Methode entfernt werden. Benutze folgendes Programmfragment und ergänze deinen Quellcode:

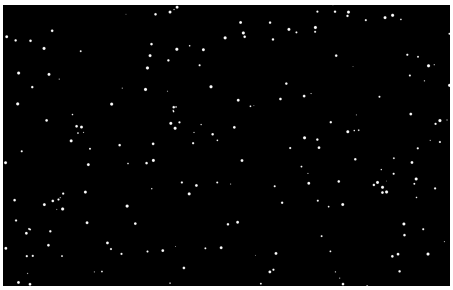
```
if (mousePressed && (mouseButton == LEFT)) {
    //linke Maustaste ist verpixeln
} else if (mousePressed && (mouseButton == RIGHT)) {
    //rechte Maustaste ist speichern
}
```

Aufgabe 3.1.5: Bildmanipulation (Abschlussaufgabe)

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.

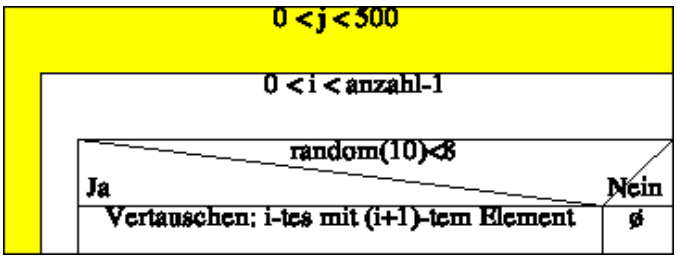
Aufgabe 3.2.2: Sortieren mit BubbleSort

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 verschiedenen Kommazahlen (float)gefüllt wird. Benutze hierfür eine Methode init().init() belegt das Feld mit den Werten von 1 bis einschließlich 100.
Erzeuge 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 **negativen** Einträgen im „zahlen"-Feld.
Beachte, dass die Rechtecke von „unten nach oben" gezeichnet werden:

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

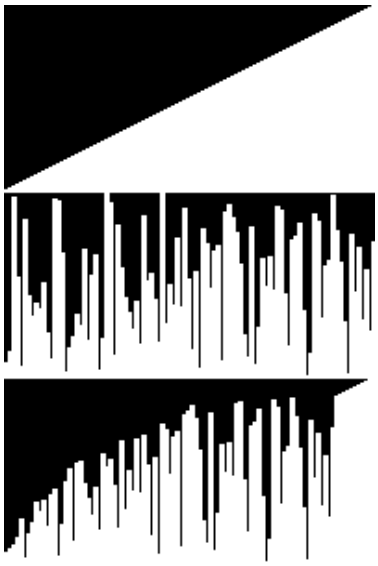
b.) Damit später etwas zu sortieren ist, muss das Feld zunächst gemischt werden. Hierfür lässt sich der innere Teil des BubbleSort-Algorithmus benutzten.
Notiere den Mischalgorithmus im unteren Bereich in ausformulierten Code als Methodenrumpf einer Methode „void mixIt()".



Java

```
void mixIt() {
  for(int i=0; i<500; i++){
    for(int idx=0; idx<zahlen.length-1; idx++){
      if(random(10)<8){
        int merken = zahlen[idx+1];
        zahlen[idx+1] = zahlen[idx];
        zahlen[idx] = merken;
      }
    }
  }
}
```

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 einzigen Sortiervorgangs über das ganze Feld.
- Zeichnen der Rechtecke.

Eventuell muss die frameRate auf 10 oder weniger 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

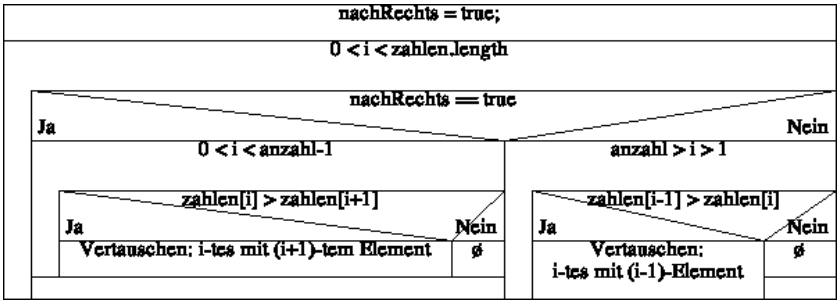
```
for(int i=0; i<zahlen.length; i++){
  for(int idx=0; idx< zahlen.length -1; idx++){
    if(zahlen[idx+1] < zahlen[idx]){
      int merken = zahlen[idx+1];
      zahlen[idx+1] = zahlen[idx];
      zahlen[idx] = merken;
    }
  }
}
```

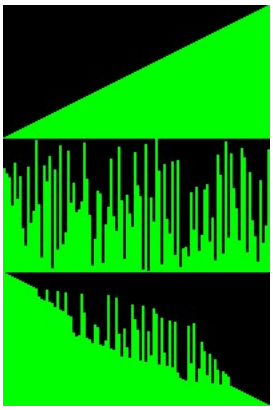
f.)* In der Regel werden drei Situationen für die Qualität eines Algorithmus benutzt: Die Laufzeit im idealen, im Durchschnitt und im schlechtesten Fall.
Speichere dein Projekt unter dem Namen „BubbleSort-Laufzeiten“ ab und verändere es so, dass im oberen Bereich der ideale Fall, im mittlere Bereich der Durchschnitt und im unteren Drittel der schlechtesten Fall sortiert wird. Benutze drei Felder zahlen1, zahlen2, zahlen3. zahlen1 ist bereits fertig sortiert, zahlen2 ist gemischt und zahlen3 ist genau verkehrt herum sortiert. Ist die Sortierung einer Variante fertig, so soll die entsprechende Anzeige von weiß auf grün wechseln.

Aufgabe 3.2.3: Andere Sortieralgorithmen[3]

Basierend auf der Idee des BubbleSort-Algorithmus lassen sich noch andere Sortieralgorithmen realisieren.

a.) Setze den Shake-Sort-Algorithmus aufbauend auf der Lösung von Aufgaben 3.2.3 um.





b.)* Anstatt einfach nur zu vertauschen, kann man auch immer das kleinste Element im verbliebenen Feld suchen und an die vorderste Position stellen. Setze diese Variante des Selection-Sort-Algorithmus um.

[3] Im Lehrplan des Jahres 2019 ist nur der BubbleSort-Algorithmus vorgegeben.

Aufgabe 3.2.4: Kleine Anwendungsaufgaben

Basierend auf Aufgabe 3.2.2 lassen sich noch ein paar weitere, immer wiederkehrende Aufgaben üben. Das Ergebnis aller folgenden Methoden sollen im unteren Drittel dargestellt werden. Das Feld muss hierzu natürlich nicht mehr sortiert werden. Wenn möglich soll der jeweilige Vorgang animiert dargestellt werden.

Benutze für alle Aufgaben ein gemischtes Feld, wie es beim Bubble-Sort-Algorithmus benutzt wurde.

a.) Da bisher die Zahlenwerte von Beginn an aus dem Intervall [1,100] kamen, würden die folgenden Aufgaben nur bedingt sinnvoll sein. Daher wird das Feld nun mit 100 zufälligen Zahlen aus dem Intervall [1,1000] gefüllt.
   - Definiere eine Variable „float maxZufallsWert = 1000;“, die für alle Methoden verfügbar ist.
   - Verändere die Initialisierung wie folgt:
     zahlen[idx] = random(1,maxZufallsWert);
   - Da die Rechtecke maximal 100px hoch sein können, wird nochmals die Funktion map() benutzt:
     rect(idx*2,100,2,-map(zahlen1[idx],1,maxZufallsWert,1,100));

b.) Programmiere eine Methode „void findMax()“, die das größte Element des Feldes „zahlen“ findet und in der Anzeige markiert.

c.) Programmiere eine Methode „int findMin()“, die das kleinste Element des Feldes „zahlen“ findet, markiert und den Index, also die Fundstelle, zurückgibt. Lass den Wert ausgeben.

d.) Summiere alle Elemente auf und gib das Ergebnis zurück: „int getSum()“

e.) Summiere alle geraden Elemente auf und gib das Ergebnis zurück: „int sumEven();“

Aufgabe 3.2.5: Anwendungsaufgaben mit eigenen Klassen

Aufbauend auf den vorangegangenen Aufgaben werden nun statt einfachen Datentypen eigene Datentypen (Klassen) benutzt.

Hinweis: Das Programm wird über die setup-Methode gestartet. draw() wird nicht benötigt.

a) Lege ein Projekt „Supermarktkasse"an. In einer eigenen Datei legst du eine Klasse „Produkt" an. Der Konstruktor der Klasse „Produkt" legt per Zufall fest, ob der Typ des Produkts ein „Food" oder „NonFood" ist, der Preis wird zufällig auf [0.99;5.99] festgelegt.

b) Definiere eine weitere Klasse „Kunde". Im Konstruktor des Kunden wird das Attribut „Produkt[] einkaufswagen;" zufällig mit 10 bis 50 Produkten belegt.

c) Definiere ein Methode „double gibGesamtpreis()", die den Gesamtpreis der Produkte im einkaufswagen zurückgibt.

d) Definiere eine Methode „double gibMaxPreis()", die den Betrag des teuersten Artikels zurückliefert.

e) Im Hauptprogramm des Projekts legst du eine Variable „Kunden[] schlange" an. Initialisiere das Feld „schlange" zufällig mit [3;10] Kunden.
Definieren eine Methode „wartezeit()", die die aktuelle Wartezeit an der Schlange ausgibt. Die Wartezeit setzt sich aus 1 Sekunde pro Produkt und 45 Sekunden für den Bezahlvorgang zusammen.

f) Ergänze eine Methode „anzahlNonFood()", die die Anzahl der gescannten NonFood-Artikel am Ende des Programmdurchlaufs ausgibt.

g*) Definiere im Hauptprogramm ein Array „plz" in der Form

```
int[] plz = {12345,12312,12342,12343,12243};
```

Die Klasse Kunde bekommt ein zusätzliches Attribut „postleitzahl", welches im Konstruktor mit einem zufälligen Element aus dem Array „plz" initialisiert wird. Definiere eine Methode „summeProPlz()" im Hauptprogramm, die die Gesamtsumme pro Kunde pro PLZ nach Abarbeitung der Kunden ausgeben kann. Eventuell werden weitere Variablen benötigt.

Aufgabe 3.2.6: Spielereien

Eine Ausgabe kann bei den folgenden Teilaufgaben entweder in die Konsole erfolgen: „println()“. Schöner wäre aber natürlich eine grafische Darstellung bzw. Ausgabe. Hierzu gebe ich Teile des Quelltextes für eine Klasse Button an, die jeder nach seinem eigenen Geschmack abwandeln darf.

```
class Button {
  float x,y,w,h;
  String text;
  color btnColor;
  color textColor;
  int fontSize;

  Button(float newX, float newY, String btnLabel) {
    active = true;
    fontSize = 12;
    x = newX;
    y = newY;
    btnColor = #444444;
    textColor = #fefefe;
    text = btnLabel;
    w = text.length()*fontSize;
    h = fontSize*2;
  }

  void show() {
    fill(btnColor);
    stroke(brightness(btnColor)*2);
    rect(x,y,w,h,5);
    push();
    translate(w/2,fontSize*1.5);
    textAlign(CENTER);
    textSize(fontSize);
    fill(textColor);
    text(text,x,y);
    pop();
  }


  boolean mouseOver() {
    return mouseX > x && mouseX < x+w &&
           mouseY > y && mouseY < y+h;
  }
}
```

a) Lege ein neues Projekt „Tippgeber“ an. Definiere ein Array „tipps“, das verschiedene Tätigkeiten vorschlägt: Lernen, Schlafen, Sport, Kino,.... Wähle zufällig einen Eintrag aus und zeige ihn auf dem Bildschirm an.

b) Lege ein neues Projekt „Tiernamen“ an.
Definiere ein Feld „namen“ in welches verschiedene Namen von Tieren enthalten sind: Ente, Elefant, Maus,...
Wähle zufällig zwei Elemente aus und verknüpfe sie zu einem neuen Tier: EnteMaus

c) Passe das Programm aus b) so an, dass es sprachlich korrekter ist: Entenmaus, Elefantenmaus, Mauselefant. Definiere hierfür zwei Felder „vorderesWort“ und „hinteresWort“.

d) Lege ein neues Projekt „Phrasendrescher“ an. Definiere für einen einfachen deutschen Satz aus den Bestandteilen „Subjekt“ + „Prädikat“ + „Objekt“ eine Phrasendrescher in dem du Bestandteile des Satzes zufällig erzeugst.

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.
   Für jede „{" wird in der nächsten Zeile um ein Tabulatorsprung eingerückt. Für jede „}" wird eine Einrückungsebene zurückgenommen.

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.

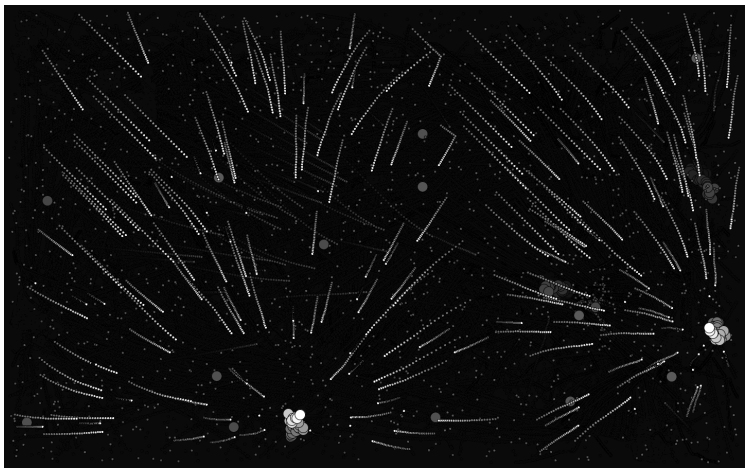
f.) Wenn das Futter kein Futter ist, sondern Gift, dann sterben die Ameisen, wenn Sie das Gift berühren. Passe dein Programm entsprechend an.

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 1 Giftköder sollen nun mindestens 2 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.

k.) Alle sich bewegenden Objekte haben gemeinsame Attribute bzw. Methodennamen. Definiere eine abstrakte Basisklasse „"auf der alle anderen Objekte basieren.

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.[4]

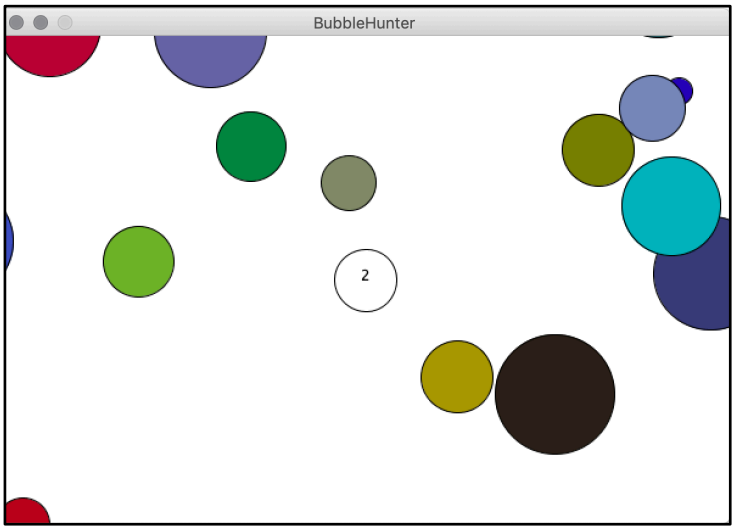
a.) Erstelle ein neues Projekt „BubbleHunter“. Hintergrundfarbe des bildschirmfüllenden Spiels soll diesmal Weiß sein.
Erstelle eine Klasse „Bubble“ analog zu den Ameisen mit den bekannten Attributen x,y,d sowie colorR, colorG,colorB, da die Bubbles bunt werden sollen. Ergänze auch die benötgte Methode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



[4] 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:

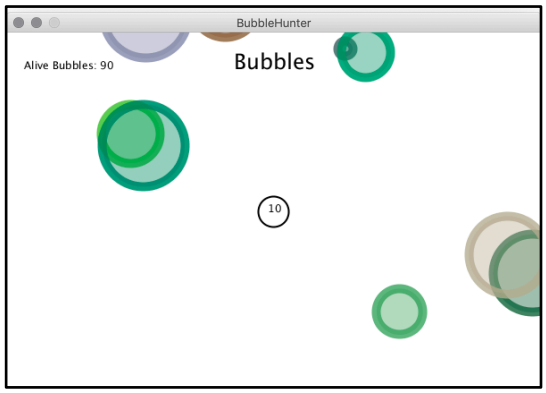
```
translate(width/2-player.x, height/2-player.y);
```

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: Spiele

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[5])	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.

Anhand des Spiels „Bubble“ werden Zustands- und Sequenzdiagramme erläutert. Nach einer anschließenden Übungsphase schließt sich das erste eigene Spiel an:

Andere Spiele werden sein: Reaktionsspiel, NimSpiel, 17+4

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

Die Hauptdatei

```
//Abwandlungen
//Bubbles verschwinden nicht, sondern werden neu positioniert
//Beim Anklicken wird nur die Farbe gewechselt
//Je länger das Spiel, desto schneller werden die Bubbles
//Die Bubbles sind am Anfang groß und werden immer kleiner

Bubble[] bubbles;
int nbrOfBubbles = 5;    //Anzahl der anzuzeigende Blasen
int curBubbles;          //Aktuelle Anzahl an zu sehenden Blasen
int curPoints;           //Aktuelle Punktezahl
int gameState;           //0:Anleitung 1:Spiel 2:Spielstand und Nochmal?
int endPoints;           //Erreichte Punkte
Button startBtn, againBtn, closeBtn;

void setup() {
  size(600,400);
  frameRate(30);
  startBtn = new Button(width*0.4,height*0.6, "Start!");
  againBtn = new Button(width*0.35,height*0.6, "AGAIN?");
  closeBtn = new Button(width*0.55,height*0.6, "Ende");

  //Startzustand des Spiels ist der Zustand 0
  gameState = 0;
}
```

Anzeige

```
void displayGameInfos() {
  textSize(12);
  textAlign(LEFT);
  text("Bubbles left: "+curBubbles, 10,20);
  textAlign(RIGHT);
  text("Points: "+curPoints, width-10,20);
}

void displayTitel() {
  fill(255,255,255);
  textSize(32);
  textAlign(CENTER);
  text("BubbleShots", width/2,50);
}
```

Anzeige

```
void displayGameInfos() {
  textSize(12);
  textAlign(LEFT);
  text("Bubbles left: "+curBubbles, 10,20);
  textAlign(RIGHT);
  text("Points: "+curPoints, width-10,20);
}

void displayTitel() {
  fill(255,255,255);
  textSize(32);
  textAlign(CENTER);
  text("BubbleShots", width/2,50);
}
```

StartScreen

```
void displayStartScreen() {
  background(0);
  displayTitel();
  textSize(18);
  text("Lass möglichsst schnell alle Ballons zerplatzen", width/2,200);
  startBtn.show();
}
```

PlayingScreen

```
void playingScreen() {
  background(0);

  displayTitel();
  displayGameInfos();

  curPoints--;
  for(int idx=0; idx<bubbles.length;idx++) {
    bubbles[idx].update();
    bubbles[idx].show();
  }
}
```

Button

```
class Button {
  float x,y,w,h;
  String text;
  color btnColor;
  color textColor;
  int fontSize;

  Button(float newX, float newY, String btnLabel) {
    active = true;
    fontSize = 12;
    x = newX;
    y = newY;
    btnColor = #444444;
    textColor = #fefefe;
    text = btnLabel;
    w = text.length()*fontSize;
    h = fontSize*2;
  }

  void show() {
    fill(btnColor);
    stroke(brightness(btnColor)*2);
    rect(x,y,w,h,5);
    push();
    translate(w/2,fontSize*1.5);
    textAlign(CENTER);
    textSize(fontSize);
    fill(textColor);
    text(text,x,y);
    pop();
  }


  boolean isClicked() {
    return mouseX > x && mouseX < x+w && mouseY > y && mouseY < y+h;
  }
}
```

EndScreen

```
void displayEndScreen() {
  background(0);
  displayTitel();
  textSize(18);
  text("Endpunkte: "+endPoints, width/2,200);
  againBtn.show();
  closeBtn.show();
}
```

GameLogic

```
void draw() {
  switch(gameState) {
     case 0:
       displayStartScreen();
       break;
     case 1:
       playingScreen();
       break;
     case 2:
       displayEndScreen();
       break;
  }
}

void mouseReleased() {
   switch(gameState) {
     case 0:
       if (startBtn.isClicked()) {
          gameState = 1;
          initGame();
       }
       break;
     case 1:
       for(int idx=0; idx<bubbles.length;idx++) {
           if (bubbles[idx].inside(mouseX,mouseY)) {
              bubbles[idx].hide();
              curBubbles--;
              curPoints+=500;
              //Wenn eine Blase geplatzt ist, kann die Schleife abgebrochen
werden
              idx=bubbles.length;
           }
       }
       //Spiel ist beendet, wenn
       if (curBubbles==0) {
         endPoints = curPoints;
         gameState = 2;
       }
       break;
     case 2:
       if (againBtn.isClicked()) {
          //Spiel neustarten
          gameState = 0;
       }
       if (closeBtn.isClicked()) {
          //Spiel neu beenden
          exit();
       }
       break;
  }
}
```

```
void initGame() {
  //Diese Methode wird bei jedem (Neu)-Start des Spiels benötigt
  //Blassen initialisieren
  bubbles = new Bubble[nbrOfBubbles];
  for(int idx=0; idx<bubbles.length;idx++) {
    bubbles[idx] = new Bubble();
  }
  //SpielAttribute anpassen
  curBubbles = nbrOfBubbles;
  curPoints = nbrOfBubbles*100;
}
```

Bubble

```
class Bubble {
  float x,y;
  float vX,vY;
  float r,g,b;
  float d;
  boolean active;

  Bubble() {
    d = random(20,100);
    x = random(2*d,width-2*d);
    y = random(2*d,height-2*d);
    vX = random(-5,5);
    vY = random(-5,5);
    r = random(0,255);
    g = random(0,255);
    b = random(0,255);
    active = true;
  }

  boolean inside(float mX, float mY) {
    return dist(x,y,mX,mY)<=d/2;
  }

  void hide() {
    x = -2*width;
    vX=0;
    vY=0;
  }

  void update() {
    if (x+d/2+vX>width || x-d/2+vX<0) {
      vX*=-1;
    }
    if (y+d/2+vY>height || y-d/2+vY-60<0) {
      vY*=-1;
    }

    x+=vX;
    y+=vY;
  }

  void show() {
    fill(r,g,b);
    ellipse(x,y,d,d);
  }

}
```

KAPITEL 4.1: Zustands-Diagramme

Aufgabe 4.1.1: Definition

Skizziere ordentlich die Grundelemente eines Zustands-Diagramms mit Hilfe zweier Zustände z1 und z2.

Aufgabe 4.1.2: Smartphone

Skizziere ordentlich das Zustandsdiagramm für das Entsperren deines Smartphones beginnend im vollständig ausgeschalteten Zustand bis zum entsperrten Zustand.

Aufgabe 4.1.3: Getränkeautomat

Skizziere ordentlich das Zustandsdiagramm für die Benutzung eines Getränkeautomaten in deiner Schule.

a) Vereinfache das Modell insofern als das der Automat nur 50ct und 1Euro-Geldstücke akzeptiert.

b) Skizziere den Zustandsübergang bei Bezahlung mit dem Handy.

Aufgabe 4.1.4: Software

Skizziere das Zustandsdiagramm für das Spiel Bubble.

KAPITEL 4.2: Sequenz-Diagramme

Nachdem nun die grundlegenden Programmiers