Aufgabenblatt 3 - Events und Funktionen

325
3 Z.

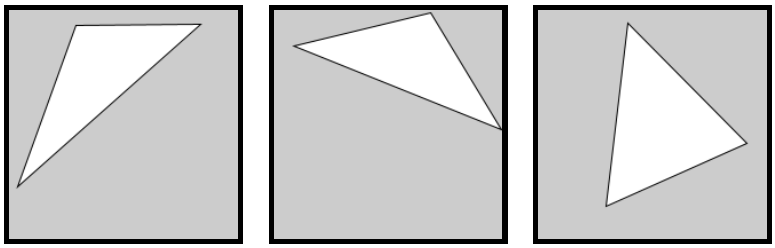
Aufgabe 1 – Zufallsdreieck

Erzeugen Sie ein zufälliges Dreieck. Füllen Sie dafür ein Array `float[] arr;` mit 6 Zufallswerten, und greifen Sie dann auf diese in dem Aufruf der `triangle(x1, y1, x2, y2, x3, y3)`-Funktion zu.

Hier wird ein Array aus „Glückszahlen“ erzeugt.

```
1 float[] arr = {random(0, 200), random(0, 200)};
2
2 println("Die Glückszahlen von Heute sind " + arr[0] + " und " + arr[1] + "!
      ");
```

So könnten Dreiecke aussehen, die von Ihrem Programm generiert wurden:



321
4 Z.

Aufgabe 2 – Array-Ausgabe

Deklarieren Sie ein Array `int[] arr;` und geben Sie mithilfe einer Schleife die Werte des Arrays in der Konsole aus.

So sieht der Zugriff auf ein Array mithilfe einer Variable aus. Wo haben Sie `i` schon einmal gesehen?

```
1 int[] arr = {1, 42, 2, 9, -5};
2 int i = 1;
3 println(arr[i]); // Ausgabe: 42
4 i = i + 1;
5 println(arr[i]); // Ausgabe: 2
```

Hinweis: *Der Ausdruck* `i < arr.length` *mit einem positiven* `i` *ist wahr, wenn* `i` *als Index für einen Zugriff auf* `arr` *verwendet werden kann.*

322

(10 Z.)
20 Z.

Aufgabe 3 – Maximum

Schreiben Sie eine Funktion `int maxOfThree(int a, int b, int c){ ... }`, welche aus drei Werten den höchsten ermittelt und zurückgibt. Testen Sie ihre Funktion mit einem Aufruf durch von Ihnen gewählten Variablen und mit einer Ausgabe des Ergebnisses in der Konsole.

So sieht zum Beispiel eine Funktion aus, die die Summe zweier Werte ermittelt und zurückgibt. Dieses Ergebniss wird dann in setup auf der Konsole ausgegeben.

```
1 void setup() {
2    println(sum(11, 31));
3 }

4 int sum(int x, int y) {
5    int result = x + y;
6    return result;
7 }
```

Hinweis: *Beginnen Sie bei Bedarf mit einer Funktion* `int maxOfTwo(int a, int b)`*, welche das Maximum von nur zwei Werten berechnet.*

323

7 Z.

Aufgabe 4 – Zeichenprogramm II

Modifizieren Sie Ihre Lösung zur Aufgabe *Zeichenprogramm* von Tag 2: Nun soll nur ein Kreis gezeichnet werden, wenn mit der Maus geklickt wurde.

Hier wird bei Mausklick die Position der Maus ausgegeben. Die `draw`-Funktion muss existieren, damit Processing die Ausführung des Programms nicht vorzeitig mit Standbild beendet.

```
1 void setup(){
2    size(400, 400);
3 }

4 void draw() { } // hier muss nichts rein.

5 void mousePressed() {
6    println("Mausposition: (" + mouseX + ", " + mouseY + ")");
7 }
```

324

11 Z.

Aufgabe 5 – Scrollen

Schreiben Sie ein Programm, in welchem ein Kreis mit dem Mausrad nach unten und oben bewegt werden kann. Speichern Sie dazu die vertikale Position des Kreises in einer Variable `int y;`. Verwenden Sie dann das Event `void mouseWheel(MouseEvent event){ ... }` und greifen Sie auf `event.getCount()` zu.

Hier wird pro Frame ein Wert in der Konsole ausgegeben, welcher die Richtung und Intensität des Scrollen als Zahlenwert ausgibt.

```
1 void draw() { }

2 void mouseWheel(MouseEvent event) {
3    print(event.getCount() + " ");
4 }
```

Hinweis: *Events wie das Scrollen werden von Processing eventuell nur erkannt, wenn das Fenster des Zeichenbereiches fokussiert ist.*

301

23 Z.
31 Z.
50 Z.

Aufgabe 6 – Einstiegsaufgabe 1: Pferderennen

Wir wollen mithilfe des `keyReleased()`-Events ein Spiel entwickeln, bei welchem die Spieler:innen so schnell wie möglich zwei Tasten auf der Tastatur drücken müssen, um vor dem/der anderen ins Ziel zu kommen.

6.1 Darstellung

Zunächst wollen wir die Pferde auf ihre Bahn platzieren - dafür können Sie Kreise verwenden. Natürlich sind Sie eingeladen, das Programm später mit Bildern realistischer darzustellen. Mehr dazu unter `PImage` in der *Processing Reference*.

Platzieren Sie die beiden Pferde übereinander in einer länglichen Zeichenfläche. Erstellen Sie zusätzlich die Funktion `void drawFinishLine()`, welche die wie im Beispiel angegeben Ziellinie zeichnet - können Sie das berühmte schwarz-weiße Muster erzeugen? Vergessen Sie nicht, die Funktion `drawFinishLine()` aufzurufen, damit diese auch gezeichnet wird.

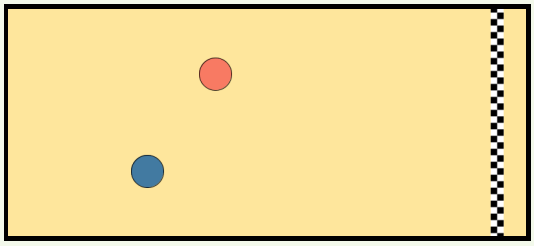
Hinweis: `strokeCap(PROJECT)` *lässt Punkte und andere Umrandungen eckig statt rund wirken.*

6.2 Bewegung

Legen Sie die `x`-Positionen der beiden „Pferde“ in Variablen fest, und entscheiden Sie in dem `keyReleased()`-Event wie folgt:

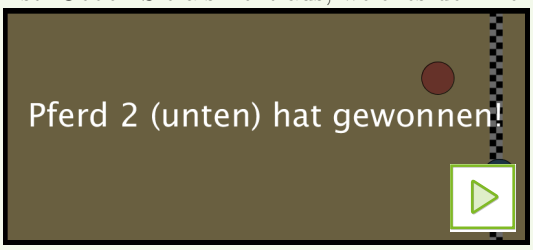
- Gilt `key == 'a'` oder `key == 'd'`, erhöhe `x` von Pferd 1 um 5.
- Gilt `key == CODED`, sowie `keyCode == LEFT` oder `keyCode == RIGHT`, erhöhe `x` von Pferd 2 um 5.

Falls noch nicht getan, müssen Sie dazu auch ihre Zeichenfunktionen in die `draw()`-Funktion platzieren. Dann können Sie das Spiel gegen sich selbst spielen.



6.3 Gewinner:in

Um dem Streit aus dem Weg zu gehen, wer denn jetzt eigentlich gewonnen hat, sollen Sie als letzten Schritt eine Anzeige entwickeln, welche erscheint sobald eines der Pferde über die Ziellinie gekommen ist. Geben Sie als Text aus, welches der Pferde gewonnen hat.



Hinweis: *Mithilfe von* `noLoop();` *können sie verhindern, dass die Pferde weiter bewegt werden können.*

302

25 Z.




Aufgabe 7 – Einstiegsaufgabe 2: Fantasy-Namensgenerator

Oft braucht man für Fantasy-Spiele, sowohl virtuell als auch real, einen passenden Namen. Schreiben Sie einen Algorithmus, der Ihnen einen zufällig generierten Namen ausgibt!

Verwenden Sie dazu vier `String[]`-Arrays:

1. Vornamen (Weiblich)
2. Vornamen (Männlich)
3. Zweitname / Bezeichnung
4. Nachname

Lassen Sie den Zufall über das Geschlecht entscheiden, und nehmen Sie dann aus jedem Array ein zufälligen Wert und zeigen Sie ihn an. Lassen Sie einen neuen Namen generieren, wenn eine Taste gedrückt wird.
Schreiben Sie dazu die Funktion `String getName()`, welche bei jedem Aufruf einen neuen Namen ausgibt.

Hinweis: *Die Funktion* `random(max)` *gibt Ihnen einen Wert von* `0` *bis* `max-1` *zurück. Gegebenenfalls müssen Sie abrunden.*

Zudem einige Hinweise zu Arrays:

1. Mit `myArray.length` erhalten Sie die Länge eines gegebenen Arrays.
2. Mit `myIntegers[] = new int[]{1, 7, 9, 2}` erhalten Sie ein Array, welches die Werte 1, 7, 9, 2 speichert. Es gilt dann z.B. `myIntegers[2] == 9`.

Olaf der Schläfrige von und zu Kunz

Knort die Programmierende von Drenor

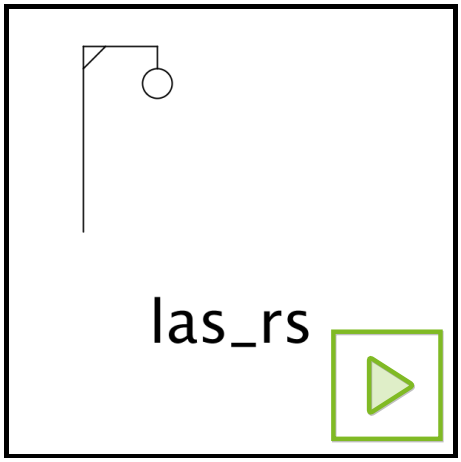
314

42 Z.
49 Z.
68 Z.
103

Aufgabe 8 – Galgenmännchen II

Fügen Sie ihrem Galgenmännchen-Spiel die Interaktion hinzu.

Hinweis: *Um diese Aufgabe zu bewältigen, benötigen Sie das Ergebnis der Galgenmännchen-Aufgabe von Tag 1 (103).*



8.1 Arrays über Arrays

Setzen Sie dazu zunächst Ihr derzeitigen Code in eine `draw()`-Funktion, und definieren Sie zudem die `setup()`-Funktion wie gewohnt.

Initialisieren Sie dann die folgenden **globalen** Variablen und Arrays:

- `String[] words`, welches eine Auswahl von Wörtern enthält, aus welchem in jedem Durchlauf eines zufällig gewählt werden soll.
- `char[] guesses`, welches alle getätigten Tipps speichert (aber keine Duplikate enthalten soll).
- `String word`, welches das zu erratende Wort enthalten soll.
- `char[] chars`, welches das zu erratende Wort als ein Array von Buchstaben speichert.
- `char[] display`, welches eine Reihe von _-Charakteren enthalten soll, und zwar genau so viele, wie das zu ratende Wort lang ist.
- `int guessCount`, welches die Anzahl der Tipps speichert (aber wieder keine Duplikate zählen soll).

8.2 Ist der Charakter im Array?

Wir benötigen eine Funktion, welche prüft, ob ein gewisser Charakter in unserem Array enthalten ist.
Deklarieren Sie die Funktion `boolean isInArray(char c, char[] array)`. Sie soll `true` ausgeben, wenn der Charakter im Array enthalten ist, ansonsten `false`.

Auf der nächsten Seite geht diese Aufgabe weiter...

8.3 Action!

Schreiben Sie nun das `keyReleased()`-Event:

- Jede Eingabe soll, falls sie noch nicht getätigt wurde, in das Array `guesses` gespeichert werden. Verwenden Sie dazu einen Aufruf ihrer `isInArray()`-Funktion.
- Falls dieser Buchstabe im Wort enthalten ist, sollen die entsprechenden Felder angezeigt werden.
- Falls dieser Buchstabe nicht enthalten ist, soll ein weiteres Stück des Galgenmännchen gezeichnet werden.

Hinweis: *Mit* `myString.toLowerCase()` *können Sie einen String in Kleinbuchstaben schreiben.*
Hinweis: *Mit* `myString.toCharArray()` *erhalten Sie sie aus einem String ein Array von Buchstaben.*
Hinweis: *Mit* `String.valueOf(myCharArray)` *erhalten Sie aus einem Array von Buchstaben einen String.*

Falls Sie möchten, können Sie das Programm erweitern und z.B. überprüfen, ob der/die Spieler:in gewonnen oder verloren hat.

313
71 Z.

Aufgabe 9 – Lights Out

Programmieren Sie das Spiel *Lights Out*:

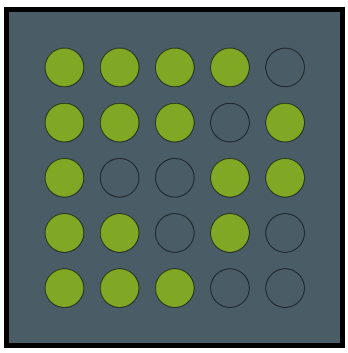
Auf einem 5x5 Felder großen Spielfeld befinden sich Lampen, die zu Beginn alle angeschaltet sind. Klickt der/die Spieler:in auf eine der Lampen, so wechselt sich deren Zustand (`an` $\Longleftrightarrow$ `aus`), aber gleichzeitig auch der Zustand aller direkt benachbarten Lampen (links, rechts, oben, unten).
Ziel des Spiels ist es, alle Lampen auszuschalten.

Ein 2D-Array vom Typ boolean (`boolean[][]`) könnte Ihnen hier behilflich sein.
Dies ist ein Array aus Arrays - welches Sie auch als solches initialisieren müssen (also mit einer `for`-Schleife über das erstellte Array iterieren und jede Zelle mit einem neuen `boolean[]`-Array belegen).

Schreiben Sie die Funktionen `boolean hasWon()` (`true`, falls alle Lampen aus sind) und `void drawGameOver()`, welches ein Overlay zeichnet, für den Fall, dass das Spiel vorbei ist.

Implementieren Sie zudem, dass bei einem Druck der Taste `r` das Spielfeld zurückgesetzt wird.

Hinweis: *Das Spiel zu gewinnen ist schwerer, als es vielleicht zunächst aussieht. Falls Sie nicht weiterkommen, können Sie die sog. „Chasing Lights“-Strategie recherchieren.*



310

54 Z.

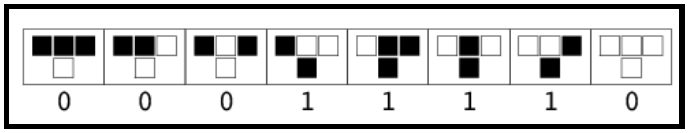
Aufgabe 10 – Zellulärer Automat

Ähnlich wie beim *Game of Life* entwickeln sich hier Zellen abhängig von deren Nachbarn über Zeit. Die Zellen werden durch ein Gatter aus schwarzen („lebend“) und weißen („tot“) Quadraten dargestellt. Nach jeder Zeiteinheit werden, abhängig von bestimmten Regeln, neue Generationen gebildet.

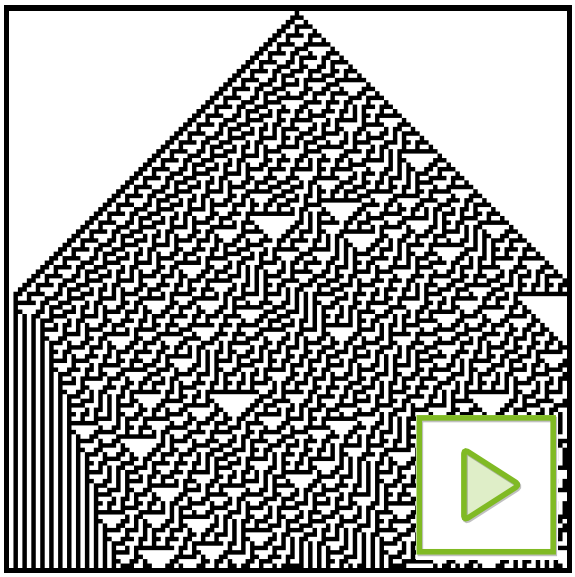
Zellen werden in einem Tabellen-artigen Konstrukt gespeichert, wobei jede Generation eine Zeile bildet. Neue Generationen werden unten angehängt und anhand der direkt über ihnen liegende Zeile gebildet.

Die „Generation 0“ wird häufig mit nur einer einzelnen Zelle erstellt.

Je nach angewandtem Regelset für die Neubildung von Zellen entstehen unterschiedliche Figuren. Hier eines der möglichen Regelsets:



Mithilfe dieses Regelsets und einer Startzelle in der Mitte bildet sich dieses Konstrukt:



Erstellen Sie einen wie oben beschriebenen zellulären Automaten und definieren Sie optional Ihr eigenes Regelset.

Schreiben Sie eine Funktion `int[] makeNewGeneration()`, welche mithilfe des Arrays ein neues Array erstellt und, nachdem es mithilfe der Regeln befüllt wurde, dieses ausgibt. Am Ausführungsort soll die Ausgabe dann das Generation-Array überschreiben.

Hinweis: *Wenn Sie die* `background(i)`*-Funktion nur einmal zu Beginn aufrufen, müssen eine vorherige Generationen nach dem einmaligen Malen nicht länger speichern, und können so ein Array immer wieder verwenden. Vergessen Sie dann nicht, eine Variable zu verwenden, welche Ihnen die derzeitige* `y`*-Position im Zeichenbrett festlegt.*

305
16 Z.
22 Z.

Aufgabe 11 – Barcode

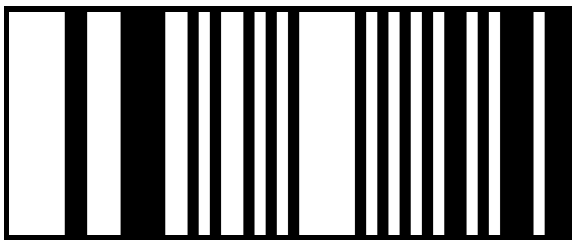
Erstellen Sie ein Barcode-Array mit 100 Zahlen, die zufällig mit 1 oder 0 belegt sind. Zeichne dieses Array, indem Sie für jede 0 ein weißes und für jede 1 ein schwarzes Rechteck nebeneinander malen. Die Rechtecke sollen vom oberen bis zum unteren Bildschirmrand reichen.

Verwenden Sie zur Speicherung des Barcodes ein `boolean[]`-Array, das Sie am Anfang mit Zufallswerten befüllen.

11.1 Erweiterung

Erweitern Sie Ihr Programm so, dass Benutzende jetzt mit einem Klick auf eines der Rechtecke die Farbe ändern können (schwarz $\iff$ weiß).

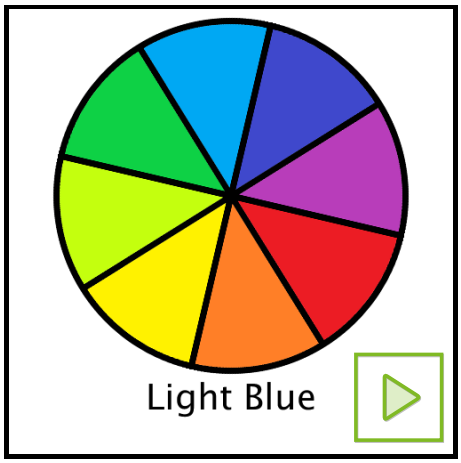
Berechnen Sie in `void mousePressed()` anhand von `mouseX`, welcher Streifen angewählt wurde, und ändern Sie dann den entsprechenden Wert.



307
29 Z.

Aufgabe 12 – Glücksrad

Programmieren Sie ein Glücksrad.



Laden Sie sich dazu das folgende Bild herunter (oder wählen Sie ein eigenes) und importieren Sie es dann mithilfe von `PImage` in Ihr Programm.

Hier klicken, um das Bild herunterzuladen.

Legen Sie Drehgeschwindigkeit und Reibung in Variablen fest, und erhören Sie die Drehgeschwindigkeit, wenn `keyPressed()` aufgerufen wird (diese Funktion wird wiederholt aufgerufen, wenn eine Taste gedrückt gehalten wird!).

Verwenden Sie `rotate(r)`, um die Drehung zu simulieren. Optional können Sie ermitteln, auf welcher Farbe das Glücksrad bei Halt steht, und diese Farbe anzeigen.

Hinweis: *Arbeiten Sie mit kleinen Werten! Ein Wert von* `0.05` *ist in Radianten ein angemessener Schritt für die Erhöhung der Geschwindigkeiten.*
Hinweis: *Vorsicht - das Glücksrad kann sehr schnell werden und unangenehmes Flackern verursachen.*

306

73 Z.

Aufgabe 13 – Statistik ohne Excel

Erstellen Sie eine fiktive Liste von Klausurergebnissen als Array mit 200 Zufallszahlen zwischen 0 und 100. Lassen Sie aus dieser Verteilung eine Statistik mit den folgenden Angaben errechnen:

1. Durchfallquote (Prozentsatz der Ergebnisse mit der Note 5)
2. Notenverteilung als Anzahl der Bewertungen „sehr gut“, „gut“, „befriedigend“, „ausreichend“ und „mangelhaft“ entsprechend den Vorgaben der THM - diese finden Sie in der allgemeinen Bachelorprüfungsordnung (hier klicken)
3. Durchschnittsnote derjenigen, die die Klausur bestanden haben

Durch das Drücken einer Taste sollen komplett neue Noten ausgewürfelt werden.

Verwenden Sie Funktionen, um die einzelnen Rechenschritte gut aufzuteilen, und um doppelten Code zu vermeiden.

Die Note einer bestimmten Prozentzahl können Sie mit der folgenden Funktion berechnen:

$$f(x) = 4 - (x - 50)/\frac{50}{3}$$

Gilt $f(x) > 4$, so soll das Ergebnis stattdessen 5 („mangelhaft“) sein.

Students: 200

Average passing grade: 2.5

Average passing percentage: 74

sehr gut: 11 befriedigend: 31

gut: 34 ausreichend: 15

mangelhaft: 109

317

71 Z.

Aufgabe 14 – Tic Tac Toe

Schreiben Sie ein Processing-Programm für das Spiel *Tic Tac Toe*! Auf den ersten Blick wirkt die Aufgabe eventuell ziemlich anspruchsvoll, aber wenn man das Problem in kleinere Teilprobleme aufteilt ist es durchaus machbar:

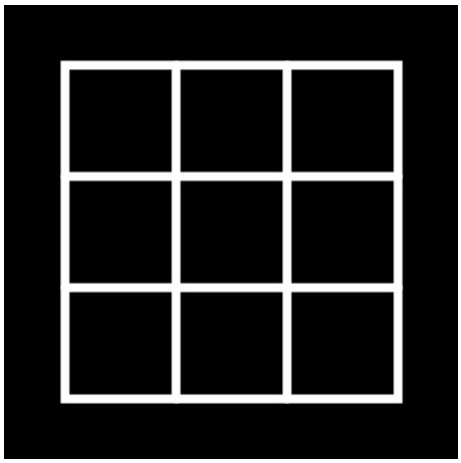
1. Zeichnen Sie als erstes das Spielfeld von *Tic Tac Toe*: Ein großes Quadrat, in dem sich 3 x 3 kleinere Quadrate befinden.

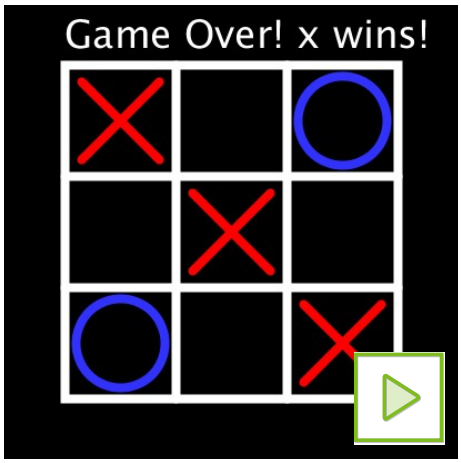
   Hinweis: *Muss für jedes Feld ein eigenes Quadrat gezeichnet werden?*

2. Initialisieren Sie ein `char[]`-Array, welches die 9 verschiedenen Felder speichert. Schreiben Sie dann eine zweidimensionale (verschachtelte) Schleife, welche mithilfe des Arrays Kreuze und Kreise einzeichnen kann.

3. Implementieren Sie das Event `void mouseClicked()`, welches neue Kreuze oder Kreise in das Array einfügt, wenn auf das entsprechende Feld geklickt wird.
   Hinweis: *Stellen Sie sicher, dass das Feld nicht schon belegt ist.*

4. Zuletzt soll das Programm selbstständig erkennen, ob ein:e Spieler:in gewonnen hat. Ist dies der Fall, so soll ein Text im Fenster ausgegeben werden und es soll unmöglich sein, weitere Züge zu machen.



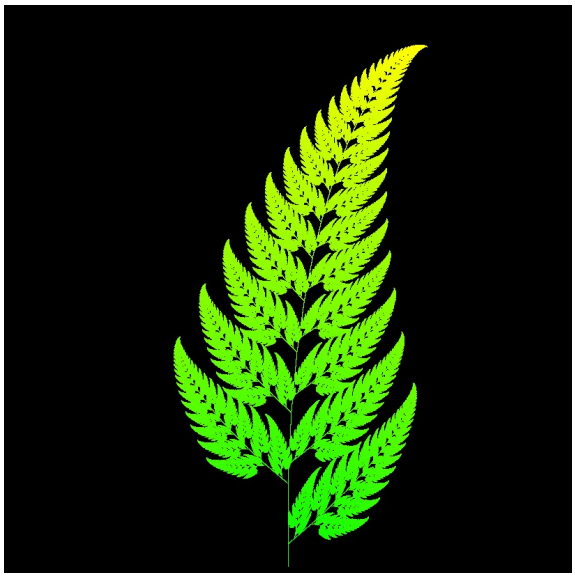


316

60 Z.

Aufgabe 15 – Barnsley Farn

Der *Barnsley Farn* ist ein Fraktal des britischen Mathematikers Michael Barnsley, mit dessen Hilfe sich aus einzelnen, (zufälligen) Punkten ein Farn erstellen lässt, der dem *schwarzstieligen Streifenfarn* nachempfunden ist.



Dabei wird vom Punkt (0,0) ausgehend jeder Punkt aus dem vorhergegangenen Punkt berechnet:

	a	b	c	d	e	f	p
f_1	0	0	0	0.16	0	0	0.01
f_2	0.85	0.04	-0.04	0.85	0	1.60	0.85
f_3	0.20	-0.26	0.23	0.22	0	1.60	0.07
f_4	-0.15	0.28	0.26	0.24	0	0.44	0.07

Die Arrays a, b, c, d, e und f bestimmen die Abbildung, und p die Wahrscheinlichkeit, ob diese Abbildung ausgewählt wird.
Die Abbildung sieht wie folgt aus:

$$f(x,y) = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} e \\ f \end{pmatrix}$$

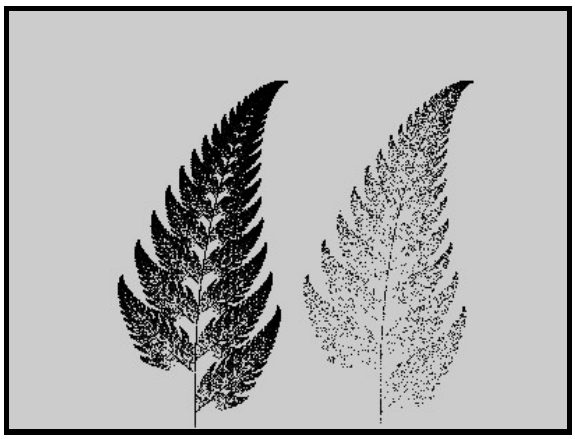
...oder vereinfacht:

$$x_{n+1} = a * x_n + b * y_n + e$$
$$y_{n+1} = c * x_n + d * y_n + f$$

Auf der nächsten Seite geht diese Aufgabe weiter...

1. Beginnen Sie damit, die Variablen sinnvoll zu speichern (ohne jede der 28 Variablen einzeln zu speichern!).
2. Wie kann die zufällige Auswahl der Funktion simuliert werden (f_1 mit einer Wahrscheinlichkeit von 1%, f_2 mit 85%, usw.)?,
3. Nun folgt die Berechnung des nächsten Punktes.

   Hinweis: *Achten Sie darauf, die neuen Werte erst in x und y zu speichern, wenn Sie die vorherigen Werte nicht mehr benötigen.*
4. Dieser Vorgang wird jetzt sooft wiederholt, bis der Farn als solcher erkennbar ist.



Links: 50.000 Punkte; Rechts: 5.000 Punkte.

Hinweis: *Eventuell muss der Farn verschoben und um einen Faktor vergrößert werden, damit er gut erkennbar ist.*
Hinweis: *Teilen Sie ihren Code in Funktionen auf, um die Lesbarkeit zu verbessern.*
Hinweis: *Diese Aufgabe kommt ohne die Verwendung von* `void draw()` *aus.*

319

31 Z.

Aufgabe 16 – Planetenmodell

Simulieren Sie einen Planeten p, der um eine Sonne s kreist, mit einer korrekten Berechnung von Anziehungskräften.

Zunächst wollen wir die gravitationsbedingte Beschleunigung des Planeten a_p durch die Sonne berechnen:

$$a_p = G \cdot \frac{m_s}{r^2}$$

Dabei ist m_s die Masse der Sonne (welche Sie mit dem arbiträren Wert 3.000.000 belegen können) und G die Gravitationskonstante - welche wir aufgrund dem Rechnen mit Pixeln statt Metern auf grobe 0.001 setzen.

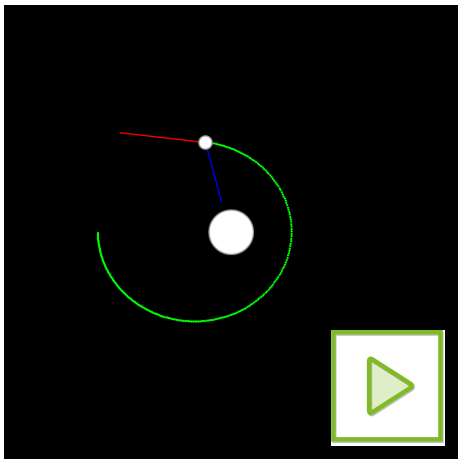
Damit die Beschleunigung funktioniert, muss dem Planeten eine initiale Geschwindigkeit (empfohlen sind 3 Pixel in eine Richtung) gegeben werden. Je nachdem, wie Sie diese Geschwindigkeit variieren, wird der Planet einen anderen Orbit haben.

Der Richtungswinkel für diese Beschleunigung des Planeten lässt sich mit der folgenden Formel berechnen:

$$\alpha_p = \operatorname{atan2}(y_2 - y_1, x_2 - x_1)$$

Dabei ist $p_p = (x_p, y_p)$ die Position des Planeten und $p_s = (x_s, y_s)$ die der Sonne. Die Funktion `atan2(y, x)` ist in Processing bereits vorhanden.

Um die Rechnungen zu erleichtern, wird die Sonne als statisches Objekt angesehen und dessen Bewegung nicht berechnet (obwohl dies mit vertauschten Indices möglich ist - versuchen Sie es also gern!).



Hinweis: *Speichern Sie separat Position und Geschwindigkeit des Planeten als Vektoren.*
Hinweis: *Am Anfang hilft es sehr, die Animation erst einmal mittels* `noLoop()` *anzuhalten, und die Geschwindigkeit und Beschleunigung als farbige Linie kenntlich zu machen.*