Musterlösungen für Blatt 5

Derzeit haben nicht alle Bonusaufgaben eine Musterlösung.

Inhaltsverzeichnis der Musterlösungen

501

Magische Miesmuschel

1.1 Globale Variablen

Dies sind im ganzen Programm immer wieder benötigte Informationen: Der Status der Maus, Anzeigezeit für Anworten, die Antworten selber usw.

```
1  PImage muschel; // Bild-Variable
2  boolean mouse = false; // ist die Maus geklickt worden?
3  int ansNum; // zufäälliger Wert für Antwort
4  int onTime = 0; // Timer für Antwort


5  // Antwortmöglichkeiten werden in den Constructor geschickt und in
6  // einem Array platziert
7  Antwort[] answ = {new Antwort("Heute nicht."),
8     new Antwort("Ich glaube nicht."),
9     new Antwort("Mhmm nein."),
10    new Antwort("Frag mich nochmal."),
11    new Antwort("Kannst du das wiederholen?"),
12    new Antwort("Nein."),
13    new Antwort("Können Schweine fliegen?"),
14    new Antwort("Ja."), new Antwort("Auf jeden Fall."),
15    new Antwort("Du darfst heute alles.")};
```

1.2 Klasse: Antwort

Hier deklarieren wir die Klasse Äntwort". Eine Klasse ist eine Zusammenstellung von Variablen und Funktionen, die auf sie anwendbar sind. In diesem Fall wurde sich entschieden, die Antwortstrings in dieser Klasse zu Kapseln, und die Funktion `void answ()` zu erstellen, welche Antwortstrings anzeigt.

```
1  // Klasse Antwort
2  class Antwort {

3     String answ;

4     // Constructor
5     Antwort(String a) { // Constructor übergibt die Variablen von einzelnen
          Objekten z.B. anws1 zur Erstellung an die Klasse
6        this.answ = a;
7     }

8     // Methode zum Darstellen des Antwort-Textes
9     void answ() {
10       textSize(width/20); // legt Schriftgröße fest
11       textAlign(CENTER); // richtet Text an der Mitte der "gewählten Text-
             Koordinaten" aus
12       text(answ, width/2, height/10); // gibt Text aus
13    }
14 }
```

1.3 mousePressed

Diese Funktion wird automatisch aufgerufen, sobald ein Mausklick erfolgt. Wir vermerken dies, indem wir den Wert der `boolean`-Variable `mouse` ändern. (Damit dies funktioniert, wird `mouse` nach jedem `draw`-Durchgang wieder negativ gesetzt.)

```
1  // Maus-Trigger für Antwort
2  void mousePressed() {
3     if (mouse == false) {
4         //Neue zufällige Zahl für neue zufällige Antwort
5        ansNum = int(random(7));
6        mouse = true;
7     }
8  }
```

1.4 setup

Hier finden sich unveränderliche Einstellungen: der Hintergrund, die Framerate, und natürlich die Größe des Fensters.

```
1  void setup() {
2     size(1000, 600);

3     // Bild der Miesmuschel wird in Variable deklariert
4     //Damit dieser Zugriff möglich ist, gehen Sie in processing auf
5     // Sketch -> Datei hinzufügen und öffnen sie die gewünschte
6     // Datei.
7     muschel = loadImage("muschel.jpg");
8     /
9     tint(240); // setzt Basis-Bildeffekt
10    image(muschel, 0, 0, width, height); // Bild wird dargestellt

11    // Framerate wird von 15xperSek heruntergesetzt damit die versch.
12    // Antworten nacheinander wahrscheinlicher sind
13    frameRate(4);
14 }
```

1.5 draw

Herzstück des Programmes. Schaut bei jedem Durchlauf, ob `void mousePressed()` einen neuen Mausklick registriert hat und lässt dann je nach aktuellem `ansNum`-wert eine Antwort anzeigen.

```
1  void draw() {

2     // Timer für Antwort
3     if (onTime >= 16) {
4        tint(240); // setzt Bildeffekt zurück
5        image(muschel, 0, 0, width, height);
6     }


7     // Aktualisierung neue Frage
8     if (mouse == true) {
9        onTime = 0;
10       strokeWeight(5);
11       noTint(); // setzt Bildeffekt
12       image(muschel, 0, 0, width, height); // löscht alte Antwort
```

Auf der nächsten Seite geht's weiter...

```
13          // Sprach-Visualisierung
14          for (int i = 0; i <= 3; i++) {
15             pushMatrix();
16             translate(width/2, height/2);
17             rotate(radians(i*10));
18             line(30, -30, 70, -70);
19             popMatrix();
20          }
21       }

22       // Abstand zum geheimen Punkt?
23       float d = dist(mouseX, mouseY, 658, 154);

24       fill(0); // Textfarbe: Schwarz

25       // "geheime" Postition für positive Antwort
26       if (mouse == true && d <= 15.0) {
27          if (ansNum <= 2) {

28          // Methodenaufrufe
29          answ[7].answ();

30          } else if (ansNum <= 5) {
31             answ[8].answ();
32          } else {
33             answ[9].answ();
34          }
35       } else if (mouse == true) { //Beliebige Antwort
36          answ[ansNum].ans();
37       }
38       mouse = false; // Mausklick wird zurückgesetzt (sonst wuerde der
             naechste Klick nicht registriert.)
39       onTime++; // Timer für Antwortwiedergabe zählt hoch

40       // Finde die Koordinaten der "geheim"-Postion auf dem Bild
41       // println(mouseX, mouseY);
42       // 658 154 "Ja"-Trigger-Punkt
43    }
```

Dino-Spiel

2.1 Vorbereitung und Logik

Zunächst legen wir eine Klasse `Obstacle`] für die Hindernisse an, da wir einige davon brauchen werden und wir so alle Objekte in einer Liste speichern können.

```
1   class Obstacle{

2       int obstacleSpeed = 8;
3       //Parameter der Hindernisses
4       int x,y,w,h;
5       PImage image;

6       public Obstacle(int speed){
7           x=width;
8           y=420;
9           w=50;
10          h=80;
11          obstacleSpeed = speed;
12          image = loadImage("cac.png");
13      }
14  }
```

Dann fügen wir eine Methode `draw()` hinzu, die das Hindernis zeichnet und eine Methode `move()` welche das Hinderniss bewegt und nach der Kollision mit dem Dino schaut.

```
1   void draw(){
2       fill(0);
3       image(image,x,y,w,h);

4   }

5  void move(){
6    //Wird nach links bewegt
7    this.x -=obstacleSpeed;
8    //Testen, ob das Dino Rechteck mit dem von derm Hinderniss überlapt und
         ruft dann "gameover()" auf
9    if (x < dino_x+dino_x_offset + dino_w+dino_w_offset &&
10     x + w > dino_x+dino_x_offset &&
11     y < dino_y+dino_y_offset + dino_h+dino_h_offset &&
12     y + h > dino_y+dino_y_offset) {
13     gameOver();
14   }

15 }
```

2.2 Das Spiel

Zunächst legen wir eine Liste an, in der wir alle Hindernisse speichern können.

```
1   ArrayList<Obstacle> obstacles = new ArrayList<Obstacle>();
```

Dann brauchen wir noch weitere Informationen über den Dino. Wir speichern die Position, die Größe und die Geschwindigkeit in Variablen. Ein interessantes Konzept um den Dino zu animieren ist die Verwendung von Bildern. Wir speichern also die Bilder in einer Liste und wechseln dann zwischen den Bildern, um den Dino zu animieren.

```
1   //Dino Koordinaten
2 int dino_y = 400;
```

```
3  int dino_x = 150;
4  int dino_h = 100;
5  int dino_w = 90;

6  //Verkleinerung der Dino Kollisionsboxen im Vergleich zur Texturgröße,
       damit die äußeren "leeren" Ecken der Textur nicht mit den Kakteen
       kollidieren
7  int dino_x_offset = 30;
8  int dino_y_offset = 10;
9  int dino_h_offset = -40;
10 int dino_w_offset = -60;

11 //Geschwindigkeit & Beschleunigung des Dino auf der y Achse
12 double y_Velocity = 0;
13 double y_Acceleration = 0;

14 //Die Verschiedenen Bilder der Animation des Dinos
15 PImage spritesheet;
16 PImage[] dinos = new PImage[5];
17 int dinoAnimIndex = 0;

18 //"Erdbeschleunigung"
19 final double groundAcc = 0.45d;

20 //Wie stark der Dino Springen soll
21 final double jumpPower = -10d;


22 int score = 0;
23 int highscore = 0;

24 // Eine List mit Hindernissen, welche auftauchen
25 ArrayList<Obstacle> obstacles = new ArrayList<Obstacle>();


26 void setup(){
27    size(1280,720);
28    surface.setTitle("Dino Spiel");
29    onStart();

30    spritesheet= loadImage("dino.png");
31    dinos[0] = spritesheet.get(1,0,44,47);
32    dinos[1] = spritesheet.get(45,0,44,47);
33    dinos[2] = spritesheet.get(89,0,44,47);
34    dinos[3] = spritesheet.get(133,0,44,47);
35    dinos[4] = spritesheet.get(177,0,44,47);
36 }

37 //Hier werden die ganzen Werte auf Standarteinstastellungen zurück gesetzt
38 void onStart(){
39   score = 0;
40   y_Velocity = 0;
41   y_Acceleration = 0;
42   dino_y = 400;
43   dino_x = 150;
44   score = 0;
45   obstacles.clear();
46 }

47 void draw(){
48    //Alle 90 Frames wird ein neues Hinderniss hinzugefügt
49    if(frameCount % 90 == 0){
50      obstacles.add(new Obstacle(8 + (score/500)));
51    }
52    //Score um 1 erhöht
```

```
53    score++;
54    //Berechnung der Y- Position des Dions
55    //Beschleunigung wird um erdbeschleunigung erhöht, geschwindigkeit um
          beschleunigung, y um geschwindigkeit
56    y_Acceleration +=groundAcc;
57    y_Velocity+= y_Acceleration;
58    dino_y+=y_Velocity;

59    //Wenn der Dino landet
60    if(dino_y>=400){
61      y_Acceleration = 0;
62      dino_y = 400;
63    }
64    //Geschwindigkeit wird jeden frame zurückgesetzt
65    y_Velocity=0;



66    //Zeichnen
67    background(240);

68    //Hier werden die Hindernisse bewegt und gezeichnet
69    for(int i=0;i<obstacles.size();i++){
70      Obstacle ob = obstacles.get(i);
71      ob.move();
72      ob.draw();
73    }

74    fill(0);
75    //Boden
76    line(100,500,1180,500);
77    //"Dino"

78    image(dinos[dinoAnimIndex],dino_x,dino_y,dino_w,dino_h);
79    if(frameCount % 10 == 0){
80      dinoAnimIndex ++;
81      if(dinoAnimIndex >= dinos.length){
82        dinoAnimIndex = 0;
83      }
84    }

85    //Kann genutzt werden um die Kollisionsbox des Dinos anzuzeigen
86    //color(255,0,0);
87    //rect(dino_x+dino_x_offset,dino_y+dino_y_offset,dino_w+dino_w_offset,
          dino_h+dino_h_offset);
88    stroke(1);

89    //Scores
90    textSize(32);
91    text("No internet",100,600);
92    text("Score: "+score+" Highscore: "+highscore,500,100);
93  }
94  //wird aufgerufen, wenn der dino "kollidiert"
95  void gameOver(){
96    if(score>highscore)
97      highscore = score;
98    onStart();
99  }
100 //Springen bei "W" oder "Leertaste"
101 void keyPressed(){
102   if((key == 'w' || key==' ') && dino_y>=400){
103       y_Acceleration= -10d;
104     }
105 }
```

511

Space Invaders

3.1 Player

Legt grundlegende Funktionen der Spielfigur fest.

```
1  class Player {
2     int xPosition;
3     int speed;
4     int direction;
5     public Player() {
6        xPosition = 250;
7        speed = 5;
8        direction = 0;
9     }
10    void show() {
11       fill(255);
12       stroke(0);
13       rect(xPosition, 450, 12, 40);
14    }
15    void move() {
16       if (xPosition <= 480 && direction == 1) {
17          xPosition = xPosition + direction *speed;
18       } else if (xPosition >= 10 && direction == -1) {
19          xPosition = xPosition + direction *speed;
20       }
21    }
22    void setDirection(int dir) {
23       direction = dir;
24    }
25    int getXPosition() {
26       return xPosition;
27    }
28 }
```

3.2 Enemy

Ähnelt der Klasse Player, verfügt allerdings über eine änderbare yPosition und braucht keine verstellbare Geschwindigkeit.

```
1  class Enemy {
2     int xPosition, yPosition;
3     int dir;
4     Enemy(int x, int y) {
5        xPosition = x;
6        yPosition = y;
7        dir = 1;
8     }
```

Auf der nächsten Seite geht's weiter...

```
9     void show() {
10       fill(180, 30, 30);
11       stroke(0);
12       circle(xPosition, yPosition, 25);
13    }

14    void move(){
15       xPosition += dir;
16    }

17    void changeDirection(){
18       dir *= -1;
19    }

20    void down(){
21       yPosition += 20;
22    }

23    boolean bottom(){
24       return (yPosition > 420);
25    }

26    boolean edge(){
27       return (xPosition > 450 || xPosition < 50);
28    }

29    int getX() {
30       return xPosition;
31    }

32    int getY() {
33       return yPosition;
34    }
35 }
```

3.3 Bullet

Dies sind die Geschosse, mit denen die Spielfigur die Aliens angreifen kann.

```
1  class Bullet {

2     int xPosition, yPosition;
3     int speed;
4     boolean isShooting;

5     Bullet() {
6        yPosition = 460;
7        speed = 3;
8        isShooting = false;
9     }

10    void show() {
11       fill(250);
12       stroke(0);
13       circle(xPosition + 6, yPosition, 10);
14    }
```

Auf der nächsten Seite geht's weiter...

```
15      void move() {
16          yPosition -= speed;
17          if (yPosition < -10) {
18              reset();
19          }
20      }

21      void reset(){
22          isShooting = false;
23          yPosition = 460;
24      }

25      void shoot(int x) {
26          xPosition = x;
27          isShooting = true;
28      }

29      boolean isShooting() {
30          return isShooting;
31      }

32      int getX() {
33          return xPosition;
34      }

35      int getY() {
36          return yPosition;
37      }
38  }
```

3.4 Spielverhalten, weitere Logik

Aktuell tun unsere Klassen noch nicht viel. Erst durch weiteren Code werden ihre Bewegungen tatsälich ausgelöst.

Hier wird durch die Anweisungen in der `draw()`-Funktion und Events wie `keyPressed` aus den bisherigen Elementen ein Spiel.

3.4.1 Setup

Initalisierung globaler Variablen und Setzen des Spielfeldes.

```
1  //Hier stehen die Variablen die wir global anlegen.
2  //Dazu haben wir eine Variable vom Typ "Player", "Bullet" und eine Liste
       vom Typ "Enemy".
3  //Die Typen sind selbstangelegte Klassen.

4  Player player;
5  Bullet bullet;
6  ArrayList<Enemy> enemies;

7  //Diese Methode wird beim Start des Programms einmal ausgeführt.
8  //Das eignet sich für initalisierung von Variablen sehr gut.
9  void setup() {
10     //Größe des Programms in (x, y) Pixel
11     size(500, 500);
```

Auf der nächsten Seite geht's weiter...

```
12      //Initialisierung der Variablen
13      player = new Player();
14      bullet = new Bullet();
15      enemies = new ArrayList<Enemy>();

16      //Schleife um neue Objekte vom Typ "Enemy" in die Liste "enemies" einzuf
            ügen.
17      //Pro Schleifendurchlauf werden 2 Objekte eingefügt und die Koordinaten
            des "Enemy"
18      //werden als Parameter übergeben (Die Zahlen in den Klammern).
19      for (int i=0; i<6; i++) {
20          enemies.add(new Enemy(75 + 50*i, 100));
21          enemies.add(new Enemy(75 + 50*i, 150));
22      }
23  }
```

3.4.2 Draw

Bewegungen und Spiellogik

```
24  //Diese Methode wird regelmäßig aufgerufen, eignet sich um Informationen
25  //dauerhaft abzurufen und alles auf die graphische Oberfläche abzubilden
26  void draw() { //Farbe des Hintergrunds
27      background(50);

28      //Methoden der Player-Klasse um den Spieler anzuzeigen und ggf. zu
            bewegen.
29      player.show();
30      player.move();

31      //Schleife durch die Liste der "Enemies" um alle Objekte anzuzeigen und
            zu bewegen.
32      // Die .size() Methode einer Liste gibt die Anzahl der enthaltenen
            Elemente zurück
33      // Die Liste.get(x) Methode gibt das Objekt der Liste an der stelle x
            zurück.
34      for (int i=0 ; i<enemies.size() ; i++) {
35          enemies.get(i).show();
36          enemies.get(i).move();

37          //Die If-Abfrage wird benötigt um zu überprüfen ob eines der "Enemy"-
                Objekte sich am Rand
38          //des Programms befindet:
39          // Sollte das der Fall sein wird nochmal eine Schleife durch alle
                Elemente der Liste durchlaufen,
40          // sodass jeder "Enemy" die Richtung wechselt und sich nach unten
                bewegt.
41              if (enemies.get(i).edge()) {
42                  for (int k = 0 ; k<enemies.size() ; k++) {
43                      enemies.get(k).changeDirection();
44                      enemies.get(k).down();
45                  }
46              }
```

Auf der nächsten Seite geht's weiter...

```
48          //Diese If-Abfrage überprüft bei jedem Schleifendurchlauf für
                jedes Element, ob es sich am unterem Rand
49          //des Programms befindet: Sollte das passieren hat man Verloren
                und das Programm wird beendet.
50          if (enemies.get(i).bottom()) {
51             exit();
52          }
53       }

54       //Diese If-Abfrage überprüft, ob das "Bullet"-Objekt geschossen wurde
             :
55       //Wenn ja, dann wird das Objekt angezeigt und auch bewegt.
56          if (bullet.isShooting()) {
57             bullet.show();
58             bullet.move();

59             //Die Schleife wird genutzt, um Kollisionen zu erkennen.
60             //Sollte die Distanz der "Bullet" kleiner 20 sein zu einem der
                    "Enemies", dann
61             //wird das "Enemy"-Objekt aus der Liste entfernt und die
                    Methode .isShoot(false) aus
62             //der Bullet-Klasse aufgrufen um das Objekt zu "deaktivieren"
63             //Kreis-kreis abstand berechnung
64             for (int i=0; i< enemies.size(); i++) {
65                if (dist(bullet.getX(), bullet.getY(), enemies.get(i).getX()
                       , enemies.get(i).getY()) < 20) {
66                   enemies.remove(i);
67                   bullet.reset();
68                }
69             }
70          }
71       }
```

3.4.3 Events

Spielerinteraktion

```
72    //Diese Methode wird immer aufgerufen wenn eine Taste gedrückt wird
73    void keyPressed() {
74    //Für die Taste 'a' wird die Richtung des Spielers auf "-1" gesetzt.
75    if (key == 'a') {
76       player.setDirection(-1);
77    }
78    //Für die Taste 'd' wird die Richtung des Spielers auf "1" gesetzt.
79    if (key == 'd') {
80       player.setDirection(1);
81    }
82    //Für die Taste ' ' (Leertaste) wird überprüft ob bereits geschossen
          wurde und
83    //schießt dementsprechend eine neuen Schuss an der X-Koordinate des
          Spieler kreiert.
84    if (key == ' ') {
85       if (!bullet.isShooting()) {
86          bullet.shoot(player.getXPosition());
87       }
88    }
89 }
```

Auf der nächsten Seite geht's weiter...

```
90 //Diese Methode wird immer aufgerufen wenn eine Taste losgelassen wird.
91 //Ist die Taste ein 'a' oder 'd' wird die Richtung des Spieler auf 0
       gesetzt.
92 void keyReleased() {
93    if (key == 'a' || key == 'd') {
94       player.setDirection(0);
95    }
96 }
```

Gummiball

4.1 globale Variablen

Die wichtigsten Variablen auf einen Blick, überall im Code zugänglich.

```
1  // Anfangsposition
2  float x = 300;
3  float y = 100;

4  // Anfangsgeschwindigkeit
5  float ySpeed = 0;
6  float xSpeed = 12;

7  // Ballradius
8  float radius = 40;

9  void setup() {
10    size(600, 600); // Fenstergröße festlegen
11    noStroke(); // Keine Outlines
12 }
```

4.2 draw

Die `draw`-Funktion ist in diesem Fall für das Verhalten des Balls verantwortlich. Er muss von den beiden Wänden und vom Boden abprallen, auf Mausklicks reagieren, und Stück für Stück seine Geschwindigkeit verlieren.

```
1  void draw() {
2     background(100); // Hintergrund zeichnen

3     // Fallbeschleunigung (Erhöhe Geschwindigkeit nach unten)
4     ySpeed += 1;

5     // Position je nach Geschwindigkeit verändern, um Bewegung zu simulieren
6     x += xSpeed;
7     y += ySpeed;

8     // Geschwindigkeit verringern, um Reibung zu simulieren
9     xSpeed *= 0.99;
10    ySpeed *= 0.99;

11    // Abprallen vom Boden
12    if (y>height -radius) { // Wenn in Berührung mit Boden
13       y = height -radius; // Korrigiere Position, falls Ball "im" Boden ist
14       if (ySpeed>0) // Falls Geschwindigkeit noch nicht invertiert
15       ySpeed *= -0.8; // Invertiere horizontale Geschwindigkeit(
             gleichzeitig wird die Geschwindigkeit reduziert)
16    }
```

Auf der nächsten Seite geht's weiter...

```
17      // Abprallen von der rechten Wand
18      if (x>width -radius) { // Wenn in Berührung mit rechter Wand
19          x = width-radius; // Korrigiere Position, falls Ball "in" der Wand
                ist
20          if (xSpeed>0) // Falls Geschwindigkeit noch nicht invertiert
21          xSpeed *= -0.8; // Invertiere horizontale  Geschwindigkeit(
                gleichzeitig wird die Geschwindigkeit reduziert)
22      }

23      // Abprallen von der linken Wand
24      if (x< radius) { // Wenn in Berührung mit linker Wand
25          x = radius; // Korrigiere Position, falls Ball "in" der Wand ist
26          if (xSpeed<0) // Falls Geschwindigkeit noch nicht invertiert
27          xSpeed *= -0.8; // Invertiere horizontale Geschwindigkeit(
                gleichzeitig wird die Geschwindigkeit reduziert)
28      }

29      // Falls Maustaste gedrückt ist
30      if (mousePressed && mouseButton==LEFT) {
31          // In Richtung Maus beschleunigen
32          xSpeed = (mouseX -x)/3;
33          ySpeed = (mouseY -y)/3;
34      }

35      // Ball an aktueller Position zeichnen
36      circle(x, y, 2*radius);
37  }
```

506

Flappy Bird

5.1 Der Vogel

Die Klasse `Bird` mit den Methoden `update()`, `jump()`, `show()`, `hover()`.

```
1  final static int birdX = 100; //x position of the bird
2  final static float gravity = 0.1;

3  final static color background = #6464FF;
4  final static color birdBody = #F4F002;
5  final static color birdBodyOutline = #000000;
6  final static color birdEye = #000000;
7  final static color birdEyeOutline = #FFFFFF;
8  final static color birdMouth = #AD2227;
9  final static color birdMouthOutline = #8C1C1F;

10 // Game Variables
11 Bird bird;

12 void setup() {
13    size(600, 800);
14    pixelDensity(2);
15    strokeWeight(5);
16    textAlign(CENTER);

17    bird = new Bird();
18 }

19 void draw() {
20    background(background);
21    bird.show();
22    bird.hover();
23    bird.update();
24 }

25 void keyReleased() {
26    bird.jump();
27 }

28 class Bird {
29    float x, y, vy;
30    int jumpingCounter;
31    int hoverValue;

32    Bird() {
33       this.x = birdX;
34       this.y = height / 2;
35       this.vy = 0;
36       jumpingCounter = 0;
37       hoverValue = 0;
38    }

39    void update() {
40       if (jumpingCounter > 0) jumpingCounter--;
41       vy += gravity;
42       y += vy;
43    }
```

```
    void show() {
        pushMatrix();
        translate(x, y);
        rotate(radians(vy *10)); //rotate based on y velocity

        fill(birdBody);
        stroke(birdBodyOutline);
        circle(0, 0, 50); //body

        fill(birdEye);
        stroke(birdEyeOutline);
        circle(5, -10, 10); //eye

        fill(birdMouth);
        stroke(birdMouthOutline);
        triangle(20, -5, 20, 5, 30, 0);
        popMatrix(); //beak
    }

    void jump() {
        if (jumpingCounter == 0) {
            vy = -4;
            jumpingCounter = 10;
        }
    }
    void hover() {
        y += sin(hoverValue++ / 10.0) *1.5;
    }
}
```

5.2 Die Röhren

Die Klasse `Pipe` mit den Methoden `show(), move()`.
Außerdem wurden Erweiterungen bei den Methoden `void setup()` und `void draw()` gemacht.

```
44 final static float baseSpeed = 1; //starting speed of pipes
45 final static float speedMultiplier = 1.0001; //increases speed each tick
46 final static int pipeWidth = 75; //thickness of pipes
47 final static int connectorThickness = 15; //connector thickness at the end
       of pipes
48 final static int pipeSpace = 125; //space between upper and lower pipe.

49 final static color pipe = #07BF20;
50 final static color pipeOutline = #059118;

51 float speed;
52 Pipe myPipe;

53 void setup() {
54    // size(600, 800);
55    // pixelDensity(2);
56    // strokeWeight(5);
57    // textAlign(CENTER);

58    speed = baseSpeed;
59    myPipe = new Pipe();
60    // bird = new Bird();
61 }

62 void draw() {
63    // background(background);
64    // bird.show();
65    // bird.hover();
66    // bird.update();
67    myPipe.move();
68    myPipe.show();
69 }

70 class Pipe {
71    float x, y;
72    Pipe() {
73       this.x = width + pipeWidth; //places pipe just outside frame
74       this.y = height / 6 + random(height -(height / 6) *2); //vertical
             location
75    }

76    void show() {
77       float top = y -pipeSpace / 2;
78       float bottom = y + pipeSpace / 2;
79       fill(pipe);
80       stroke(pipeOutline);
81       rect(x, 0, pipeWidth *0.8, top); //top pipe
82       rect(x -pipeWidth *0.1, top -connectorThickness, pipeWidth,
             connectorThickness);
83       rect(x, bottom, pipeWidth *0.8, height); //bottom pipe
84       rect(x -pipeWidth *0.1, bottom, pipeWidth, connectorThickness);
85    }

86    void move() {
87       x -= speed;
88    }
89 }
```

5.3 Röhren in der Endlosschleife

```
final static int pipeGenerationRate = 225; //time interval between new
    pipes. lower = more difficult

ArrayList<Pipe> pipes;
int tick;

void setup() {
   // size(600, 800);
   // pixelDensity(2);
   // strokeWeight(5);
   // textAlign(CENTER);

   pipes = new ArrayList<Pipe>();
   tick = 1000; //immediately generates first pipe
   // speed = baseSpeed;
   // bird = new Bird();
}

void draw() {
   // background(background);
   // bird.show();
   // bird.hover();
   // bird.update();

   if (tick++ >= pipeGenerationRate) { //generate pipe if tick is high
       enough
      generatePipe();
      tick = 0;
   }
   loadPipes();
}

void generatePipe() {
   pipes.add(new Pipe());
}

void loadPipes() {
   for (int i = 0; i < pipes.size(); i++) {
      Pipe p = pipes.get(i);
      p.move();
      p.show();
   }
}
```

5.4 Beispiel für ein vollständig ausgearbeitetes Floppy Bird

```
// Settings
final static float baseSpeed = 1; //starting speed of pipes
final static float speedMultiplier = 1.0001; //speedincrease each tick
final static int pipeWidth = 75; //thickness of pipes
final static int connectorThickness = 15; //thickness of pipe-connector
final static int pipeSpace = 225; //space between upper and lower pipe
final static int pipeGenerationRate = 225;
final static int birdX = 100; //x position of the bird
final static float gravity = 0.1;

// Colors
final static color background = #6464FF;
final static color floor = #CCB735;
final static color floorOutline = #8C5b08;

final static color birdBody = #F4F002;
final static color birdBodyOutline = #000000;
final static color birdEye = #000000;
final static color birdEyeOutline = #FFFFFF;
final static color birdMouth = #AD2227;
final static color birdMouthOutline = #8C1C1F;

final static color pipe = #07BF20;
final static color pipeOutline = #059118;

final static color titleText = #EDED53;
final static color gameOverColor = #000000;
final static color gameOverText = #C10508;
final static color scoreColor = #FCFC00;
final static color highScoreIndicator = #8E1D0E;
final static color scoreIsHighScore = #FCCE00;

// Game Variables
ArrayList<Pipe> pipes;
float speed;
int counter;
int tick;
int score = 0;
int highScore = 0;
boolean gameOver;
boolean gameStarted;
Bird bird;
```

```
// Methods
void setup() {
   size(600, 800);
   pixelDensity(2);
   strokeWeight(5);
   textAlign(CENTER);

   speed = baseSpeed;
   tick = 1000; //immediately generates first pipe
   bird = new Bird();
   gameOver = false;
   gameStarted = false;
   pipes = new ArrayList<Pipe>();
   highScore = (score > highScore) ? score : highScore;
   score = 0;
}

void draw() {
   background(background);
   bird.show();
   if (gameOver) { //player died
      showGameOverScreen();
   } else if (gameStarted) { //player alive, game started
      if (tick++ >= pipeGenerationRate) { //generate pipe
         generatePipe();
         tick = 0;
      }
      loadPipes();
      speed *= speedMultiplier;
      bird.update();
      gameOver = bird.deathCheck();
      showScore();
      showFloor();
   } else { //game not started yet
      showStartMenu();
      bird.hover();
      showFloor();
   }
   //saveFrame("mySketch-####.png");
}

void keyPressed() {
   if (gameOver) {
      setup(); //resets the game if gameOver is true
   } else if (gameStarted) {
      bird.jump();
   } else {
      gameStarted = true;
      bird.jump();
   }
}
```

```
38 void showScore() {
39    if (score > highScore) {
40       fill(highScoreIndicator);
41       textSize(15);
42       text("HIGHSCORE!", 100, 120);
43       fill(scoreIsHighScore);
44    } else {
45       fill(scoreColor);
46    }
47    textSize(75);
48    text(score, 100, 100);
49 }

50 void showStartMenu() {
51    fill(titleText);
52    textSize(60);
53    text("Floppy Bird", width / 2, height / 4);
54    textSize(20);
55    text("Press any key to play", width / 2, height / 4 + 50);
56    text("Highscore: " + highScore, width / 2, height / 4 + 75);
57 }

58 void showGameOverScreen() {
59    background(gameOverColor);
60    fill(gameOverText);
61    textSize(50);
62    text("Game Over", width / 2, height / 2);
63    textSize(25);
64    text("Press any key to go back to the menu", width / 2, height / 2 + 50)
          ;
65 }

66 void showFloor() {
67    stroke(floorOutline);
68    fill(floor);
69    rect(0, height-50, width, height);
70 }

71 void generatePipe() {
72    pipes.add(new Pipe());
73 }

74 void loadPipes() {
75    for (int i = 0; i < pipes.size(); i++) {
76       Pipe p = pipes.get(i);
77       p.move();
78       p.show();
79       if (p.x < -pipeWidth) { //remove pipes that left the screen
80          pipes.remove(i--); //i must be decremented, otherwise you would
               skip the following pipe
81          score++;
82       }
83    }
84 }
```

```
// Classes
class Pipe {
   float x, y;
   Pipe() {
      this.x = width + pipeWidth; //places pipe just outside frame
      this.y = height / 6 + random(height -(height / 6) *2); //vertical location
   }

   void show() {
      float top = y -pipeSpace / 2;
      float bottom = y + pipeSpace / 2;
      fill(pipe);
      stroke(pipeOutline);
      rect(x, 0, pipeWidth *0.8, top); //top pipe
      rect(x -pipeWidth *0.1, top -connectorThickness, pipeWidth, connectorThickness);
      rect(x, bottom, pipeWidth *0.8, height); //bottom pipe
      rect(x -pipeWidth *0.1, bottom, pipeWidth, connectorThickness);
   }

   void move() {
      x -= speed;
   }
}

class Bird {
   float x, y, vy;
   int jumpingCounter;
   int hoverValue;

   Bird() {
      this.x = birdX;
      this.y = height / 2;
      this.vy = 0;
      jumpingCounter = 0;
      hoverValue = 0;
   }

   void update() {
      if (jumpingCounter > 0)
      jumpingCounter--;
      vy += gravity;
      y += vy;
   }

   void show() {
      pushMatrix();
      translate(x, y);
      rotate(radians(vy *10)); //rotate based on y velocity

      fill(birdBody);
      stroke(birdBodyOutline);
      circle(0, 0, 50); //body

      fill(birdEye);
      stroke(birdEyeOutline);
      circle(5, -10, 10); //eye

      fill(birdMouth);
      stroke(birdMouthOutline);
      triangle(20, -5, 20, 5, 30, 0);
      popMatrix(); //beak
   }
```

```
85      void jump() {
86          if (jumpingCounter == 0) {
87              vy = -4;
88              jumpingCounter = 10;
89          }
90      }

91      boolean deathCheck() {
92          for (int i = 0; i < pipes.size(); i++) {
93              Pipe p = pipes.get(i);
94              if (x > p.x && x < p.x + pipeWidth) { //"is bird within a pipe?"
95                  if (y < p.y -pipeSpace / 2 || y > p.y + pipeSpace / 2) //"is
                        bird not between the pipes?"
96                  return true;
97              }
98          }
99          if (bird.y > height -50) //"is bird below ground?"
100         return true;
101         return false;
102     }

103     void hover() {
104         y += sin(hoverValue++ / 10.0) *1.5;
105     }
106 }
```

512

Feuerwerk

6.1 Rakete

Legt die Attribute und Methoden aller Raketen fest, sodass sie fliegen, explodieren, dargestellt werden und zurückgesetzt werden können.

```
1  class Rocket {
2      // jede Rakete hat eine Explosion
3      Explosion ex;
4
4      // Geschwindigkeit
5      float yv;
6
6      // X Position
7      int xPos;
8
8      // Y Position
9      int yPos;
10
10     // Hoehe, an der die Rakete explodieren soll
11     int explosionHeight;
12
12     // zeigt an ob Rakete explodiert ist
13     boolean exploded;
14
14     // zeigt an ob Explosion abgeschlossen ist
15     boolean expired;
16
16     Rocket(int x) {
17         // Rakete ist noch nicht explodiert
18         this.exploded = false;
19
19         // Explosion ist noch nicht abgeschlossen
20         this.expired = false;
21
21         // Geschwindigkeit wird zufällig gesetzt
22         this.yv = random(minVelocity, maxVelocity);
23
23         // X Position wird übergeben
24         this.xPos = x;
25
25         // Raketen starten immer am unteren Fensterrand
26         this.yPos = height;
27
27         //Explosionshoehe wird zufaellig gewaehlt
28         this.explosionHeight = int(random(height*0.1, height*0.4));
29     }
```

Auf der nächsten Seite geht's weiter...

```
34      void update() {
35          if (!expired) {
36              if (exploded) {
37                  //updated Explosion und vermerkt, ob diese abgeschlossen // ist
38                  expired = ex.update();
39              } else {
40                  if (this.yPos <= explosionHeight) {
41                      // dann wird an der Raketenposition eine neue
42                      // Explosion erschaffen
43                      ex = new Explosion(xPos, yPos);
44                      exploded = true;
45                  } else {
46                       // Rakete bewegt sich nach oben mit Geschwindigkeit // yv
47                      this.yPos -= yv;
48                      // zeichnet Rakete
49                      drawRocket();
50                  }
51              }
52          } else {
53              resetRocket(); //setzt Rakete zurueck
54          }
55      }
56      void resetRocket() { //setzt Rakete zurueck (Sieht fuer Beobachter aus,
            als ob eine neue Rackete entstanden ist)
57          this.exploded = false;
58          this.expired = false;
59          //Geschwindigkeit wird zufällig gesetzt
60          this.yv = random(minVelocity, maxVelocity);
61          // X Position wird zufaellig neu gesetzt
62          this.xPos = int(random(10, width-10));
63          // Raketen starten wieder am unteren Fensterrand
64          this.yPos = height;
65          //Explosionshoehe wird zufaellig gewaehlt
66          this.explosionHeight = int(random(height*0.1, height*0.4));
67      }
68      void drawRocket() {
69          // zufaellige Farbe für Raketenkopf
70          fill(color(random(255), random(255), random(255)));
71          // Raketenkopf
72          circle(xPos, yPos, 5);
73          // Der Raketenschweif ist Gelb und besteht aus insgesamt 4
74          // Kreisen hinereinander
75          fill(color(255, 255, 0));
76          circle(xPos, yPos+5, 4);
77          circle(xPos, yPos+4, 3);
78          circle(xPos, yPos+3, 2);
79          circle(xPos, yPos+2, 1);
80      }
81  }
```

6.2 Explosion

Eine Explosion - das ist eine Menge Partikel! Aber nicht nur: auch die Explosionsposition, das Alter der Explosion, und wie viele Partikel bereits ausserhalb des Fensters sind, sind von Bedeutung.

In Methoden müssen oft Aufrufe getätigt werden, bei denen jeden Partikel einer Explosion zB. `update` aufgerufen wird.

```
1  class Explosion {
2      //Anzahl der Partikel ausgehend von der Explosion
3      int particle_count = 800; // Könnte man auch random machen
4      //Variable, um Frames seit Explosion zu zaehlen (benutzt für
5      // Partikelposition)
6      int tick;
7      // Counter der die Partikel zaehlt, welche aus dem Fenster gefallen //
           sind
8      int deadCounter;
9      Particle[] particles; //Array das die Partikel enthaelt
10     Explosion(int x, int y) { // Konstruktor
11         // Tick wird zurueck gesetzt
12         tick = 0;
13         //Erstellung des Partikelarrays
14         particles = new Particle[particle_count];
15         //Füllt jede Stelle des Arrays (Initialisierung des Arrays)
16         for (int i = 0; i < particles.length; i++) {
17             // Jedes Partikel startet am Explosionspunkt
18             particles[i] = new Particle(x, y);
19         }
20     }
21     // Updated alle Partikel im Array und liefert Boolean zum Check, ob //
           die Explosion abgeschlossen ist
22     boolean update() {
23         deadCounter = particle_count;
24         for (Particle p : particles) {
25             // Ist das Partikel unterhalb des unteren Fensterrahmen?
26             // Dann ignorieren.
27             if (p.yPos < height ) {
28                 // ruft update() des Partikels auf und übergibt die tick //
                       variable
29                 p.update(tick);
30                 // fuer jedes Partikel, das noch im Fenster ist, wird
31                 // deadcounter verringert. Es bleibt die Anzahl jener
32                 // Partikel, die nicht im Fenster sind.
33                 deadCounter --;
34             }
35         }
36         //tick erhöht sich, nachdem jedes Parikel updated wurde
37         tick++;
```

Auf der nächsten Seite geht's weiter...

```
38          if (deadCounter >= particle_count *0.9 || tick >= 53) {
39              //ist ein Grossteil der Partikel aus dem Bild gefallen? ODER
40          //ist die Explosion aelter als 52 Frames?
41              return true;
42          } else {
43              return false;
44          }
45      }
46  }
```

6.3 Partikel

Die kleinsten Objekte, mit denen wir uns hier beschäftigen, haben im Gegensatz zu Raketen eine gebogene Flugbahn.

Darum brauchen sie noch ein paar weitere Attribute, beispielsweise einen Faktor, der das Einwirken der Schwerkraft simuliert.

```
1   class Particle {

2       // Y Startpunkt des Partikel
3       float ystart;

4       // X Startpunkt des Partikel
5       float xstart;

6       // Geschwindigkeit
7       float velocity;

8       // Partikelgroesse
9       int particleSize = 3;

10      // X Position
11      float xPos;

12      // Y Position
13      float yPos;

14      // Richtung der Flugbahn
15      float direction;

16      Particle(int xstart, int ystart) {

17          // Geschwindigkeit wird zufällig gesetzt
18          this.velocity = random(minVelocity, maxVelocity);

19          // Partikel startet am X-Wert des Startpunkts, spaeter
20          // aendert sich dieser Wert
21          this.xPos = xstart;
```

Auf der nächsten Seite geht's weiter...

```
29        // Partikel startet am Y-Wert des Startpunkts, spaeter
30        // aendert sich dieser Wert
31        this.yPos = ystart;

32        // X Startwert des Partikel wird gespeichert (fuer
33        // Positionsberechnung) dieser Wert wird nicht veraendert
34        this.xstart = xstart;

35        // Y Startwert des Partikel wird gespeichert (fuer //
              Positionsberechnung) dieser Wert wird nicht veraendert
36        this.ystart = ystart;

37        // Richtung der Flugbahn wird zufällig gewählt (0 bis (2*pi) // = 360
              Grad)
38        this.direction = random(0, 2*PI);
39    }

40    void update(int tick) {
41        // Ergibt die Sekunden seit der Explosion (ausgehend von der
42        // Annahme, dass es einmal pro Frame aufgerufen wird)
43        float seconds = (tick/frameRate);

44        // das Zeit-Orts-Gesetz der gleichförmigen Bewegung in
45        // x-Richtung
46        this.xPos = xstart + velocity *cos(direction) *seconds;

47        // das Zeit-Orts-Gesetz der gleichförmigen Bewegung in
48        // y-Richtung
49        this.yPos = ystart + (velocity *sin(direction)) *seconds -0.5 *
              gravity *sq(seconds);

50        // Partikel bekommen zufaellige Farbe, die mit jedem Tick
51        // dunkler wird
52        fill(color(random(255), random(255), random(255), (255-(tick*5))));

53        // Zeichnen der Partikel
54        circle(this.xPos, this.yPos, particleSize);

55    }

56 }
```

6.4 Feuerwerk

Dies ist der Teil des Programms, ohne den nichts läuft: `setup()`, `draw()` und die wichtigsten Variablen.

```
1  int margin = 10; // Abstand für Raketen zum Bildrand
2  int rocketCount = 10; // Raketenanzahl
3  float minVelocity = 5; // Minimale Geschwindigkeit der Partikel
4  float maxVelocity = 150;   // Maximale
5  float gravity = -220; // Stärke der Gravitation (ist negativ da Y-
       Koordinaten nach unten steigen)

6  Rocket[] rockets; // Array für Raketen

7  void setup() {
8     frameRate(60); // Framerate
9     size(1200,1000); // Fenstergröße
10    background(0); // Hintergrund
```

```
11      rockets = new Rocket[rocketCount]; // Initialisierung des Raketen-Arrays
```

Auf der nächsten Seite geht's weiter...

```
10  // für jede Stelle im Raketen-Array wird eine neue Rakete erstellt
11     for (int i = 0; i< rockets.length; i++) {
12        //neue Rakete erhält eine zufaellige x-Position
13        rockets[i] = new Rocket(int(random(margin, width-margin)));
14     }
15  }

16  void draw() {
17     background(0); // Hintergrund
18     for (Rocket rock : rockets) { // update jede Rakete im Array
19        rock.update();
20     }
21  }
```

508

PI-Rechner

7.1 Vorbereitung und Logik

Die wichtigsten Variablen auf einen Blick, überall im Code zugänglich und das setup. Die Klasse PVektor hat die Attribute (float x, float y) und dient als Punkt im Koordinatensystem.

```
1  PVector center; // Kreismittelpunkt

2  float r = 200; // Radius
3  int n = 2; // Beginnwert der Stücke, in die der Kreis geteilt wird.

4  void setup() {
5     size(800, 600);
6     //Der Mittelpunkt unseres Kreises soll in der Mitte des Fensters liegen.
7     center = new PVector(width/2, height/2);
8  }

9  void draw() {
10    //Je weiter rechts der Mauszeiger ist, in desto mehr Teile wird der
          Kreis aufgeteilt.
11    n = mouseX / (width/30) + 2;

12    background(0);
13    noFill();
14    stroke(255);

15    strokeWeight(4);

16    circle(center.x, center.y, 2*r);
17    // Die Verschiebung, die einen Punkt vom Mittelpunkt auf den Zirkel
          unseres Kreises setzt.
18    PVector p = new PVector(0, -r);
19    for (int i = 0; i<n; i++) {
20       stroke(255);
21       // zeichnet die durch den Kreis führende Linie, um ihn in Dreiecke
             einzuteilen
22       line(center.x, center.y, p.x+center.x, p.y+center.y);

23       // Wir speichern den aktuellen Punkt auf dem Zirkel als alten Punkt
24       PVector pPrev = p;
25       // Wir wandern einen Schritt weiter entlang des Zirkels.
26       // Unser neuer Punkt ist dazu auf dem Zirkel um den Grad 2pi/n ==
             360/n zu verschieben.
27       p = rotZ(p, 360/n);

28       // rote Linien, die dem Umfang des Kreises nachzeichnen
29       stroke(255,0,0);
30       // Die Linien zeichnen wir vom letzten Punkt auf dem Zirkel zum
             aktuellen Punkt auf dem Zirkel.
31       line(pPrev.x+center.x, pPrev.y+center.y, p.x+center.x, p.y+center.y);
32    }

33    // Unser aktueller Winkel der Dreiecke nach der Formel 2pi/n == 360/n
34    float alpha = 360/(2*n);
35    // Errechnet die Länge der (bei uns in rot dargestellte) Gegenkathete zu
          a der rechtwinkligen Dreiecke
36    float opposite = sin(radians(alpha))*r;
37    // aktuell errechneter Umfang aus den Gegenkatheten zu a der
          rechtwinkligen Dreiecke (von den es n*2 gibt)
38    float circumference = opposite *2*n;
39    // Errechnung von pi
40    float pi = circumference / (2*r);
```

Auf der nächsten Seite geht's weiter...

```
41    // Ausgabe des für pi errechneten Wertes im Fenster
42    textSize(100);
43    text(pi, 0, 100);
44    // Ausdrucken des errechneten Wertes in der Konsole
45    print(pi + "\n");
46 }
```

7.2 Verschieben eines Punktes

```
1  // Schiebt einen Punkt auf einer Kreisbahn um den Winkel angle weiter
       rotiert dabei um den Koordinatenmittelpunkt.
2  PVector rotZ(PVector pos, float angle) {
3     //Verschiebung auf x
4     float x = pos.x *cos(radians(angle)) + pos.y *sin(radians(angle));
5     //Verschiebung auf y
6     float y = pos.x *-sin(radians(angle)) + pos.y *cos(radians(angle));
7     return new PVector(x, y);
8  }
```

503

Game of Life

```
// Game of Life

int resolution = 5;
int size = 200;
int genCounter = 0;

// zwei Boards, welche sich gegenseitig immer wieder überschreiben
boolean[][] board1 = new boolean[size][size];
boolean[][] board2 = new boolean[size][size];


void setup() {
   size(1000, 1000); // resolution *size
   noStroke();
   frameRate(32);

   // das erste Board einmal mit zufälligen Werten füllen
   for (int i = 0; i < size; i++) {
      for (int j = 0; j < size; j++) {
         board1[i][j] = random(1) >= 0.5;
      }
   }
}

void draw() {
   if (genCounter % 2 == 0) { // beim ersten Durchlauf, und sonst jeder
       zweite
      drawBoard(board1);
      nextGen(board1, board2);
   } else { // ab dem zweiten Durchlauf, jeder zweite
      drawBoard(board2);
      nextGen(board2, board1);
   }
   drawText();
   genCounter ++;
}

void drawText() {
   fill(120, 120, 120);
   rect(width / 2 -60, 5, 120, 60);
   textSize(45);
   textAlign(CENTER);
   fill(0);
   text(genCounter, width / 2, 50);
}

// Visualisierung des Boards
void drawBoard(boolean[][] board) {
   for (int i = 0; i < size; i++) {
      for (int j = 0; j < size; j++) {
         if (board[i][j]) fill(0); // anhand des boolean wird entschieden
             welche Farbe verwendet wird, true = schwarz
         else fill(255);
         rect(i *resolution, j *resolution, resolution, resolution);
      }
   }
}
```

```java
// das neue Brett wird beladen anhand der Daten aus dem alten Brett
void nextGen(boolean[][] boardOld, boolean[][] boardNew) {
   for (int i = 0; i < size; i++) {
      for (int j = 0; j < size; j++) {
         if (boardOld[i][j] && (neighbours(boardOld, i, j) == 2 ||
             neighbours(boardOld, i, j) == 3)) boardNew[i][j] = true; //
             lebendig && 2 || 3 -> true
         else if (neighbours(boardOld, i, j) == 3) boardNew[i][j] = true;
             // tod && 3 -> true
         else boardNew[i][j] = false;
      }
   }
}

// Berechnung der Anzahl an Nachbarn um die Regeln des Game of Life
    anzuwenden
// um einen IndexOutOfBounds zu vermeiden prüfen wir zuerst ob die werte
    zwischen 0 und size liegen
// continue wird verwendet um direkt in den nächsten for-Durchlauf zu gehen
int neighbours(boolean[][] board, int i, int j) {
   int counter = 0;
   for (int a = -1; a < 2; a++) {
      if (!(i + a < size && i + a >= 0)) continue;
      for (int b = -1; b < 2; b++) {
         if (!(j + b < size && j + b >= 0)) continue;
         if (board[i + a][j + b] && !(a == 0 && b == 0)) counter++;
      }
   }
   return counter;
}
```

505

Schwarmverhalten

```processing
int insektenAnzahl = 20;
float insektMaxSpeed = 10;

// Dieser Array enthält alle Mueckenobjekte
Insekt insekten[] = new Insekt[insektenAnzahl];

void setup() {
  size(400, 400);

  // Erstelle Mücken an zufälligen Positionen
  for (int i = 0; i<insektenAnzahl; i++) {
    insekten[i] = new Insekt(random(width), random(height));
  }
}

void draw() {
  // Hintergrund
  background(160, 160, 250);

  // Bewege Muecken und zeichne sie an
  for (int i = 0; i<insekten.length; i++) {
    for(int j = i+1; j<insekten.length; j++) {
      // Die verschachtelte Schleife sorgt dafür, dass jede Muecke sich von
          jeder anderen abstoßt
      insekten[i].collide(insekten[j]);
    }
    insekten[i].update();
  }
}
```

```
24  class Insekt {
25    float x;
26    float y;
27    float xSpeed;
28    float ySpeed;
29    float maxSpeed = insektMaxSpeed;

30    Insekt(float x, float y) {
31      this.x = x;
32      this.y = y;
33    }

34    // bei geringem Abstand beschleunige in entgegengesetzte Richtung
35    void collide(Insekt m) {
36      if(dist(x,y,m.x,m.y)<10) {
37        xSpeed -= (m.x-x)/2/mag(m.x-x,m.y-y)*10;
38        ySpeed -= (m.y-y)/2/mag(m.x-x,m.y-y)*10;
39        m.xSpeed -= (x-m.x)/2/mag(m.x-x,m.y-y)*10;
40        m.ySpeed -= (y-m.y)/2/mag(m.x-x,m.y-y)*10;
41      }
42    }

43    void update() {
44      // bewege Muecke
45      x += xSpeed;
46      y += ySpeed;

47      // beschleunige in Richtung Mauszeiger
48      xSpeed += (mouseX -x) / 100;
49      ySpeed += (mouseY -y) / 100;

50      // begrenze Geschwindigkeit auf maxSpeed
51      xSpeed = constrain(xSpeed, -maxSpeed, maxSpeed);
52      ySpeed = constrain(ySpeed, -maxSpeed, maxSpeed);

53      // zeichne Muecke
54      drawInsekt();
55    }

56    void drawInsekt() {
57      fill(0);
58      circle(x, y, 5);
59    }
60  }
```

507

Minesweeper

10.1 Spielfeld

```
1  int fieldWidth = 50;
2  int fieldHeight = 50;
3  Board b;

4  void setup() {
5     size(800, 800);
6     b = new Board();
7  }

8  void draw() {
9     background(200);
10    for (int i = 0; i < 16; i++) {
11       fill(0);
12       line(fieldWidth *i, 0, fieldHeight *i, height);
13       line(0, fieldHeight *i, width, fieldHeight *i);
14    }
15 }

16 class Board {
17    Field[][] board = new Field[16][16];

18    Board() {
19       for (int i = 0; i < board.length; i++) {
20          for (int j = 0; j < board[0].length; j++) {
21             board[i][j] = new Field(i, j);
22          }
23       }
24    }
25 }


26 class Field {
27    int x, y;

28    Field(int x, int y) {
29       this.x = x;
30       this.y = y;
31    }
32 }
```

10.2 Die Bomben

```
1  int fieldWidth = 50;
2  int fieldHeight = 50;
3  int bombCount = 20;
4  Board b;

5  void setup() {
6     size(800, 800);
7     b = new Board();
8     b.generateBombs(bombCount);
9  }

10 void draw() {
11    background(200);
12    for (int i = 0; i < 16; i++) {
13       fill(0);
14       line(fieldWidth *i, 0, fieldHeight *i, height);
15       line(0, fieldHeight *i, width, fieldHeight *i);
16    }
17    b.show();
18 }

19 class Board {
20    Field[][] board = new Field[16][16];

21    Board() {
22       for (int i = 0; i < board.length; i++) {
23          for (int j = 0; j < board[0].length; j++) {
24             board[i][j] = new Field(i, j);
25          }
26       }
27    }

28    void show() {
29       for (int i = 0; i < board.length; i++) {
30          for (int j = 0; j < board[0].length; j++) {
31             board[i][j].show();
32          }
33       }
34    }

35    void generateBombs(int num) {
36       for (int i = 0; i < num; i++) {
37          int ranX = int(random(16));
38          int ranY = int(random(16));
39          if (!board[ranX][ranY].isBomb) {
40             board[ranX][ranY].isBomb = true;
41          } else {
42             i--;
43          }
44       }
45    }
46 }


47 class Field {
48    int x, y;
49    boolean isBomb;

50    Field(int x, int y) {
51       this.x = x;
52       this.y = y;
53    }
```

```
54      void show() {
55          fill(255);
56          rect(x *50, y *50, 50, 50);
57          if (this.isBomb) {
58              fill(255, 0, 0);
59              circle(this.x *50 + 25, this.y *50 + 25, 40);
60          }
61      }
62  }
```

10.3 Die umliegenden Bomben eines Feldes

```
int fieldWidth = 50;
int fieldHeight = 50;
int bombCount = 20;
Board b;

void setup() {
   size(800, 800);
   b = new Board();
   b.generateBombs(bombCount);
   b.getBombCount();
}

void draw() {
   background(200);
   for (int i = 0; i < 16; i++) {
      fill(0);
      line(fieldWidth *i, 0, fieldHeight *i, height);
      line(0, fieldHeight *i, width, fieldHeight *i);
   }
   b.show();
}

class Board {
   Field[][] board = new Field[16][16];

   Board() {
      for (int i = 0; i < board.length; i++) {
         for (int j = 0; j < board[0].length; j++) {
            board[i][j] = new Field(i, j);
         }
      }
   }

   void show() {
      for (int i = 0; i < board.length; i++) {
         for (int j = 0; j < board[0].length; j++) {
            board[i][j].show();
         }
      }
   }

   void generateBombs(int num) {
      for (int i = 0; i < num; i++) {
         int ranX = int(random(16));
         int ranY = int(random(16));
         if (!board[ranX][ranY].isBomb) {
            board[ranX][ranY].isBomb = true;
         } else {
            i--;
         }
      }
   }

   void getBombCount() {
      for (int i = 0; i < board.length; i++) {
         for (int j = 0; j < board[0].length; j++) {
            int counter = 0;
            for (int offsetX = -1; offsetX < 2; offsetX++) {
               for (int offsetY = -1; offsetY < 2; offsetY++) {
                  if (offsetX == 0 && offsetY == 0) continue;
                  if (i + offsetX >= 0 && i + offsetX < board.length && j +
                       offsetY >= 0 && j + offsetY < board.length && board[
                      i + offsetX][j + offsetY].isBomb) counter++;
```

```
55                  }
56              }
57              board[i][j].surroundingBombs = counter;
58          }
59      }
60  }
61  }

62  class Field {
63      int x, y, surroundingBombs;
64      boolean isBomb;

65      Field(int x, int y) {
66          this.x = x;
67          this.y = y;
68      }

69      void show() {
70          fill(255);
71          rect(x *50, y *50, 50, 50);
72          if (this.isBomb) {
73              fill(255, 0, 0);
74              circle(this.x *50 + 25, this.y *50 + 25, 40);
75          } else if (this.surroundingBombs > 0) {
76              fill(0);
77              textSize(20);
78              textAlign(CENTER);
79              text(surroundingBombs, x *50 + 25, y *50 + 25);
80          }
81      }
82  }
```

10.4 Das Aufdecken der Felder

```
1  int fieldWidth = 50;
2  int fieldHeight = 50;
3  int bombCount = 20;
4  Board b;

5  void setup() {
6     size(800, 800);
7     b = new Board();
8     b.generateBombs(bombCount);
9     b.getBombCount();
10 }

11 void draw() {
12    background(200);
13    for (int i = 0; i < 16; i++) {
14       fill(0);
15       line(fieldWidth *i, 0, fieldHeight *i, height);
16       line(0, fieldHeight *i, width, fieldHeight *i);
17    }
18    b.show();
19 }

20 void mouseReleased() {
21    if (mouseButton == LEFT) {
22       b.board[mouseX / 50][mouseY / 50].reveal();
23    }
24 }

25 class Board {
26    Field[][] board = new Field[16][16];

27    Board() {
28       for (int i = 0; i < board.length; i++) {
29          for (int j = 0; j < board[0].length; j++) {
30             board[i][j] = new Field(i, j);
31          }
32       }
33    }

34    void show() {
35       for (int i = 0; i < board.length; i++) {
36          for (int j = 0; j < board[0].length; j++) {
37             board[i][j].show();
38          }
39       }
40    }

41    void generateBombs(int num) {
42       for (int i = 0; i < num; i++) {
43          int ranX = int(random(16));
44          int ranY = int(random(16));
45          if (!board[ranX][ranY].isBomb) {
46             board[ranX][ranY].isBomb = true;
47          } else {
48             i--;
49          }
50       }
51    }

52    void getBombCount() {
53       for (int i = 0; i < board.length; i++) {
54          for (int j = 0; j < board[0].length; j++) {
55             int counter = 0;
```

```
56              for (int offsetX = -1; offsetX < 2; offsetX++) {
57                  for (int offsetY = -1; offsetY < 2; offsetY++) {
58                      if (offsetX == 0 && offsetY == 0) continue;
59                      if (i + offsetX >= 0 && i + offsetX < board.length && j +
                             offsetY >= 0 && j + offsetY < board.length && board[
                            i + offsetX][j + offsetY].isBomb) counter++;
60                  }
61              }
62              board[i][j].surroundingBombs = counter;
63          }
64      }
65  }
66  }

67  class Field {
68      int x, y, surroundingBombs;
69      boolean isBomb, isRevealed;

70      Field(int x, int y) {
71          this.x = x;
72          this.y = y;
73      }

74      void reveal() {
75          if (this.isRevealed) return;
76          isRevealed = true;
77      }

78      void show() {
79          if (isRevealed) {
80              fill(255);
81              rect(x *50, y *50, 50, 50);
82              if (this.isBomb) {
83                  fill(255, 0, 0);
84                  circle(this.x *50 + 25, this.y *50 + 25, 40);
85              } else if (this.surroundingBombs > 0) {
86                  fill(0);
87                  textSize(20);
88                  textAlign(CENTER);
89                  text(surroundingBombs, x *50 + 25, y *50 + 25);
90              }
91          }
92      }
93  }
```

10.5 Das Aufdecken von Feldern ohne angrenzende Bomben

```
1  int fieldWidth = 50;
2  int fieldHeight = 50;
3  int bombCount = 20;
4  Board b;

5  void setup() {
6     size(800, 800);
7     b = new Board();
8     b.generateBombs(bombCount);
9     b.getBombCount();
10 }

11 void draw() {
12    background(200);
13    for (int i = 0; i < 16; i++) {
14       fill(0);
15       line(fieldWidth *i, 0, fieldHeight *i, height);
16       line(0, fieldHeight *i, width, fieldHeight *i);
17    }
18    b.show();
19 }

20 void mouseReleased() {
21    if (mouseButton == LEFT) {
22       b.board[mouseX / 50][mouseY / 50].reveal();
23    }
24 }

25 class Board {
26    Field[][] board = new Field[16][16];

27    Board() {
28       for (int i = 0; i < board.length; i++) {
29          for (int j = 0; j < board[0].length; j++) {
30             board[i][j] = new Field(i, j);
31          }
32       }
33    }

34    void show() {
35       for (int i = 0; i < board.length; i++) {
36          for (int j = 0; j < board[0].length; j++) {
37             board[i][j].show();
38          }
39       }
40    }

41    void generateBombs(int num) {
42       for (int i = 0; i < num; i++) {
43          int ranX = int(random(16));
44          int ranY = int(random(16));
45          if (!board[ranX][ranY].isBomb) {
46             board[ranX][ranY].isBomb = true;
47          } else {
48             i--;
49          }
50       }
51    }

52    void getBombCount() {
53       for (int i = 0; i < board.length; i++) {
54          for (int j = 0; j < board[0].length; j++) {
55             int counter = 0;
```

```
56              for (int offsetX = -1; offsetX < 2; offsetX++) {
57                  for (int offsetY = -1; offsetY < 2; offsetY++) {
58                      if (offsetX == 0 && offsetY == 0) continue;
59                      if (i + offsetX >= 0 && i + offsetX < board.length && j +
                             offsetY >= 0 && j + offsetY < board.length && board[
                            i + offsetX][j + offsetY].isBomb) counter++;
60                  }
61              }
62              board[i][j].surroundingBombs = counter;
63          }
64      }
65  }
66  }

67  class Field {
68      int x, y, surroundingBombs;
69      boolean isBomb, isRevealed;

70      Field(int x, int y) {
71          this.x = x;
72          this.y = y;
73      }

74      void reveal() {
75          if (this.isRevealed) return;
76          isRevealed = true;
77          if (this.surroundingBombs == 0 && !this.isBomb) {
78              for (int offsetX = -1; offsetX < 2; offsetX++) {
79                  for (int offsetY = -1; offsetY < 2; offsetY++) {
80                      if (offsetX == 0 && offsetY == 0) continue;
81                      if (x + offsetX >= 0 && x + offsetX < 16 && y + offsetY >= 0
                             && y + offsetY < 16) {
82                          b.board[x + offsetX][y + offsetY].reveal();
83                      }
84                  }
85              }
86          }
87      }

88      void show() {
89          if (isRevealed) {
90              fill(255);
91              rect(x *50, y *50, 50, 50);
92              if (this.isBomb) {
93                  fill(255, 0, 0);
94                  circle(this.x *50 + 25, this.y *50 + 25, 40);
95              } else if (this.surroundingBombs > 0) {
96                  fill(0);
97                  textSize(20);
98                  textAlign(CENTER);
99                  text(surroundingBombs, x *50 + 25, y *50 + 25);
100             }
101         }
102     }
103 }
```

10.6 Spielende

```
1  int fieldWidth = 50;
2  int fieldHeight = 50;
3  int bombCount = 20;
4  Board b;

5  void setup() {
6     size(800, 800);
7     b = new Board();
8     b.generateBombs(bombCount);
9     b.getBombCount();
10 }

11 void draw() {
12    background(200);
13    for (int i = 0; i < 16; i++) {
14       fill(0);
15       line(fieldWidth *i, 0, fieldHeight *i, height);
16       line(0, fieldHeight *i, width, fieldHeight *i);
17    }
18    b.show();
19 }

20 void mouseReleased() {
21    if (!b.gameOver && mouseButton == LEFT) {
22       b.board[mouseX / 50][mouseY / 50].reveal();
23    }
24 }



25 class Board {
26    Field[][] board = new Field[16][16];
27    boolean gameOver, isWon;

28    Board() {
29       for (int i = 0; i < board.length; i++) {
30          for (int j = 0; j < board[0].length; j++) {
31             board[i][j] = new Field(i, j);
32          }
33       }
34    }

35    void generateBombs(int num) {
36       for (int i = 0; i < num; i++) {
37          int ranX = int(random(16));
38          int ranY = int(random(16));
39          if (!board[ranX][ranY].isBomb) {
40             board[ranX][ranY].isBomb = true;
41          } else {
42             i--;
43          }
44       }
45    }

46    void getBombCount() {
47       for (int i = 0; i < board.length; i++) {
48          for (int j = 0; j < board[0].length; j++) {
49             int counter = 0;
50             for (int offsetX = -1; offsetX < 2; offsetX++) {
51                for (int offsetY = -1; offsetY < 2; offsetY++) {
52                   if (offsetX == 0 && offsetY == 0) continue;
53                   if (i + offsetX >= 0 && i + offsetX < board.length && j +
                          offsetY >= 0 && j + offsetY < board.length && board[
```

```
                        i + offsetX][j + offsetY].isBomb) counter++;
                }
            }
            board[i][j].surroundingBombs = counter;
        }
    }
}

void isGameOver() {
    if (gameOver) return;
    for (int i = 0; i < board.length; i++) {
        for (int j = 0; j < board[0].length; j++) {
            if (!board[i][j].isRevealed && !board[i][j].isBomb) {
                gameOver = false;
                return;
            }
        }
    }
    gameOver = true;
    isWon = true;
}

void show() {
    isGameOver();
    for (int i = 0; i < board.length; i++) {
        for (int j = 0; j < board[0].length; j++) {
            board[i][j].show();
        }
    }
    if (gameOver) {
        if (isWon) {
            fill(0, 255, 0);
            textSize(50);
            textAlign(CENTER);
            text("gewonnen", 400, 400);
        } else {
            fill(255, 0, 0);
            textSize(50);
            textAlign(CENTER);
            text("verloren", 400, 400);
        }
    }
}
}

class Field {
    int x, y, surroundingBombs;
    boolean isBomb, isRevealed;

    Field(int x, int y) {
        this.x = x;
        this.y = y;
    }

    void reveal() {
        if (this.isRevealed) return;
        if (this.isBomb) {
            b.gameOver = true;
            b.isWon = false;
        }
        isRevealed = true;
        if (this.surroundingBombs == 0 && !this.isBomb) {
            for (int offsetX = -1; offsetX < 2; offsetX++) {
                for (int offsetY = -1; offsetY < 2; offsetY++) {
                    if (offsetX == 0 && offsetY == 0) continue;
```

```
113                 if (x + offsetX >= 0 && x + offsetX < 16 && y + offsetY >= 0
                        && y + offsetY < 16) {
114                     b.board[x + offsetX][y + offsetY].reveal();
115                 }
116             }
117         }
118     }
119 }

120 void show() {
121     if (isRevealed || b.gameOver) {
122         fill(255);
123         rect(x *50, y *50, 50, 50);
124         if (this.isBomb) {
125             fill(255, 0, 0);
126             circle(this.x *50 + 25, this.y *50 + 25, 40);
127         } else if (this.surroundingBombs > 0) {
128             fill(0);
129             textSize(20);
130             textAlign(CENTER);
131             text(surroundingBombs, x *50 + 25, y *50 + 25);
132         }
133     }
134 }
135 }
```

10.7 Bomben markieren

```
int fieldWidth = 50;
int fieldHeight = 50;
int bombCount = 20;
Board b;

void setup() {
   size(800, 800);
   b = new Board();
   b.generateBombs(bombCount);
   b.getBombCount();
}

void draw() {
   background(200);
   for (int i = 0; i < 16; i++) {
      fill(0);
      line(fieldWidth *i, 0, fieldHeight *i, height);
      line(0, fieldHeight *i, width, fieldHeight *i);
   }
   b.show();
}

void mouseReleased() {
   if (!b.board[mouseX / 50][mouseY / 50].isMarked && !b.gameOver &&
       mouseButton == LEFT) {
      b.board[mouseX / 50][mouseY / 50].reveal();
   }
   if (!b.board[mouseX / 50][mouseY / 50].isRevealed && !b.gameOver &&
       mouseButton == RIGHT) {
      b.board[mouseX / 50][mouseY / 50].isMarked = true;
   }
}



class Board {
   Field[][] board = new Field[16][16];
   boolean gameOver, isWon;

   Board() {
      for (int i = 0; i < board.length; i++) {
         for (int j = 0; j < board[0].length; j++) {
            board[i][j] = new Field(i, j);
         }
      }
   }

   void generateBombs(int num) {
      for (int i = 0; i < num; i++) {
         int ranX = int(random(16));
         int ranY = int(random(16));
         if (!board[ranX][ranY].isBomb) {
            board[ranX][ranY].isBomb = true;
         } else {
            i--;
         }
      }
   }

   void getBombCount() {
      for (int i = 0; i < board.length; i++) {
         for (int j = 0; j < board[0].length; j++) {
            int counter = 0;
```

```
53              for (int offsetX = -1; offsetX < 2; offsetX++) {
54                  for (int offsetY = -1; offsetY < 2; offsetY++) {
55                      if (offsetX == 0 && offsetY == 0) continue;
56                      if (i + offsetX >= 0 && i + offsetX < board.length && j +
                             offsetY >= 0 && j + offsetY < board.length && board[
                            i + offsetX][j + offsetY].isBomb) counter++;
57                  }
58              }
59              board[i][j].surroundingBombs = counter;
60          }
61      }
62  }

63  void isGameOver() {
64      if (gameOver) return;
65      for (int i = 0; i < board.length; i++) {
66          for (int j = 0; j < board[0].length; j++) {
67              if (!board[i][j].isRevealed && !board[i][j].isBomb) {
68                  gameOver = false;
69                  return;
70              }
71          }
72      }
73      gameOver = true;
74      isWon = true;
75  }

76  void show() {
77      isGameOver();
78      for (int i = 0; i < board.length; i++) {
79          for (int j = 0; j < board[0].length; j++) {
80              board[i][j].show();
81          }
82      }
83      if (gameOver) {
84          if (isWon) {
85              fill(0, 255, 0);
86              textSize(50);
87              textAlign(CENTER);
88              text("gewonnen", 400, 400);
89          } else {
90              fill(255, 0, 0);
91              textSize(50);
92              textAlign(CENTER);
93              text("verloren", 400, 400);
94          }
95      }
96  }
97  }

98  class Field {
99      int x, y, surroundingBombs;
100     boolean isBomb, isRevealed, isMarked;

101     Field(int x, int y) {
102         this.x = x;
103         this.y = y;
104     }

105     void reveal() {
106         if (this.isRevealed) return;
107         if (this.isBomb) {
108             b.gameOver = true;
109             b.isWon = false;
110         }
```

```
111         isRevealed = true;
112         if (this.surroundingBombs == 0 && !this.isBomb) {
113             for (int offsetX = -1; offsetX < 2; offsetX++) {
114                 for (int offsetY = -1; offsetY < 2; offsetY++) {
115                     if (offsetX == 0 && offsetY == 0) continue;
116                     if (x + offsetX >= 0 && x + offsetX < 16 && y + offsetY >= 0
                            && y + offsetY < 16) {
117                         b.board[x + offsetX][y + offsetY].reveal();
118                     }
119                 }
120             }
121         }
122     }

123     void show() {
124         if (isRevealed || b.gameOver) {
125             fill(255);
126             rect(x *50, y *50, 50, 50);
127             if (this.isBomb) {
128                 fill(255, 0, 0);
129                 circle(this.x *50 + 25, this.y *50 + 25, 40);
130             } else if (this.surroundingBombs > 0) {
131                 fill(0);
132                 textSize(20);
133                 textAlign(CENTER);
134                 text(surroundingBombs, x *50 + 25, y *50 + 25);
135             }
136         } else {
137             if (isMarked) {
138                 fill(0, 0, 255);
139                 rect(this.x *50 + 10, this.y *50 + 10, 30, 30);
140             }
141         }
142     }
143 }
```

509

Doodle-Jump

11.1 Spielverhalten

```
1  // Punktestand
2  int score = 0;
3  // Highscore
4  int highScore = 0;
5  // Framerate: regelt die Geschwindigkeit des Spiels
6  int framerate = 120;
7  // Anzahl an Plattformen im Spiel
8  int nPlatforms = 4;
9  // Array der Plattformen in Spiel
10 Platform[] platforms = new Platform[nPlatforms];
11 // Spielcharakter
12 Player player;

13 void setup() {
14    // Fenstergroeße (Breite x Hoehe)
15    size(600, 1000);
16    //Erstellt neue Platformen mit entsprechenden Abstaenden
17    initPlatforms();
18    //Erstellt den Spielcharakter mittig am unteren Fensterrand
19    player = new Player(width/2, height);
20    // setzt Framerate
21    frameRate(framerate);
22 }

23 void draw() {
24    // Schwarzer Hintergrund
25    background(0);
26     // Behandelt die Kollision mit einer Platform
27    playerCollides();
28    // Bewegt den Spielcharakter
29    updatePlayer();
30    // Bewegt alle Plattformen
31    updatePlatforms();
32    // Zeigt Score an
33    drawScore();
34    // Behandelt Berührung des unteren Bildschirmrandes durch den
          Spielcharakter
35    gameOver();
36 }

37 void updatePlayer() {
38    player.jump(); // Bewegt den Spielcharakter horizontal
39    player.move(); // Bewegt den Spielcharakter vertikal
40    player.drawPlayer(); // Bildet den Spielcharakter ab
41 }

42 void updatePlatforms() {
43    for (Platform p : platforms) { // Behandelt alle Plattformen
44       p.move(); // Bewegt Plattform
45       p.drawPlatform(); // Stellt Plattform dar
46    }
47 }
```

Auf der nächsten Seite geht's weiter...

```
48 void playerCollides() {
49    if (player.vy>0) { // Ueberpruefung ob der Spielcharakter am Fallen ist
50       for (Platform p : platforms) { // Behandelt alle Plattformen
51          // Hat Spieler die Höhe für eine Kollision mit betrachteter
                Plattform ?
52          if (inBetween(player.y+player.radius, p.y, p.rectH)) {
53             // Hat Spieler die X-Position für eine Kollision mit
                   betrachteter Plattform (rechts) ? || // Hat Spieler die X-
                   Position für eine Kollision mit betrachteter Plattform (
                   links) ?
54             if (inBetween(player.x+player.radius, p.x, p.rectW) ||
55             inBetween(player.x-player.radius, p.x, p.rectW)) {
56                player.bounce(); // Spieler springt von Plattform aus
57                score++; // Score wird incrementiert
58                // Framerate steigert sich bei jedem Sprung (hoehere
                      Schwierigkeit bei groesserem Score)
59                frameRate += 1;
60             }
61          }
62       }
63    }
64 }

65 void initPlatforms() {
66    // Berechnet den Abstand der Plattformen zueinander
67    int yOffsetPlatforms = height/(nPlatforms);
68    //Generiert alle Platformen mit entsprechendem Abstand
69    for (int i = 0; i<nPlatforms; i++) {
70       //Berechnet die Start-Y-Position anhand des Abstands und der
             Plattformen
71       int yOffset = yOffsetPlatforms*(i+1);
72       //Erstellt neue Plattform an entsprechender yPosition
73       platforms[i] = new Platform(yOffset);
74    }
75 }

76 void gameOver() { // Behandelt wenn ein Spielercharakter den unteren
       Bildschirmrand beruehrt
77    // Ueberprueft ob der Spieler den unteren Spielrand beruehrt
78    if (player.y>=height) {
79       // Falls Punktestand > Highscore wird neuer Highscore gesetzt
80       highScore = max(score, highScore);
81       // Punktestand wird zurueckgesetzt
82       score = 0;
83       // Framerate wird zurueck gesetzt
84       frameRate = 120;
85    }
86 }

87 void drawScore() { // Zeigt Punktestand an
88    fill(#00ff00);
89    textSize(20); // Textgroesse
90    text("Score: "+score, 10, 20); // Setzt Scoretext
91    text("High-Score:"+max(score, highScore), 10, 40); // Setzt
          Highscoretext
92 }

93 boolean inBetween(float val, float min, float limit) { // Prueft ob val
       zwischen min und limit liegt
94    return min <= val && val<=(min+limit);
95 }
```

11.2 Plattformen

Die Klasse `Plattform` stellt Methoden für das Verhalten der Plattformen bereit. Dazu gehören das nach-unten-sinken, und das Neuspawnen nach Verlassen des Fensters.

```
class Platform {
   int x; // X Koordinate
   int y; // Y Koordinate
   int vy = 2; // Platform Geschwindigkeit

   int rectW = 100; // Breite der Platform
   int rectH = 10; // Hoehe der Platform
   int margin = 20; //Abstand zum Bildschirmrand

   Platform(int yval) {
      x = int(random(margin, width-margin-rectW)); // Generiert zufaellige
          X-Position für die neue Plattform im erlaubten Bereich
      y = yval; // setzt Hoehe der Plattform auf uebergebenen Wert
   }

   void move() { //Bewegt Plattform
      y += vy; //Auf Hoehe wird Geschwindigkeit addiert
      if (y>height) { //Erreicht Plattform den unteren Fensterrand?
         resetPosition(); //Plattform wird auf Anfang zurückgesetzt
      }
   }

   void resetPosition() { //Plattform wird auf Anfang zurückgesetzt
      x = int(random(margin, width-margin-rectW)); // Generiert zufaellige
          X-Position für die Plattform im erlaubten Bereich
      y = 0; // Plattform wird auf oberen Fensterrand gesetzt
   }

   void drawPlatform() { //Zeichnet Plattform
      fill(#00ff00); //Plattform Farbe
      rect(x, y, rectW, rectH); // Ein Rechteck repraesentiert die
          Plattform
   }
}
```

11.3 Spielcharakter

In der Klasse `Player` beschreiben wir das Verhalten des Spielcharakters: Abspringen von Plattformen, schließliches Einsetzen der Schwerkraft und sein Aussehen.

```
class Player {

   float x; // X Koordinate
   float y; // Y Koordinate
   float vy = 0; // Spieler Fallgeschwindigkeit
   float vx = 4; // Spieler horizontale Geschwindigkeit
   float g = 0.2; //"Gravitation"
   float v = -10; // Initiale Geschwindigkeit nach Abprallen (negativ heißt
       nach oben(y))
   int radius = 10; // Spieler Größe

   Player(int x, int y) { // Konstruktor für neuen Spieler
      this.x = x; // setzt X Position
      this.y = y; // setzt Y Position
   }
```

Auf der nächsten Seite geht's weiter...

```
13      void move() { // Bestimmt aktuelle Position des Spielers
14          if (keyPressed) { // Ist eine Taste gedrückt
15              if (keyCode==LEFT) { // Ist die gedrückte Taste die linke
                    Pfeiltaste?
16                  x-= vx; // X-Position wird nach links verschoben
17              } else if (keyCode==RIGHT) { // Ist die gedrückte Taste die rechte
                    Pfeiltaste?
18                  x+= vx; // X-Position wird nach rechts verschoben
19              }
20          }
21          x = x \% width; // Rechter Ueberlauf des Spielfeld
22          if (x<=0) { // Erreicht der Spielcharakter den linken Spielrand (oder
                ist darunter) ?
23              x=width; // Spielcharakter wird auf rechten Rand gesetzt
24          }
25      }

26      void jump() { // Berechnet Spungposition des Spielcharakters
27          vy += g; // "Gravitation" beschleunigt die nach unten
28          y += vy; // Neue Y-Position ergibt sich durch Addieren der
                Geschwindigkeit
29          if (y>height-radius) { // Ueberpruefe, ob Spielcharakter unter den
                unteren Spielrand kommt
30              bounce(); // Neuer Sprung wird initialisiert
31          }
32      }

33      void bounce() { // Neuer Sprung wird initialisiert
34          vy = v; // Geschwindigkeit des Spielcharakters wird auf den
                Initialisierungswert gesetzt
35      }

36      void drawPlayer() { // Stellt den Spielcharakter dar
37          fill(#00ff00); // Spielcharakter Farbe
38          ellipse(x, y, radius*2, radius*2); // Spielcharakterkoerper
39      }
40  }
```

Jump'n'Run

Die Hauptklasse (v01):

```
1    Player ply = new Player(500, 200); // Erstellen Speilfigur mit
         Startposition
2    ArrayList<Block> blocks = new ArrayList<Block>();

3    void setup() {
4       size(1000, 600);
5       blocks.add(new Block(570, 100, 260, 50));
6       blocks.add(new Block(170, 200, 260, 50));
7       blocks.add(new Block(550, 300, 300, 50));
8       blocks.add(new Block(150, 400, 300, 50));
9       blocks.add(new Block(10, 500, width-20, 50)); //Boden
10      textAlign(CENTER);
11   }

12   void draw() {
13      background(50);
14      // Zeichnen der Bloecke
15      for (Block b : blocks) {
16         b.show();
17      }

18      //Aktualisierung Spielfigur
19      ply.update();

20      //Darstellung Anleitung
21      text("Pfeiltasten zum laufen und springen", width/2, 20);
22      text("R um Spieler zurückzusetzen", width/2, 40);
23   }
```

Die Klasse *Block* zum Erstellen von Hindernissen:

```
1   // Klasse zum Erzeugen von Bloecken

2   class Block {
3      // Attribute
4      PVector min; //Minimum
5      PVector max; //Maximum
6      float w; //Breite
7      float h; //Hoehe

8      // Konstruktor
9      Block(float x, float y, float w, float h) {
10        min = new PVector(x, y);
11        max = new PVector(x+w, y+h);
12        this.w = w;
13        this.h = h;
14     }

15     // Kollisionskontrolle
16     boolean playerCollision() {
17        if (abs(ply.min.y -(min.y+h/2)) > ply.h/2+h/2 || abs(ply.min.x -(
              min.x+w/2)) > ply.w/2+w/2) {
18           return false;
19        }
20        return true;
21     }

22     // Darstellung Block
23     void show() {
24        fill(100);
25        rect(min.x, min.y, w, h);
26     }
27  }
```

Die Klasse *Player* zum Erstellen der Spielfigur:

```
1   // Klasse zum Erzeugen der Spielfigur

2   class Player {
3      //Attribute
4      PVector min;
5      PVector max;
6      PVector lastPos;
7      PVector vel;
8      PVector acc;
9      float h = 80;
10     float w = 40;
11     float groundVel = 0;
12     float gravity = 1;


13     // Konstruktor
14     Player(float x, float y) {
15        min = new PVector(x, y);
16        max = new PVector(x+width, y+height);
17        lastPos = new PVector(x, y);
18        vel = new PVector(0, 0);
19        acc = new PVector(0, 0);
20     }

21     // Aktualisierung Attribute und Darstellung Spielfigur
22     // Trägheit: Objekte in Bewegung bleiben in Bewegung
23     void update() {
24        vel.x = min.x -lastPos.x -groundVel;
25        vel.y = min.y -lastPos.y;
26        vel.x *= 0.7;
27        vel.y *= 0.98;

28        accelerate();

29        float nextX = min.x + vel.x + acc.x + groundVel;
30        float nextY = min.y + vel.y + acc.y;

31        lastPos.x = min.x;
32        lastPos.y = min.y;

33        min.x = nextX;
34        min.y = nextY;

35        acc.x = 0;
36        acc.y = gravity;

37        solve();

38        //Darstellung Spielfigur
39        fill(220);
40        rect(min.x-w/2, min.y-h/2, w, h, 10, 10, 10, 10);
41     }

42     // Beschleunigung Spielfigur
43     void accelerate() {
44        //Run
45        if (keyIsDown(LEFT)) acc.x -= 2;
46        if (keyIsDown(RIGHT)) acc.x += 2;

47        //Jump
48        if (keyIsDown(UP) && vel.y==0 && onGround()) {
49           acc.y -= 20;
50        }
```

```
51          }

52          //Pruefung, ob Spielfigur auf einem der Bloecke/Boden sitzt
53          boolean onGround() {
54             for (Block b : blocks) {
55                if (onBlock(b)) {
56                   return true;
57                }
58             }
59             return false;
60          }

61          // Pruefung, ob Spielfigur auf Block sitzt
62          boolean onBlock(Block b) {
63             if (min.x+w/2-5 < b.min.x || min.x-w/2+5 > b.max.x) {
64                return false;
65             }
66             float yBelowPlayer = min.y+h/2+1;
67             if (yBelowPlayer > b.min.y && yBelowPlayer < b.max.y) {
68                return true;
69             }
70             return false;
71          }

72          //Loesung Kollision
73          //bei Kollision -> Versetzen Spielfigur auf naechste freie Position
74          void solve() {
75             for (Block b : blocks) {
76                if (b.playerCollision()) {
77                   PVector cToC = centerToCenter(ply, b);
78                   PVector fixedPos = min.copy();
79                   if (abs(cToC.x)-(w/2 + b.w/2) > abs(cToC.y)-(h/2 + b.h/2)) {
80                      if (cToC.x>0) {
81                         fixedPos.x = b.min.x -(ply.w/2);
82                      } else {
83                         fixedPos.x = b.max.x + (ply.w/2);
84                      }
85                   } else {
86                      if (cToC.y>0) {
87                         fixedPos.y = b.min.y -(ply.h/2);
88                      } else {
89                         fixedPos.y = b.max.y + (ply.h/2);
90                      }
91                   }
92                   min = fixedPos;
93                }
94                if (max.y > b.min.y) {
95                }
96             }
97          }

98          PVector centerToCenter(Player ply, Block b) {
99             float xVal = (b.min.x+b.w/2)- ply.min.x;
100            float yVal = (b.min.y+b.h/2)- ply.min.y;
101            return new PVector(xVal, yVal);
102         }
103      }
```

Die Klasse zum Verarbeiten von Key-Events:

```
1   //Klasse zur Verarbeitung von Key-Events

2   ArrayList<Integer> pressedKeys = new ArrayList<Integer>();

3   void keyPressed() {
4      if (!pressedKeys.contains(keyCode)) {
5         pressedKeys.add(keyCode);
6      }
7      if (key=='r') {
8         ply.min = new PVector(500,100);
9         ply.lastPos = ply.min.copy();
10     }
11  }

12  void keyReleased() {
13     if(pressedKeys.contains(keyCode)) {
14        pressedKeys.remove((Integer)keyCode);
15     }
16  }

17  boolean keyIsDown(int i) {
18     return pressedKeys.contains(i);
19  }

20  boolean keyIsDown(char c) {
21     return keyIsDown((int)Character.toUpperCase(c));
22  }
```

Breakout

13.1 Klasse Ball

```
  class Ball {
//Position des Balls
float xPos, yPos;
//Position des Balls im vorherigen Frame
float xPosOld, yPosOld;
//Vertikale und horizontale Geschwindigkeit des Balls
float vx, vy;
//Durchmesser, Radius des Balls
float d, r;
float speed = 3;

Ball(float xPos,float yPos,float d,float r,float vx,float vy) {
  this.xPos = xPos;
  this.yPos = yPos;
  this.d = d;
  this.r = r;
  this.vx = vx;
  this.vy = vy;
  xPosOld = xPos;
  yPosOld= yPos;
}

void draw(){
  fill(152, 255, 0);
  circle(xPos, yPos, 10);
 }
}
```

13.2 Klasse Brick

```
  class Brick {
  //Die Chance das ein powerup spawnen soll
 final float powerUpChance = 0.1f;
float posX;
float posY;
boolean active;
  //Anzahl der momentanen leben
int life;

Brick(float posX, float posY) {
  this(posX, posY, 1);
}

Brick(float posX, float posY, int life) {
  this.posX = posX;
  this.posY = posY;
  this.active = true;
  this.life = life;
}
// Zeichnet Block
void display() {
  if (this.active) {
    rect(posX, posY, blockWidth, blockHeight);
    fill(0, 0, 0);
    //text(""+life, posX, posY+blockHeight);
  }
}

/* Prüft ob der Ball die Box berührt
```

```
27    Alte Position wird benutzt, um Richtung des Balls zu bestimmen
28    */
29   int collide(float x, float y, float xOld, float yOld, float r) {
30     if (!active) return 0; // Wenn der Block nicht aktiv ist, kann er nicht
           berührt werden

31     // Wenn der Ball die Box berührt, wird die Richtung des Balls bestimmt
32     if (x + r > posX // rechts
33       && x -r < posX + blockWidth // links
34       && y + r > posY // unten
35       && y -r < posY + blockHeight // oben
36       ) {
37       if (xOld + r < posX // links
38         || xOld -r > posX + blockWidth) { // rechts
39         return 1;
40       } else {
41         return 2;
42       }
43     }

44     return 0;
45   }
46     //Bei jedem Treffer wird die Lebenanzahl um eins reduziert
47   void hit() {
48     life--;
49     score++;
50     if (life<=0) {
51       active = false;
52     }
53     if(random(1)<powerUpChance) {
54       powerups.add(new Powerup(posX,posY,Type.values()[(int)random(Type.
             values().length)]));
55     }
56   }
57 }
```

13.3 Klasse Paddle

```
1     class Paddle {
2   int paddleHeight, paddleWidth, paddleOffset;

3   Paddle(int paddleHeight, int paddleWidth, int paddleOffset) {
4     this.paddleHeight = paddleHeight;
5     this.paddleWidth = paddleWidth;
6     this.paddleOffset = paddleOffset;
7   }

8   void draw(int x,int y){
9     fill(255);
10    rect(x, y, paddle.paddleWidth, paddle.paddleHeight);
11  }
12 }
```

13.4 Klasse Powerup

```
1     class Powerup{
2   boolean pickedUp;
3   float inWorldPosX,inWorldPosY;
4   final int r = 10;
```

```
5   Type type;

6   Powerup(float inWorldPosX,float inWorldPosY,Type t){
7     this.inWorldPosX = inWorldPosX+blockWidth/2;
8     this.inWorldPosY = inWorldPosY+blockHeight/2;
9     pickedUp = false;
10    this.type = t;
11  }


12  void update(Paddle p,float paddleX,float paddleY){
13    inWorldPosY += 1;

14    if ((inWorldPosX + r > paddleX && inWorldPosX + r < paddleX + p.
          paddleWidth
15        || inWorldPosX -r > paddleX && inWorldPosX -r < paddleX + p.
              paddleWidth)
16        && inWorldPosY + r > paddleY) {
17        pickedUp = true;
18        // PADDLE_WIDTH,BALL_SPEED,BALL_SIZE,BALL_COUNT
19        switch(type){
20          case PADDLE_WIDTH:
21            p.paddleWidth += 10;
22            break;
23          case BALL_SPEED:
24            for(Ball b : balls){
25              b.speed += 1;
26            }
27            break;
28          case BALL_SIZE:
29            for(Ball b : balls){
30              b.r += 1;
31            }
32            break;
33          case BALL_COUNT:
34            balls.add(new Ball(paddleX+p.paddleWidth/2, 500, 10, 5, 0.5f, 1.3f
                  ));
35            break;
36        }
37     }
38  }

39  void draw(){
40    if(!pickedUp){
41      fill(203,91,240);
42      ellipse(inWorldPosX,inWorldPosY,r,r);
43    }
44  }
45  }
```

13.5 enum Type

```
1     //Die Verschiedenen Powerup-Typen
2   enum Type{
3     PADDLE_WIDTH,BALL_SPEED,BALL_SIZE,BALL_COUNT;
4    }
```

13.6 Hauptdatei

```
1     Paddle paddle;
2   ArrayList<Ball> balls = new ArrayList<Ball>();
```

```
3  float blockHeight, blockWidth;
4  ArrayList<Powerup> powerups = new ArrayList<Powerup>();
5  // Array aus Steinen, die getroffen werden müssen
6  Brick[] bricks;
7  // Farben für Steine
8  color[] colors;

9  int score = 0;


10 boolean running;

11 void setup() {
12   size(800, 600);
13   textSize(30);

14   bricks = flag();
15   colors = new color[10];
16   balls.add(new Ball(width/2, 500, 10, 5, 0.5f, 1.3f));

17   blockHeight = 25;
18   blockWidth = width/12;

19   paddle = new Paddle(20, 200, 50);

20   running = false;

21   // Farben für Regenbogenmuster in den Steinen
22   colors[0] = color(255, 0, 0);
23   colors[1] = color(255, 153, 0);
24   colors[2] = color(255, 255, 0);
25   colors[3] = color(152, 255, 0);
26   colors[4] = color(41, 255, 0);
27   colors[5] = color(0, 255, 150);
28   colors[6] = color(0, 150, 255);
29   colors[7] = color(0, 0, 255);
30   colors[8] = color(120, 0, 255);
31   colors[9] = color(190, 0, 255);
32 }

33 void draw() {
34   int paddleY = height -paddle.paddleOffset;
35   int paddleX;
36   if (pmouseX > width -paddle.paddleWidth/2) {
37     paddleX = width -paddle.paddleWidth;
38   } else if (pmouseX < paddle.paddleWidth/2) {
39     paddleX = 0;
40   } else {
41     paddleX = pmouseX -paddle.paddleWidth/2;
42   }

43   background(100);
44   paddle.draw(paddleX, paddleY);

45   for (int i = 0; i< bricks.length; i++) {
46     fill(colors[i/10]);
47     bricks[i].display();
48   }
49   for (Ball b : balls) {
50     b.draw();
51   }
52   for (int i=0;i<powerups.size();i++) {
53     Powerup p=powerups.get(i);
54     p.update(paddle, paddleX, paddleY);
55     if(p.pickedUp){
```

```
56      powerups.remove(p);
57    }
58    p.draw();
59  }

60  for (int j=0;j< balls.size() ; j++) {
61    Ball ball = balls.get(j);
62    //Abprallen von Wänden (links, rechts, oben)
63    if ((ball.xPos < ball.r && ball.vx < 0) || (ball.xPos > width -ball.r &&
          ball.vx > 0)) ball.vx *= -1;
64    if (ball.yPos < ball.r && ball.vy < 0) ball.vy *= -1;

65    //Abprallen von Schläger
66    if ((ball.xPos + ball.r > paddleX && ball.xPos + ball.r < paddleX +
          paddle.paddleWidth
67      || ball.xPos -ball.r > paddleX && ball.xPos -ball.r < paddleX + paddle.
            paddleWidth)
68      && ball.yPos + ball.r > paddleY) {
69      //Trefferpunkt ermittlen
70      float hitpoint = (ball.xPos -mouseX) / 100;
71      ball.vx += hitpoint;
72      //Winkel sollte nicht zu flach sein
73      if (ball.vx > 1.3) {
74        ball.vx = 1.3;
75      } else if (ball.vx < -1.3) {
76        ball.vx = -1.3;
77      }

78      if (ball.vy > 0) ball.vy *= -1; //Ball nur nach oben abprallen lassen
79    }
80    if (ball.yPos > 600) {
81      balls.remove(ball);
82      if (balls.size()<=0) {
83        running = false;
84        text("Game Over, press r to reset", 250, 300);
85        return;
86      }
87    }

88    //Abprallen von Blöcken
89    for (int i = 0; i < bricks.length; i++) {
90      int hit = bricks[i].collide(ball.xPos, ball.yPos, ball.xPosOld, ball.
            yPosOld, ball.r);
91      if (hit == 1) {
92        ball.vx *= -1;

93        bricks[i].hit();
94      }
95      if (hit == 2) {
96        ball.vy *= -1;
97        bricks[i].hit();
98      }
99    }
100 }




101 if (victory()) {
102   running = false;
103   text("Sieg", 400, 300);
104 }

105 if (running) {
106   for (Ball ball : balls) {
```

```
107       ball.xPosOld = ball.xPos;
108       ball.yPosOld = ball.yPos;

109       ball.xPos += ball.vx *ball.speed;
110       ball.yPos += ball.vy *ball.speed;
111     }
112     text("Score: " + score, 10, 30);
113   } else {
114     text("Click to start", width/2 -300, height/2 + 100);
115     text("Use mouse to move paddle", width/2 -100, height/2+50);
116     text("Use key 'r' to reset", width/2 -100, height/2+100);
117     text("Use keys 1-5 to change map", width/2 -100, height/2+150);
118   }
119 }

120 // Repräsentiert einen Block mit seiner Position und seinem Status



121 // R drücken um Spiel zu reseten
122 void keyPressed() {
123   if (key == 'r' || key == '1' || key == '2' || key == '3' || key == '4'||
          key == '5') {
124     if (key == '1') {
125       bricks = flag();
126     } else if (key == '2') {
127       bricks = rectangle();
128     } else if (key == '3') {
129       bricks = circle();
130     } else if (key == '4') {
131       bricks = triangle();
132     }else if (key == '5') {
133       bricks = rectangle2();
134     } else {
135       for (int i = 0; i< bricks.length; i++) {
136         bricks[i].active = true;
137       }
138     }
139     running = false;
140     balls.clear();
141     balls.add(new Ball(width/2, 500, 10, 5, 0.5f, 1.3f));

142     score =0;
143   }
144 }

145 // Klicken um das Spiel zu starten und den Ball zu aktivieren
146 void mouseClicked() {
147   running = true;
148 }

149 // Berechnung ob das Spiel vorbei ist bzw. der Spieler gewonnen hat
150 boolean victory() {
151   for (int i = 0; i < bricks.length; i++) {
152     if (bricks[i].active) return false;
153   }
154   return true;
155 }

156 Brick[] rectangle() {
157   Brick[] bricks = new Brick[100];
158   for (int i = 0; i < bricks.length; i++) {
159     bricks[i] = new Brick(width/12 + i%10*width/12, int(i/10)*25 + 50);
160   }
161   return bricks;
```

```
162 }

163 Brick[] rectangle2() {
164   Brick[] bricks = new Brick[100];
165   for (int i = 0; i < bricks.length; i++) {
166     bricks[i] = new Brick(width/12 + i%10*width/12, int(i/10)*25 + 50,(int)
          (((i/10f)*25f + 50f)/(height/3) *2f)+1);
167   }
168   return bricks;
169 }

170 Brick[] circle() {
171   Brick[] bricks = new Brick[100];
172   int centerX = width/2;
173   int centerY = height/2;
174   int radius = 250;
175   for (int i = 0; i < bricks.length; i++) {
176     float angle = i *TWO_PI / bricks.length;
177     float x = centerX + cos(angle) *radius;
178     float y = centerY + (sin(angle) *radius) / 2;
179     bricks[i] = new Brick(x, y);
180   }
181   return bricks;
182 }

183 Brick[] triangle() {
184   Brick[] bricks = new Brick[100];
185   int centerX = width/2;
186   int centerY = height/2 -200;
187   int oneThird = int(bricks.length / 3);
188   int twoThirds = int((bricks.length / 3) *2);
189   int step = 10;
190   for (int i = 0; i < oneThird; i += 2) { // Balken unten
191     bricks[i] = new Brick(centerX -i *step, centerY + 330);
192     bricks[i + 1] = new Brick(centerX + i *step, centerY + 330);
193   }
194   for (int i = oneThird; i < twoThirds; i++) { // linke Seite
195     bricks[i] = new Brick(centerX -(oneThird -i) *step, centerY -(oneThird -
          i) *step);
196   }
197   for (int i = twoThirds; i < bricks.length; i++) { // rechte Seite
198     bricks[i] = new Brick(centerX + (twoThirds -i) *step, centerY + (i -
          twoThirds) *step);
199   }
200   return bricks;
201 }

202 Brick[] flag() {
203   Brick[] bricks = new Brick[100];
204   for (int i = 0; i < bricks.length; i++) {
205     bricks[i] = new Brick(
206       width / 12 + (i % 10) *(width / 12),
207       (int(i / 10) *25 + 50) + 15 *sin((i % 10) / 1.5));
208   }
209   return bricks;
210 }
```

516

Schwarmverhalten II

14.1 Main

```
1  // This Code results in boids simulating swarm behaviour without a set
       direction

2  int boidAmount = 35;
3  Boid[] boids = new Boid[boidAmount];
4  float boidPerceptionRange = 150;
5  int maxOneDimensionalBoidSpeed = 5;
6  float maxSpeed = sqrt(maxOneDimensionalBoidSpeed*maxOneDimensionalBoidSpeed
       + maxOneDimensionalBoidSpeed+maxOneDimensionalBoidSpeed);
7  float cohesionWeight = 1.0;
8  float alignmentWeight = 1.0;
9  float separationWeight = 0.5;
10 float affectOldMovementBy = 0.2;
11 float margin = 100.0;
12 boolean drawFieldOfVision = false;

13 void setup() {
14    size(1800, 1000);
15    background(255);
16    // initalise boids
17    for (int i = 0; i < boidAmount; i++) {
18       boids[i] = randomValueBoid();
19    }
20 }

21 void draw() {
22    background(255);
23    for (int i = 0; i < boidAmount; i++) {
24       Boid myBoid = boids[i];
25       ArrayList<Boid> neighbours = new ArrayList<Boid>();
26       neighbours = getNeighbours(boids, i);
27       // deal with neighbours
28       if (neighbours.size() >= 1) {
29          calculateMovement(neighbours, myBoid);
30       }
31       myBoid.limitSpeed();
32       boidReactToEnvironment(i);
33       // show boids
34       boids[i].show();
35    }
36 }

37 // create a new Boid with random values inside the window. Make sure it is
       not too fast
38 Boid randomValueBoid() {
39    return new Boid(random(0, width), random(0, height), random(-
          maxOneDimensionalBoidSpeed, maxOneDimensionalBoidSpeed), random(-
          maxOneDimensionalBoidSpeed, maxOneDimensionalBoidSpeed));
40 }

41 // moves the boid and makes sure it stays in the window
42 void boidReactToEnvironment(int i) {
43    respawnLostBoids(i);
44    boids[i].turnIfOutOfBounds(margin, width-margin, margin, height-margin);
45    boids[i].move();
46 }
```

Auf der nächsten Seite geht's weiter...

```
47 // if a boid ended up out of frame, respawn it in the window
48 void respawnLostBoids(int index) {
49    if (boids[index].boidIsLost(0, width, 0, height)) {
50       boids[index] = randomValueBoid();
51    }
52 }

53 // calculates the future movement of the boid
54 void calculateMovement(ArrayList<Boid> neighbours, Boid myBoid) {
55    PVector alignment, separation, cohesion;
56    separation = calculateSeparation(neighbours, myBoid);
57    // cohesion
58    cohesion = calculateCohesion(neighbours, myBoid);
59    // alignment
60    alignment = calculateAlignment(neighbours, myBoid);
61    // weight the input of the three rules
62    myBoid.weighVectors(separation, cohesion, alignment);
63 }

64 // rule "alignment": move in the average dicrection your neighbours are
       moving. This should result in flock members usually moving in similar
       directions
65 PVector calculateAlignment(ArrayList<Boid> neighbours, Boid currentBoid) {
66    PVector sum = new PVector(0, 0);
67    for (int n = 0; n < neighbours.size(); n++) {
68       sum = vAddition(neighbours.get(n).getMovement(), sum);
69    }
70    PVector avg = skalarMultiplication(1.0/neighbours.size(), sum);
71    return vSubtraction(avg, currentBoid.getMovement());
72 }

73 // rule "separation": move away from your neighbours. If they are close,
       move more to avoid them. This should result in a healthy distance
       between members of the flock.
74 PVector calculateSeparation(ArrayList<Boid> neighbours, Boid currentBoid) {
75    PVector sum = new PVector(0, 0);
76    for (int n = 0; n < neighbours.size(); n++) {
77       // to make sure that the movement away from a neighbour is stronger
             the closer it is, use the inverseDistance: calculate the distance
             between neighbour and the end of your range of vision.
78       PVector inverseDistance = calculateInverseDistanceVector(currentBoid,
             neighbours.get(n));
79       // combine your vectors
80       sum = vAddition(sum, inverseDistance);
81    }
82    // divide through number of neighbours for average
83    return skalarMultiplication(1.0/neighbours.size(), sum);
84 }
```

Auf der nächsten Seite geht's weiter...

```
85  // rule "cohesion": move towards the center of your neighbours. This should
        in members usually staying with the flock
86  PVector calculateCohesion(ArrayList<Boid> neighbours, Boid currentBoid) {
87     PVector sum = new PVector(0, 0);
88     // get the center of the Neighbours by calculationg an average of their
           positions
89     for (int n = 0; n < neighbours.size(); n++) {
90        sum = vAddition(neighbours.get(n).getPosition(), sum);
91     }
92     PVector centerOfNeighboursPositions = skalarMultiplication(1.0/
           neighbours.size(), sum);
93     // calculate a vector describing the path from the current boid to the
           neighbours' center
94     return vSubtraction(centerOfNeighboursPositions, currentBoid.position);
95  }

96  // finds all boids in the percetion range of boids[boidIndex] and returns
       them as ArrayList
97  ArrayList<Boid> getNeighbours(Boid[] boids, int boidIndex) {
98     ArrayList<Boid> neighbours = new ArrayList<Boid>();
99     for (int j = 0; j < boidAmount; j++) {
100       // don't treat a boid as its own neighbour!
101       if (j != boidIndex) {
102          // use local variables for easier access
103          Boid otherBoid = boids[j];
104          PVector myPos = boids[boidIndex].getPosition();
105          PVector otherPos = otherBoid.getPosition();

106          // in perception range?
107          if (dist(myPos.x, myPos.y, otherPos.x, otherPos.y) <
                boidPerceptionRange) {
108             neighbours.add(otherBoid);
109          }
110       }
111    }
112    return neighbours;
113 }

114 PVector calculateInverseDistanceVector(Boid myBoid, Boid otherBoid) {
115    // use local variables for easier access
116    PVector myPos = myBoid.getPosition();
117    PVector otherPos = otherBoid.getPosition();
118    // get distance between boids
119    float dist = dist(myPos.x, myPos.y, otherPos.x, otherPos.y);
120    float partsOfRange = (boidPerceptionRange / dist);
121    // get vector to get from otherPos to myPos -> if applied to myPos,
           moves it away from otherPos
122    PVector fromMyToOtherVector = vSubtraction(myPos, otherPos);
123    PVector prod = skalarMultiplication(partsOfRange -1, fromMyToOtherVector
           );
124    return prod;
125 }
```

14.2 Boid

```
1  class Boid {
2     PVector position;
3     PVector movement;

4     Boid(float positionX, float positionY, float movementX, float movementY)
          {
5        this.position = new PVector(positionX, positionY);
6        this.movement = new PVector(movementX, movementY);
7     }

8     void limitSpeed(){
9        float speed = dist(this.movement.x, this.movement.y, 0, 0);
10       // ensure that maxSpeed is not exceeded.
11       // keep direction
12       if (speed > maxSpeed) {
13          this.movement = skalarMultiplication(maxSpeed,
                skalarMultiplication(1/speed, this.movement));
14       }
15    }

16    void move() {
17       // change Boids position
18       this.position = vAddition(this.position, this.movement);
19    }


20    void show() {
21       // Seeing the field of vision of the boids can be interesting when
             experimenting
22       if (drawFieldOfVision) {
23          noFill();
24          circle(this.position.x, this.position.y, 2 *boidPerceptionRange);
25          fill(0);
26       }
27       // show boid as circle and orientation as line
28       circle(this.position.x, this.position.y, 5);
29       line(this.position.x, this.position.y, this.position.x + this.
             movement.x, this.position.y + this.movement.y);
30    }

31    // apply the weights to control the input of the rules on the boids path
32    void weighVectors(PVector separation, PVector cohesion, PVector
          alignment) {
33       // one vector represing each of the three rules and apply weights
34       separation = skalarMultiplication(separationWeight, separation);
35       cohesion = skalarMultiplication(cohesionWeight, cohesion);
36       alignment = skalarMultiplication(alignmentWeight, alignment);
37       // combine rules
38       PVector rulesCombined = vAddition(separation, vAddition(cohesion,
             alignment));
39       // combine the weightened new and old movement for more fluid
             behaviour
40       PVector newMovementPart = skalarMultiplication(affectOldMovementBy,
             rulesCombined);
41       this.movement = vAddition(this.movement, newMovementPart);
42    }
```

Auf der nächsten Seite geht's weiter...

```
43      // inverse direction of movement if position of boid is out of the area
            definied via arguments
44      void turnIfOutOfBounds(float minX, float maxX, float minY, float maxY) {
45          float turnBy = 0.5;
46          if (this.position.x < minX) {
47              this.movement.x = this.movement.x + turnBy;
48          }
49          if (this.position.x > maxX) {
50              this.movement.x = this.movement.x -turnBy;
51          }
52          if (this.position.y < minY ) {
53              this.movement.y = this.movement.y + turnBy;
54          }
55          if (this.position.y > maxY ) {
56              this.movement.y = this.movement.y -turnBy;
57          }
58      }

59      // check if a boid is outside the window
60      boolean boidIsLost(float minX, float maxX, float minY, float maxY) {
61          if (this.position.x < minX || this.position.x > maxX) {
62              return true;
63          } else if (this.position.y < minY || this.position.y > maxY) {
64              return true;
65          } else {
66              return false;
67          }
68      }

69      void setPosition(float x, float y) {
70          this.position = new PVector(x, y);
71      }

72      PVector getPosition() {
73          return this.position;
74      }

75      PVector getMovement() {
76          return this.movement;
77      }
78  }

79  PVector vAddition(PVector a, PVector b) {
80      return new PVector(a.x + b.x, a.y + b.y);
81  }

82  PVector vSubtraction(PVector a, PVector b) {
83      return new PVector(a.x -b.x, a.y -b.y);
84  }


85  PVector skalarMultiplication(float skalar, PVector a) {
86      return new PVector( skalar *a.x, skalar *a.y);
87  }
```