MONA NEUBAUER

Programming Basics

Max Rheiner

17. 9.2013 - 1.10. 2013

ZHdK, VIAD, 1. Semester

[1] VORKENNTNISSE

Meine Erfahrungen mit Programmieren sind bis anhin stark eingeschränkt. Bis auf den Code, den ich für meine Webseite benütze und bewirtschafte sowie die Einsicht, die ich in ein paar MATLAB Codes bei meiner Masterarbeit hatte, habe ich bis vor dem 16.September noch nie selbst einen Code geschrieben. Somit war ich gespannt auf die neue Dimension, die sich mir eröffnet.
In der ersten Lektion wird uns ein bisschen Speck durch den Mund gezogen, indem uns Max tolle Projekte vorführt und zeigt was alles möglich sein kann mit Programmieren und im Speziellen mit Processing.
Besonders angetan bin ich von Daniel Rozin's Trash Mirror, Julius Popp's News Waterfall und vom Stone Spray Project, aber auch vom Generativen Design des MIT Labs. Beim Generativen Design geht es darum Regeln zu definieren, anhand von welchen das Programm Bilder und Bildwelten nach den vordefinierten Regeln kreiert.

Nach der Einführung lernen wir was Variabeltypen sind, wie zum Beispiel 'int, float, boolean, char und string'. Variabeln sind alphanummerisch wobei auf die Gross- und Kleinschreibung geachtet werden muss.
Zudem gibt es übergeordnete Variabeltypen (sogenannte globale Variabeln), auf welche das Programm von überall zugreift. Die Variabeln können aber auch lokal definiert werden, somit sind sie nur lokal gültig.
Als nächstes lernen wir Funktionen kennen. Funktionen können auch als definierte Aufgaben verstanden werden. Es kann sich hierbei um bereits bekannte Aufgaben handeln, d.h. Processing ist eine solche Funktion bereits bekannt (zum Beispiel bei 'draw') oder die Funktion oder Aufgabe wird im

Code definiert und kann an beliebiger Stelle wieder aufgerufen werden. Falls ich also zum Beispiel die Aufgabe ‘Haareschneiden’ vordefiniert habe, kann ich diese an jeder Stelle im Programm aufrufen.
Wenn ich sicher bin, dass ich meine Haar immer entsprechend geschnitten haben möchte, habe ich die Möglichkeit diese in einer Bibliothek zu speichern und Processing kann darauf zurückgreifen.

Als nächstes lernten wir Bedingungen kennen:

```
if([boolischer Ausdruck]) {
}
else {
}
```

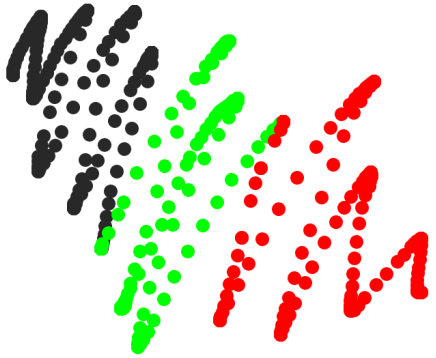
Mit diesem Befehl kann ich Rahmenbedingungen geben, unter welchen meine Funktion ausgeführt werden soll. Denn natürlich sollen meine Haare nur geschnitten werden, wenn sie auch zu lang sind – und ich möchte selber definieren was ‘zu lange Haare’ bedeutet.

[2] ZEICHENPROGRAMM

Als erste Aufgabe mussten wir ein Zeichenprogramm generieren, in welchem mit den Tasten ‘1’ - ‘5’ die Farbe verändert werden. Mit der Linken-Maustaste wird gezeichnet und mit der Rechten-Maustaste radiert. In meiner ersten Lösung verwendete ich eine Ellipse als Pinsel. In meinem zweiten Lösungsansatz wird eine Linie mit zufällig variierender Strichdicke generiert und mit der Taste ‘4’ kann ich eine zufällige Farbe wählen.
Als wichtiges Memo möchte ich die Variabeln ‘mouseX, mouseY, pmouseX, pmouseY’ anfügen, welche mir erlauben die Mausposition im jetzigen und vorherigen Frame zu bestimmen.

Zum Code:

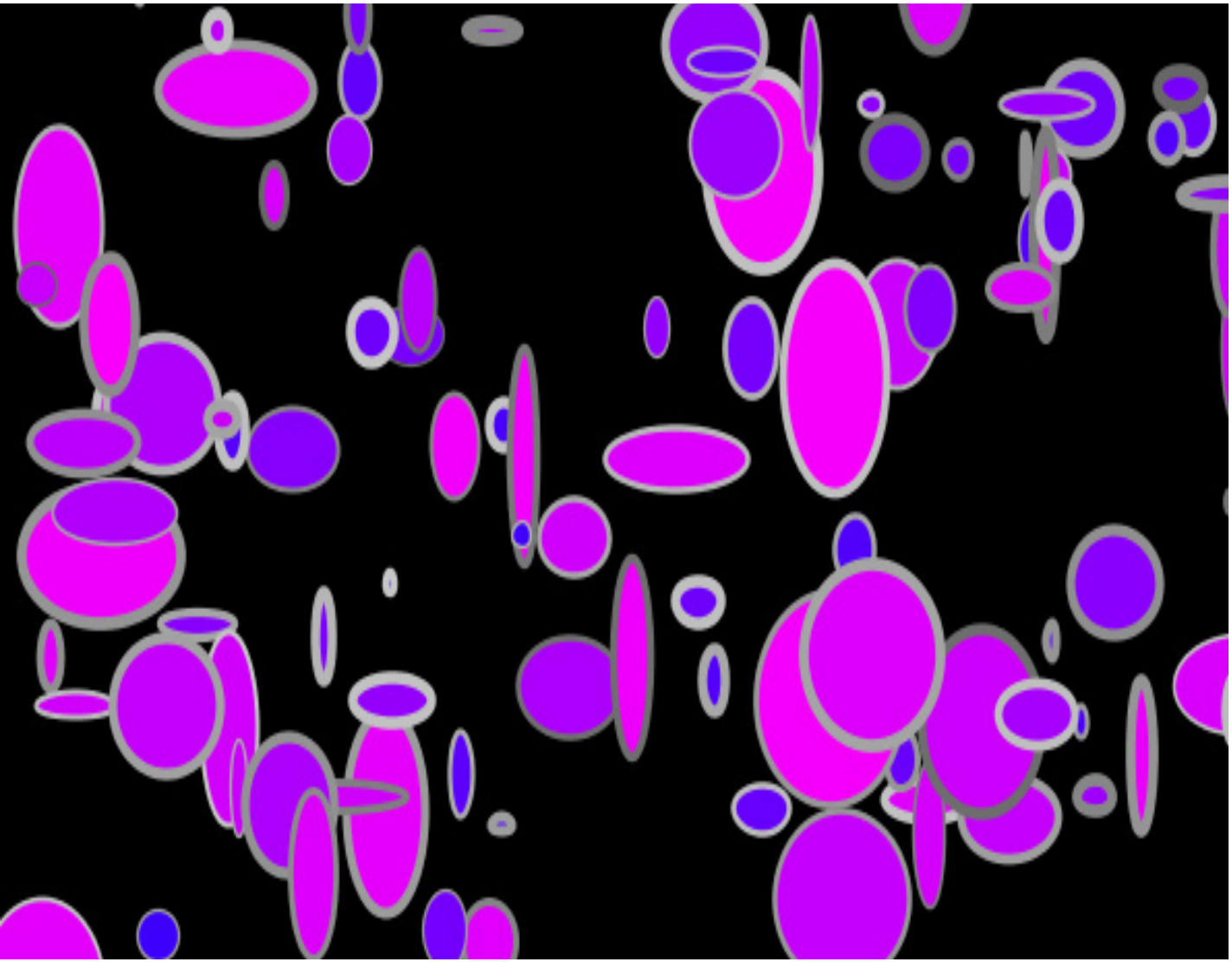
http://blogs.iad.zhdk.ch/codingspace/2013/09/19/zeichenprogramm-2/



Tiff Export aus meinem Zeichenprogramm. Als Pinsel wird eine Ellipse verwendet

Tiff Export aus meinem zweiten Zeichenprogramm. Als Pinsel wird eine Linie verwendet, mit zufälliger Strichdicke

Zufallsellipse: Mit jedem Klick wird eine Ellipse zufälliger Grösse generiert. Je grösser die Ellipse desto rötlicher ist die Füllfarbe.



[3] ZUFALLSELLIPSE

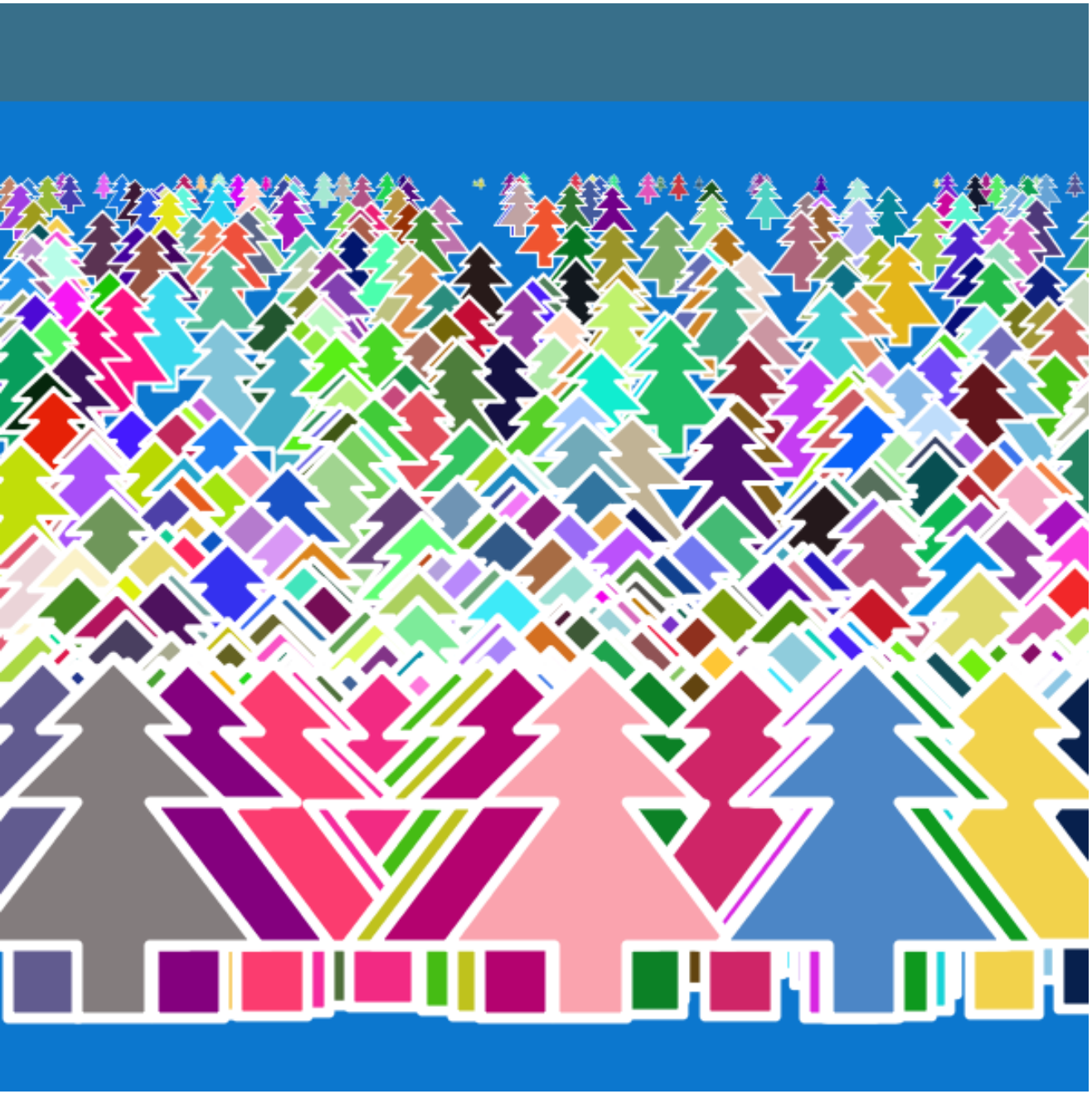
Aufgabe: Zufällige Ellipsen sollen beim Klicken generiert werden. Je grösser die Ellipse, desto rötlicher soll die Füllfarbe sein. Zusätzlich wollte ich mein Bild abspeichern.

Die Lösung der vorgegebenen Aufgabe fand ich, indem ich eine zufällige Variabel definierte, die von der Grösse sowie der Farbe abhängig ist:

```
void draw()
{
  number1 = random(60, 250);
  stroke(random(100, 200));
  fill(number1, 0, 252);
  strokeWeight(random(1, 5));
  ellipse(random(0, width), random(0, height), number1/300*random(10,
  100), number1/300*random(10, 100));
}
```

Als zusätzliche Memos habe ich folgende Funktionen herausgeschrieben:
redraw(); - bestimmt wann draw erneut ausgeführt werden soll
frameRate(); - kann die Rate, in der draw(); aufgerufen wird, kontrollieren
noLoop(); - die Funktion draw wird nur einmal aufgerufen.
Um das Bild abzuspeichern verwendete ich saveFrame();

Zum Code: http://blogs.iad.zhdk.ch/codingspace/2013/09/19/zufallsellipse/



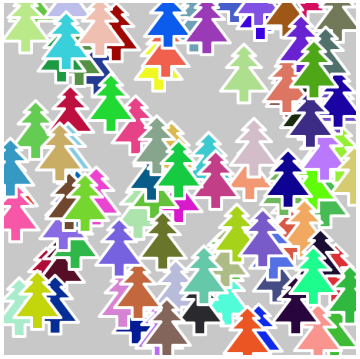
Zufällig generierter Wald: Die Horizontfarbe, die Hintergrundfarbe, sowie die Farbe der einzelnen Bäume werden per Zufall generiert. Die Bäume werden von oben nach unten gezeichnet und sind bei kleinem y kleiner.

[4] WALD

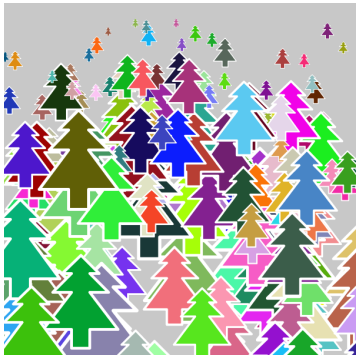
Aufgabe war ein Programm zu schreiben, das einen Wald generiert. In meinem ersten Lösungsansatz zeichnete ich Bäume mit einer Zufallsfarbe an einer zufälliger Position auf der gesamten Zeichenoberfläche. Mit der Taste ‘5’ wird das Bild neu generiert, mit der Taste ‘1’ kann das Bild als .tiff gespeichert werden.
In einem zweiten Schritt wollte ich, dass die Bäume von oben nach unten gezeichnet werden, damit die Bäume mit einem grösseren y-Wert die Bäume mit einem kleinen y überlagern. Zudem sollten oben mehr Bäume sein als unten und, da ich eine perspektivische Wirkung erzielen wollte, sollten sie oben kleiner sein. Schwierigkeiten hatte ich vor allem, eine Funktion zu finden, welche die Bedingung erfüllt bei einem kleinen i (der i.Baum) exponentiell mehr Bäume zu zeichnen. Mit einer Cosinus-Funktion konnte ich dies lösen. Ich musste allerdings das i, welches ich als ‘int’ definiert hatte, zuerst in ein ‘float’ umrechnen, ansonsten wurde der Wert ‘0’ ausgegeben.

Gelernt habe ich bei dieser Übung vor allem über mich selbst: ich war so konzentriert auf den Code und die Funktionsweise meines Programms, dass ich meinen Wald eher unbewusst als bewusst gestaltet habe.

Zum Code: http://blogs.iad.zhdk.ch/codingspace/2013/09/20/wald/



Gleich grosse Bäume, welche zufällig auf der Oberfläche gezeichnet werden.



Die Bäume sind bei kleinem y kleiner. Sie werden auf der Zeichenfläche zufällig gezeichnet.

Rotierende Smileys: Das grosse Smiley folgt dem Mauszeiger, während das kleine Smiley in gleich bleibendem Abstand um das grössere gleichmässig rotiert.

[5] ROTATION EINER GRAFIK

Aufgabe war ein Programm zu schreiben in dem ein Smiley dem Mauszeiger folgt. Dazu soll ein kleineres Smiley um das Grössere in einer Umlaufbahn kreisen.
Hier lernte ich die Befehle

pushMatrix()
popMatrix()

kennen. Anhand der Funktion pushMatrix() kann das gesamte Koordinatensystem verschoben und rotiert werden (die Matrix wird transformiert). Mit popMatrix() wird das originale Koordinatensystem für folgende Operationen/ Funktionen wieder hergestellt.

Zum Code:

http://blogs.iad.zhdk.ch/codingspace/2013/09/24/rotation-einer-graphik/

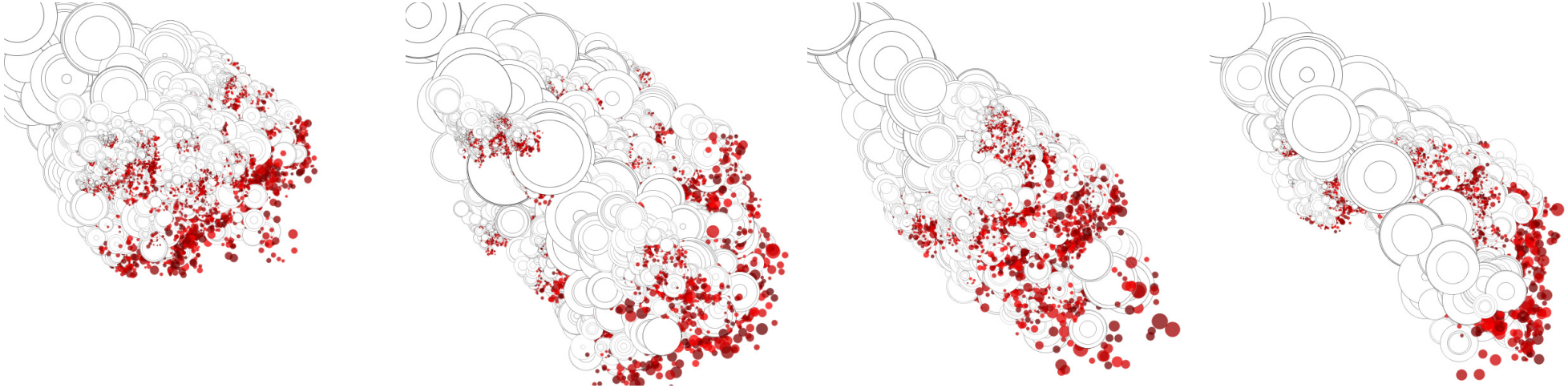
[6] REKURSIVE FUNKTIONEN

Rekursive Funktionen eigenen sich für generatives Design. Die Regeln der Gestaltung werden im Programm vorgegeben und das Programm zeichnet das Design nach diesen vorbestimmten Regeln.

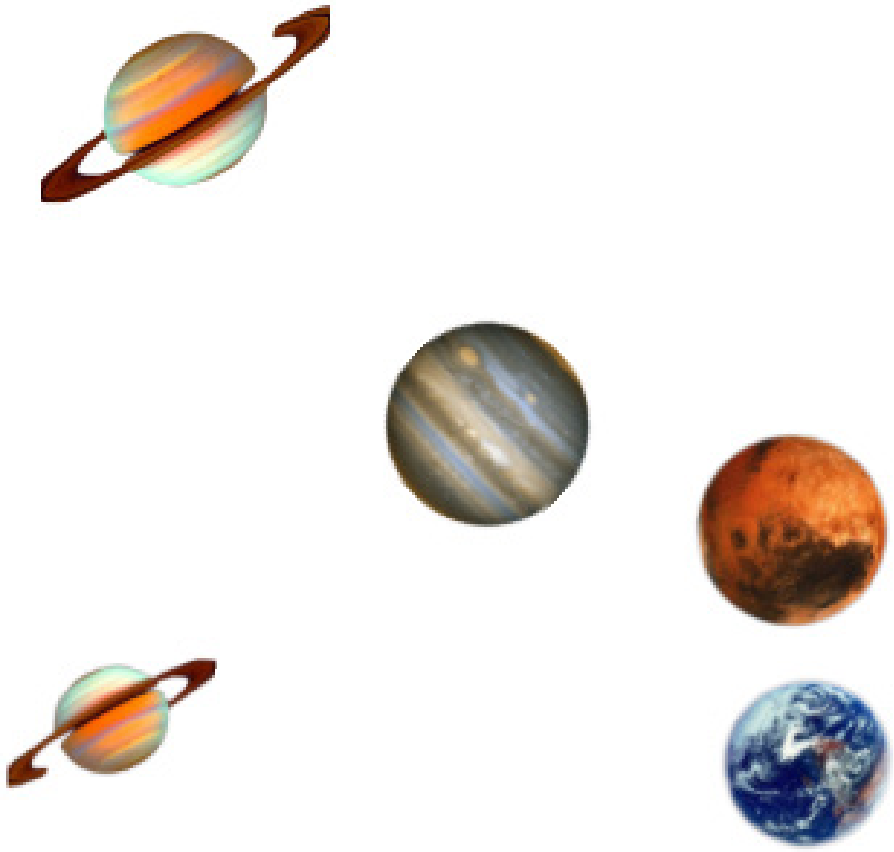
Memo für rekursive Funktionen:
return; – kann für Unterbruch einer Funktion benützt werden

Zum Code:

http://blogs.iad.zhdk.ch/codingspace/2013/09/24/rekursive-funktionen/



Planetensystem: Der Planet oben Links folgt dem Mauszeiger. Die Erde (unten rechts) dreht um ihre eigende Achse, während die anderen drei Planeten in ihrer eigenen Laufbahn mit unterschiedlicher Umlaufgeschwindigket um die Erde kreisen.



[7] PLANETENSYSTEM

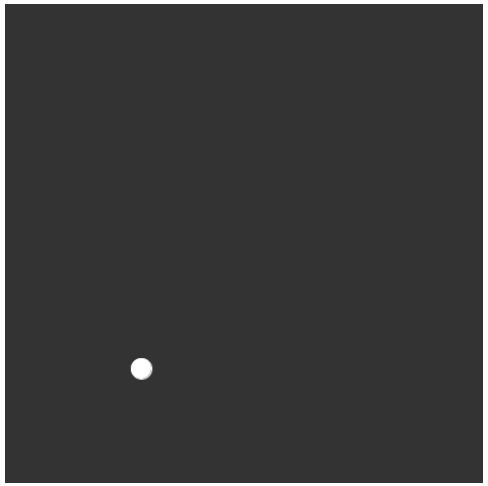
Von einem existierenden Code und Bildmaterial ausgehend sollte ein Planet generiert werden, der dem Mauszeiger folgt. Die geladenen Bilder mussten freigestellt werden. Ich habe zudem mein Planetensystem um die Erde kreisen lassen, wobei jeder Planet eine eigene Umlaufbahn bekam, mit eigener Umlaufgeschwindigkeit.
Erkenntnisse:

Durch meinen Image- und Zahlen-Array
float[] Rotg = new float[4];
PImage[] imageList = null;
imageList = new PImage[4];
imageList[0] = loadImage(„./images/4.png“);
imageList[1] = loadImage(„./images/2.png“);
imageList[2] = loadImage(„./images/3.png“);
imageList[3] = loadImage(„./images/1.png“);
Rotg[0] = 0.01;
Rotg[1] = 0.02;
Rotg[2] = -0.01;
Rotg[3] = 0.011;

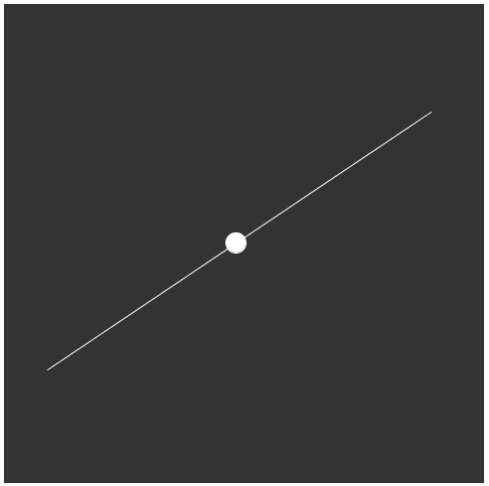
konnte ich Tabellen zusammenstellen, welche für die i.Stelle die entsprechende Rotationsgeschwindigkeit zum entsprechenden Bild verwendete.

Zum Code:

http://blogs.iad.zhdk.ch/codingspace/2013/09/24/planetensystem/



Beim ersten Klick wird der Startpunkt der Linie und der Ball gezeichnet



Beim zweiten Klick wird der Endpunkt der Linie definiert. Der Ball beginnt auf der Linie hin und her zu oszilieren

[8] BALL AUF EINER LINIE

Von einem Code ausgehend sollte eine Linie per Mausklick gezeichnet werden. Ein Ball bewegt sich auf der Linie, wobei wir das Beispiel so anpassen sollten, dass er sich vorwärts und rückwärts auf der Linie bewegt.
Ich habe die Aufgabe mit dem Lösungsansatz

- Anzahl Klicks ungerade = Start
- Anzahl Klicks gerade = Ende

glöst. Hierzu habe ich die Operation modulo (%) verwendet.

Memo modulo:

- 5%4 = 1 (der Rest von 5:4 ist 1)
- 10%5 =0 (der Rest von 10:5 ist 0)

Im Bespiel des Balles natürlich :
click_number%2 == 0
click_number%2 == 1

Zum Code:

http://blogs.iad.zhdk.ch/codingspace/2013/09/24/animation-ball-auf-einer-linie/

BreakOut: Als Bühne wollte ich das ‚Racket' als Schiff zeichnen, die ‚Bricks' sollten zu Fischen werden, die aus dem Wasser springen und mit dem Ball getroffen werden.
Hierzu wollte ich anstelle einer Grafik eigene Zeichnungen, verwenden damit der Eindruck entsteht, als spiele ich mit ausgeschnittenem Papier auf Papier.
Zudem wollte ich den Ton verändern, dass sobald die Fische getroffen werden, ein ‚Splash' erklingt.

[9] ENDAUFGABE – BREAKOUT

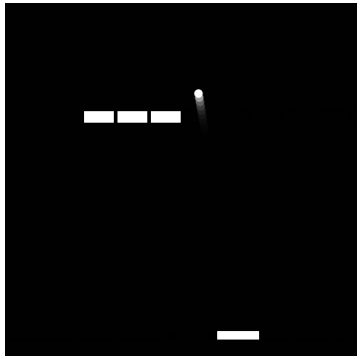
Ausgehend von einem Code des Spieles ‚BreakOut' gestalteten wir unser eigenes Spiel.

Bevor ich mit der Gestaltung begann, formulierte ich für mich das Konzept meines Spieles. Ich wollte ein Boot zeichnen, welches die Fische, die aus dem Wasser springen mit dem Ball treffen muss.
Dannach begann ich mit dem Programmieren aller Funktionen. Meine Fische sollten in einer zufälligen Ellipse an einem zufälligen x Wert aus dem Wasser sprangen. Dies erreichte ich anhand von:

```
void drawRot()
{
_rot += _rotInk;
    pushMatrix()
```

Startseite für meine Lösung des Programmes

Ausgangslage des Spieles



```
    translate(_randomcenterX, _randomcenterY);
    rotate(radians(_rot));
    translate(-_radius,0);
    rotate(radians(-90));
    pushStyle();
    imageMode(CENTER);
    image(_brick,0,0);
    popStyle();
  popMatrix();
}
```

Da ich hierbei das gesamte Koordinationsystem für jeden einzelnen Fisch rotierte und translatierte, bestand für mich die grösste Schwierigkeit, die Fische zu lokalisieren für die Kollision mit dem Ball. Dies konnte ich mit der Funktion getPos();

```
PVector getPos()
 {
   PMatrix2D trans1 = new PMatrix2D();
   PMatrix2D trans2 = new PMatrix2D();
   PMatrix2D rot1 = new PMatrix2D();
   PMatrix2D res = new PMatrix2D();

   trans1.translate(_randomcenterX, _randomcenterY);
   rot1.rotate(radians(_rot));
   trans2.translate(-_radius,0);
   res.apply(trans1);
   res.apply(rot1);
   res.apply(trans2);
   PVector vec = new PVector();
   res.mult(new PVector(),vec);

   return vec;
 }
```

Zudem musste ich folgenden Code einfügen, damit die Collision gewährleistet wurde:

```
void set(PVector pos,PVector dir,float dampV)
 {
   _pos = pos.get();
   _dir = dir.get();
   _dampV = dampV;
 }
```

Nachdem ich alle Funktionen so hatte, wie ich wollte, begann ich mit der Gestaltung des Spieles. Dabei wollte ich ein Gefühl von etwas Handgemachten vermitteln, als ob ich mit ausgeschnittenen Zeichnungen auf einem Papier spielen würde.

Zum Code:

http://blogs.iad.zhdk.ch/codingspace/2013/10/04/endaufgabe-break-out/