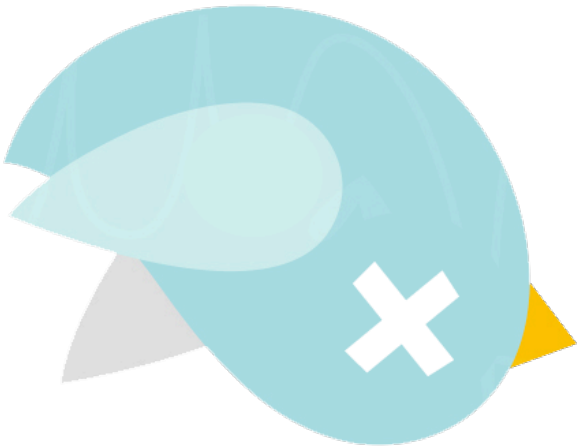
BASIC PROGRAMMING

DOKU THOMAS SCHERTENLEIB

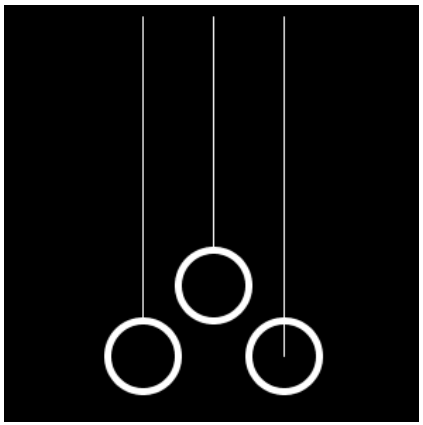


Inhaltsverzeichnis

18.09.2013 – Erste Übungen und Malkasten

Am zweiten Tag wurden wir über die Grundfunktionen von Processing aufgeklärt. So erfuhren wir, was “size” und “background” etc. bedeutet, und konnten zum ersten Mal ein wenig an den Parametern einfacher Formen rumspielen. Danach hat uns Max mit der Bedeutung von Variablen vertraut gemacht. So konnten wir dann zum Beispiel anhand einer definierten globalen Variable Längen definieren und diese dann in den Koordinaten zweier Ellipsen zum Ausdruck bringen.

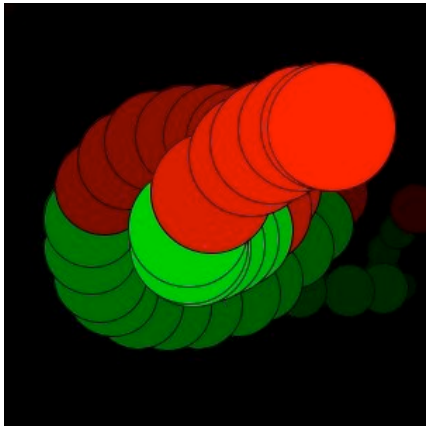
Einfache Formen



Globale Variablen

Das erarbeitete Wissen konnten wir danach weiter anwenden. Diesmal in der “draw” Funktion als Animation. In einer Version zeigte sich das in einer Ellipse, die wächst.

In einer anderen Version, die wir um Teil zusammen in der Klasse gemacht haben, sehen eine Ellipse, die wächst und in der unteren Hälfte grün und in der oberen Hälfte rot ist.



Ellipse mit Schweif

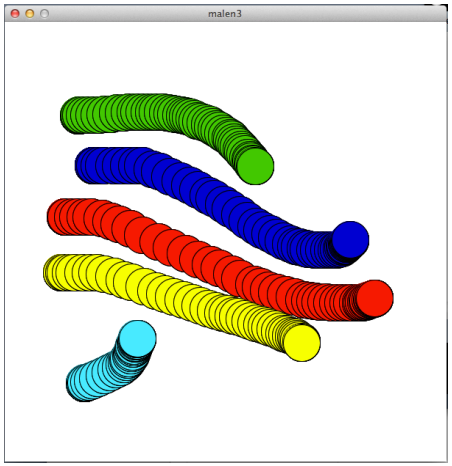
Detailiertere Beschreibung und Code auf http://blogs.iad.zhdk.ch/codingspace/2013/09/23/18-09-2013/

18.09.2013 – Erste Übungen und Malkasten

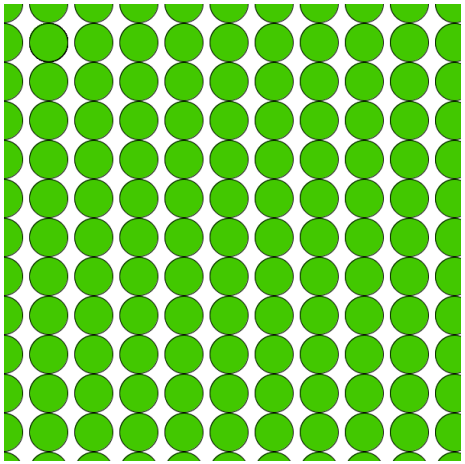
Als letzte Aufgabe an diesem Tag mussten wir ein kleines Malprogramm schreiben. In diesem Malprogramm sollte es mindestens möglich sein, dass man mit den Zahlen 1 bis 5 auf dem Keyboard verschiedene Farben auswählen, mit der linken Maustaste malen und mit der rechten Maustaste radieren kann.

Ich habe das so gelöst, dass ich zuerst mit der “color” Funktion eine Farbe definiert habe, die Farbe habe ich “farbe” genannt. In der “draw” Funktion habe ich danach mit der “if (mousePressed) “Bedingung” gearbeitet. Mit switch(mouseButton) LEFT oder RIGHT wird entschieden, ob man entweder mit “farbe” oder mit weiss (radieren) malt. Gemalt wird mit einer Ellipse die sich immer an der Mausposition befindet. In der „keyPressed“ Funktion werden dann mit „switch(key)“ mit den Tasten 1 bis 5 auf dem Keyboard die Farben gewechselt. Mit den Tasten „w“ und „q“ kann man die Pinselgrösse verändern. Dies funktioniert so, dass ich eine globale Variable mit der Grösse 50 definiert habe. Diese Variable haben ich „groesse“ genannt und danach in die beiden Radiusgrössen der Ellipse eingegeben. Jedes Mal, wenn jetzt „w“ gedrückt wird, wird „groesse“ um 5 Pixel vergrössert und wenn „q“ gedrückt wird, um 5 Pixel verkleinert.

Schlussendlich wollte ich mich auch noch an einer „Repetition“ versuchen, dies habe ich mit einer „for“ Schleife versucht. Wenn „r“ gedrückt wird, dann wird die Ellipse mit dem Abstand vom Nullpunkt der X und Y-Achse zu den Koordinaten der Mausposition in beiden Richtungen zwanzig Mal repetiert.



Malkasten



Repetition

Detailiertere Beschreibung und Code auf http://blogs.iad.zhdk.ch/codingspace/2013/09/24/19-09-2013/

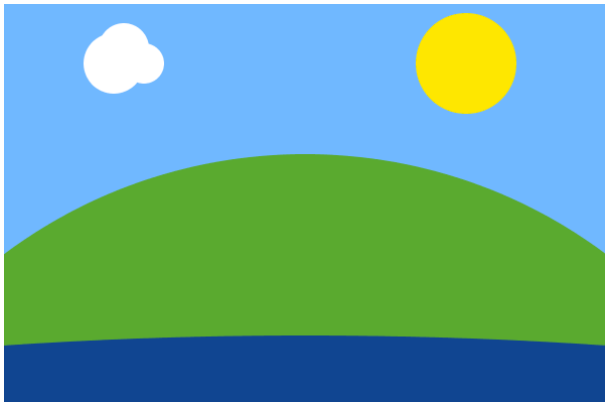
19.09.2013 - Wald

Wir hatten die Aufgabe einen zufälligen Wald zu zeichnen. Zuerst habe ich das so gelöst, dass ich im Illustrator einen Baum gezeichnet habe, den dann mit einer “for” Schlaufe auf der X und Y-Achse repetiert habe.

Baum (erster Anlauf)

In einem zweiten Anlauf habe ich zuerst eine Funktion erstellt, die mit „if“ und „else if“ zwei Typen von Bäumen zeichnet. Einen Laubbaum und eine Tanne. Die Farben der Blätter beider Baumarten werden im Grünbereich random generiert.

Die Bäume werden immer zufällig verteilt, wobei immer eine Tanne vor einem Laubbaum steht. Um zu erreichen, dass die Bäume hintereinander und nicht übereinander standen, habe ich sie mit „translate“ in der X-Position in einem bestimmten Bereich verschoben und in der Y-Position von der übergeordneten Variable „i“ einer „for“ Schlaufe abhängig gemacht.



Version 1 Wald (zweiter Anlauf)

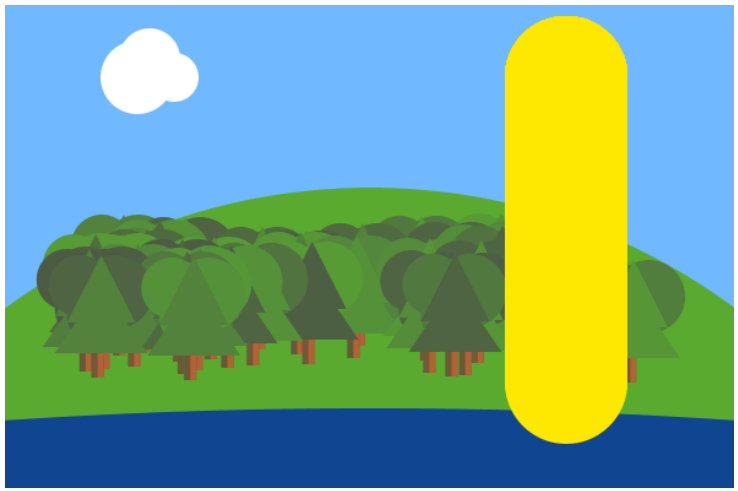
Weil ich die Sonne animieren wollte, hatte ich ab diesem Zeitpunkt zwei verschiedene Versionen der Übung. In der einen Version ist der “background” in der “draw” Funktion, wobei in diesem Fall die Bäume übermalen werden und die Sonne schön animiert ist.

Detailiertere Beschreibung und Code auf http://blogs.iad.zhdk.ch/codingspace/2013/09/24/19-09-2013/

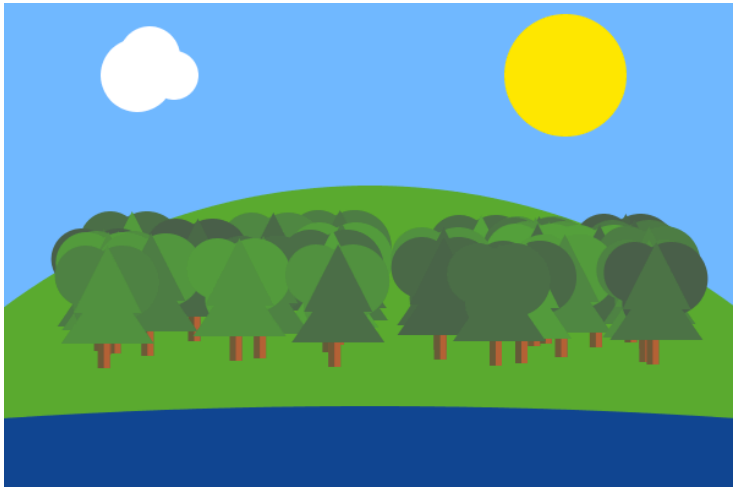
19.09.2013 - Wald

Und in der anderen Version sind die Bäume schön zu sehen aber die Sonne zieht einen Strich wegen der Animation.

Ich habe noch bis Mittwoch den 25. September versucht daran herum zu tüfteln und es dabei noch mit einer “ArrayList” und dem “PVector” versucht. Dabei habe ich es geschafft alle Bäume “random” zu platzieren und sie mit den x und y Vektoren immer wieder an dieselbe Stelle zu zeichnen. So blieb dieses Mal der Schweif der Sonne aus. Da ich aber auch die Farbe der Bäume “random” erstelle, wechseln diese nun ständig die Farbe. Dieses Problem habe ich leider nicht mehr beheben können. Ich habe es auf die gleiche Weise versucht wie mit der Position, was aber irgendwie nicht funktioniert hat.



Version 2 Wald (zweiter Anlauf)



Version 3 Wald (zweiter Anlauf mit PVector)

Detailiertere Beschreibung und Code auf http://blogs.iad.zhdk.ch/codingspace/2013/09/24/19-09-2013/

20.19.2013 - Smiley, Rekursive Funktionen und Fraktale

Smiley:

Zuerst habe ich aus den beiden „pushMatrix“ und „popMatrix“ Befehlen einen einzigen gemacht. So beziehen sich alle vorherigen Funktionen und Anweisungen des grossen Smileys auch auf den kleinen Smiley. Den kleinen Smiley musste ich danach auf der X und Y-Achse noch um 60 Pixel verschieben, so dass dieser sich in einer Umlaufbahn um den grossen befindet.

Damit der kleine Smiley um den grossen rotiert, habe ich eine globale „float“ Variable mit dem Namen „orbit“ bestimmt und diese in der „draw“ Funktion als „orbit = orbit+0.01“ angegeben.

Weil ich wollte, dass der kleine Smiley gleichzeitig auch noch um seine eigene Achse rotiert, habe ich eine zweite globale „float“ Variable mit dem Namen „drehung“ bestimmt und dem kleinen Smiley eine rotate-Funktion „drehung+=0.1“ angegeben. Indem ich die „mouseX“ und „mouseY“ Koordinaten in die „translate“ Funktion des grossen Smileys eingegeben habe konnte ich erzielen, dass der Smiley der Maus folgt. Jetzt störte mich nur noch, dass der grosse Smiley abhängig von der Mausposition skaliert und rotiert, weshalb ich die beiden Funktionen mit „//“ auskommentiert habe.

Detailiertere Beschreibung und Code auf http://blogs.iad.zhdk.ch/codingspace/2013/09/25/20-09-2013-smiley-rekursive-funktionen-und-fraktale/

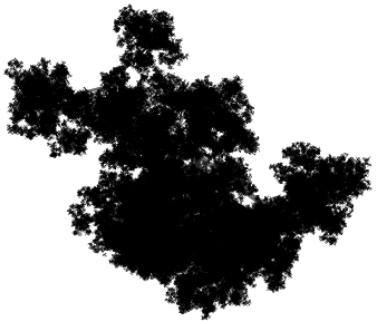
20.09.2013 - Smiley, Rekursive Funktionen und Fraktale

Rekursive Funktionen und Fraktale:

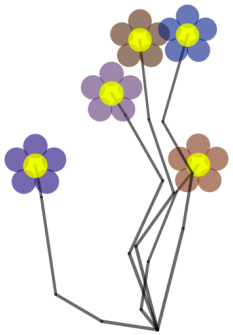
Danach haben wir uns mit rekursiven Funktionen und Fraktalen auseinandergesetzt. Wir bekamen einen Baum, bei dem man n-Mal Äste und Verzweigungen generieren konnte und dem mal beim n-ten Mal sagen konnte, dass er jetzt bitte aufhören sollte Äste zu generieren. Statt einem weiteren Ast wird dann am Schluss ein Blatt generiert.

In einem ersten Anlauf habe ich einfach die Blätter entfernt und ein bisschen mit den Parametern herumgespielt. Dabei ist so etwas wie eine Wolke entstanden.

Danach habe ich einen Blumenstrauss erstellt. Die Blätter der Blüten sind mit einer “for” Schleife erstellt, die um die Blüte rotiert.



erster Anlauf



Blumenstrauss

Detailiertere Beschreibung und Code auf http://blogs.iad.zhdk.ch/codingspace/2013/09/25/20-09-2013-smiley-rekursive-funktionen-und-fraktale/

22.09.2013 - Planeten, Animation und Klassen

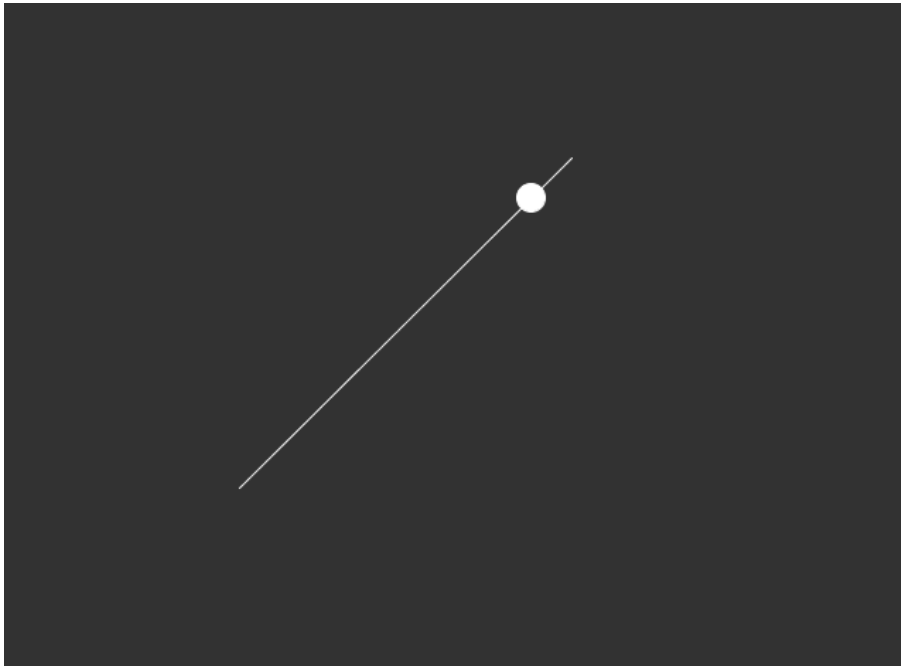
Planeten:

An der Aufgabe habe ich nichts kreativ erweitert. Ich habe einfach das Bild hereingeladen und mit translate an mousX und mouseY befestigt. Die anderen Planeten habe ich auskommentiert.

Animation – Ball auf einer Linie:

Wir bekamen einen Code. Mit diesem Code konnten wir bei drücken der Maustaste und gleichzeitigem ziehen eine Linie erstellen, auf der sich ein Ball in Richtung der Maus bewegt. Wir sollten diesen Code so verändern, dass die Linie erst beim zweiten Mausklick sichtbar wird, sie aber vom ersten Mausklick zum zweiten geht. Zusätzlich soll sich der Ball immer von einem Punkt zum anderen bewegen. Ich habe die Aufgabe mit einem clickCount gelöst. Das hin und her gehen des Balls habe ich mit einer Sinusfunktion gelöst.

Da dieses Ergebnis nicht besonders elegant war, war ich sehr dankbar, dass wir für diese Aufgabe noch eine Musterlösung von Max bekamen.



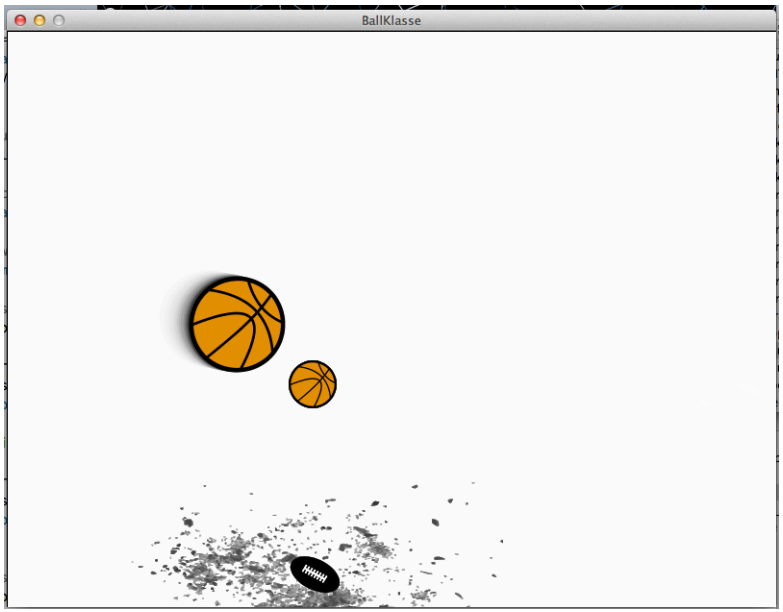
Ball auf einer Linie

Detailiertere Beschreibung und Code auf http://blogs.iad.zhdk.ch/codingspace/2013/10/02/22-09-2013-planeten-animation-und-klassen/

22.09.2013 - Planeten, Animation und Klassen

Klassen – Bouncing Ball

Ich habe diese Aufgabe grundsätzlich so gelöst, wie es in der Aufgabestellung gefragt war und nur in diesem Sinne kreativ erweitert, dass ich dem Programm gesagt habe, dass immer wenn sich der Ball am Rand der Spielfläche befindet ein Bild von Staub geladen werden soll. So sieht es aus, als würde jedes Mal Staub aufwirbeln, wenn der Ball an einer Seite aufprallt.



Detailiertere Beschreibung und Code auf http://blogs.iad.zhdk.ch/codingspace/2013/10/02/22-09-2013-planeten-animation-und-klassen/

Endaufgabe - Homing Pigeons

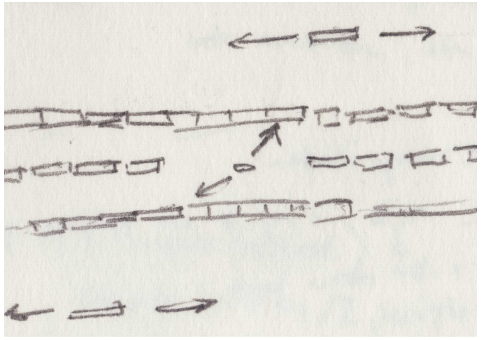
Von BreakOut zu HomingPigeon:

Schlussendlich bekamen wir von Max die Aufgabe das von ihm programmierte Spiel „BreakOut“ sinnvoll zu erweitern. Dabei ist bei mir das Spiel „HomingPigeons“ entstanden, bei dem man eine Brieftaube spielt, die zuerst Briefe aufsammelt und diese danach in ein Haus bringen soll. Dabei muss die Taube verschiedene Hürden überwinden. In einer hypothetischen fertigen Version reichen diese Hürden von Giftflaschen, die der von den Tauben genervte Nachbar streut, bis zur Hauskatze, der die Taube ein gefundenes Fressen ist. In der jetzigen Fassung kann man das erste Level spielen, in dem die grösste Hürde ist an Briefe zu kommen, die in zerstörbaren Briefkästen sind. Auch habe ich mich in der abgegebenen Fassung noch lange mit einem Countdown auseinandergesetzt, der von hundert auf null zählen sollte, wenn eine Giftflasche aufgenommen wurde. Ist der Countdown auf null, ist das Spiel auch „Game Over“. Da der Countdown aber immer noch nicht funktioniert habe ich ihn im Code auskommentiert.

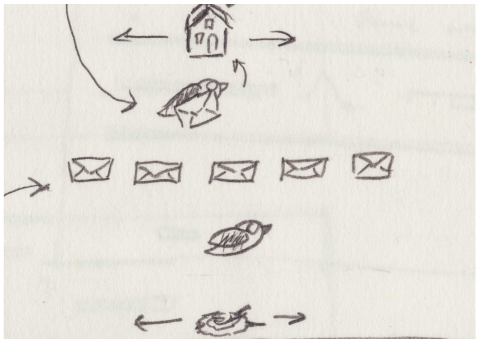
Erste Ideen:

Zuerst wollte ich das spiel so machen, dass man zu zweit gegen einander spielen konnte. Gewonnen wird nach Punkten und verlorenen Bällen.

Danach wollte ich ein das spiel so mache, das man das spiel auch zu zweit spielen kann. Dieses Mal aber indem man zusammen spielt. Jemand bewegt das Häuschen und jemand anderes das Nest. So wäre in diesem Fall das Häuschen bis man es betreten kann wie ein zweiter Schläger gewesen, bis man es nach einsammeln aller Briefe betreten hätte können.



Idee 1

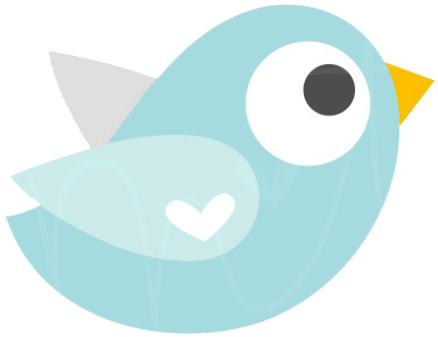


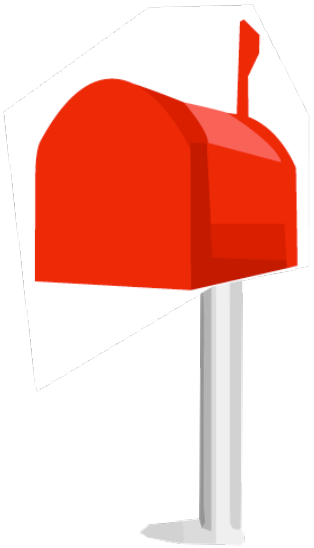
Idee 2

Endaufgabe - Homing Pigeons

Grafik und Sound:

Das Schwierigste an der ganzen Aufgabe fand ich sich im Code zurecht zu finden. Hatte man sich aber erst einmal ein wenig darin zurecht gefunden, wurde es immer einfacher im Code zu navigieren und kleine Veränderungen vorzunehmen. Ich habe damit angefangen, dass aus dem Racket ein Taubennest mit Küken gemacht habe, aus dem Ball die Brieftaube und aus den Bricks Briefe. Für die „Letterbox“ und für das „Poison“ habe ich die Klasse der „Bricks“ erweitert. So konnte ich spezifisch Veränderungen vornehmen, die Grundeinstellung der Bricks jedoch beibehalten.

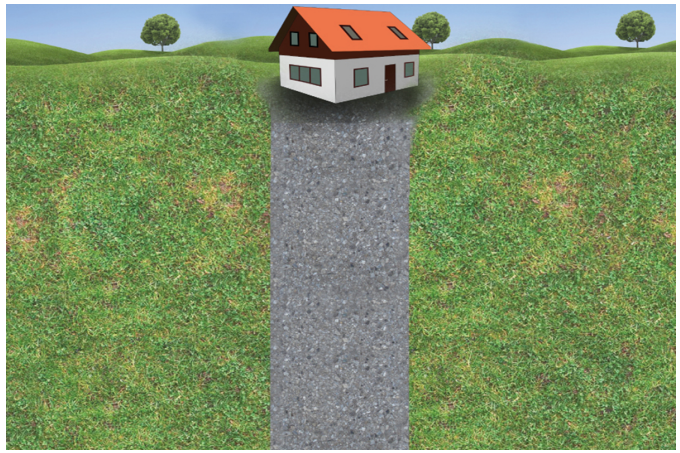




Endaufgabe - Homing Pigeons

Zusätzlich habe ich für alle Modi eine spezifisches Hintergrundbild erstellt und dieses dann immer thematisch mit Sound untermalt.

Menu



In Game

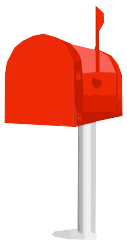
Level Finished

Game Over

Endaufgabe - Homing Pigeons

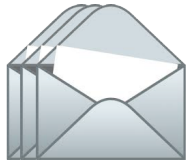
Wie die Briefkästen geöffnet werden können:

Das erste Mal ein wenig schwierig wurde das, als ich erreichen wollte, dass die Briefkästen mehrmals getroffen werden müssen, bis sie verschwinden und sie bei jedem mal treffen ein wenig mehr kaputt gehen. Dafür habe ich Typen definiert. Für „Letterbox“ Typ=1, für „Poison“ Typ=2 und für „Brick“ Typ=0. Zusätzlich habe ich einen Zähler „zaehler“ in „Letterbox“ eingebaut. Dann habe ich gesagt, dass wenn der Typ nicht 1 ist, dann wird er beim Aufprall mit dem Ball entfernt. Wenn der Typ 1 ist und der „zaehler“ grösser als 0, dann wird ihm immer 1 abgezogen. Ist der „zaehler“ alles andere als grösser als 0, wird die „Letterbox“ aufgelöst. Zusätzlich wird der „Letterbox“ angegeben, dass wenn der „zaehler“ grösser als 2 ist, dann ist im Game auf dem Bildschirm das Bild eines geschlossenen Briefkastens zu sehen, wenn er grösser als 1 ist, das Bild eines offenen Briefkastens und bei allem anderen das Bild eines Bündels von Briefen.



Briefkasten 1

Briefkasten 2



Briefkasten 3

Endaufgabe - Homing Pigeons

Wie man die Levels designen kann:

Da ich ein Spiel machen wollte, in dem man mehrere Levels spielen kann und ich die Levels selber designen wollte, habe ich die zufällige Verteilung der Bricks aus dem Spiel herausgenommen. So konnte ich die Bricks danach einzeln auf dem Koordinatensystem verschieben.

Wie man das Level beenden kann:

Ziel des Spiels ist es, die eingesammelten Briefe schlussendlich in einem kleinen Häuschen am oberen Spielfeldrand abzugeben. Wenn man am Schluss mit der Brieftaube auf dieses Häuschen gelangt, ist das Level beendet. Dazu habe ich einen neuen Gamemodus erstellt, den Gamemodus „LevelFinished“. In jeder „updateScene()“ überprüfe ich ob schon alle Briefe getroffen wurden. Falls ja, kann das Spiel beendet werden in dem der Vogel in den Bereich des Häuschens in der Mitte des oberen Bildrandes geht. Wenn die endFlag true ist und die Taube in den Bereich, in dem sich das Haus befindet, ist das Level beendet in dem auf den Gamemodus „LevelFinished“ gewechselt wird. In anderen Worten: Die Brieftaube kann in das Häuschen gehen und das Level beenden, wenn alle Briefe und entweder null, eine oder beide Giftflaschen aufgenommen wurden.