Programming Basics Dokumentation

17. - 27. September 2013

Jonas Scheiwiller

Einleitung

Dieses Dokument soll die zwei Wochen dokumentieren, die wir im Kurs ‘Programming Basics’ mit Processing – eine Programmiersprache mit dazugehöriger Umgebung – verbracht haben.

Um den Text flüssig und leicht leserlich zu gestalten werde ich versuchen auf Code-Fragmente und Befehle zu verzichten.
Viel eher möchte ich mich auf gesetzte und erreichte Ziele, wichtige Einsichten und auf meinen generellen Lernprozess konzentrieren.

Codes und Blog-Posts, die während des Kurses entstanden sind und die zwei Wochen Beschäftigung mit Processing ebenfalls beschreiben, sind unter dem nebenstehenden Link zu finden.
Ebenfalls wird im weiteren Text auf am rechten Rand immer wieder auf einzelne Artikel verwiesen, auch um eine Einsicht in den dortigen Code zu gewähren.

http://blogs.iad.zhdk.ch/codingspace/author/jonasscheiwiller/

Erste Aufgabe

Nach den absoluten Programmier-Basics, welche für mich von anderen Programmiersprachen her bekannt und dementsprechend leicht zu verstehen waren, wurde uns die erste Aufgabe gestellt: ein Zeichnungseditor sollte erstellt werden.
Da man durch Anwendung und probieren am besten lernt, half auch diese Aufgabe, auch wenn Sie mit den verlangten Features noch relativ einfach umzusetzen war, zu einem besseren Verständnis wie Processing funktioniert und was die Syntax von relativ grundlegenden Befehlen ist.

http://blogs.iad.zhdk.ch/codingspace/2013/09/19/processing-drawing-editor/

Im genannten Beispiel, lernte ich beispielsweise, was die draw-Funktion genau macht, welche für jedes Frame aufgerufen wird (falls nicht anders definiert). Eine solche sich automatisch repetierende Funktion war für mich doch etwas ziemlich Neues.
Ebenfalls interessant war festzustellen, dass pMouse die Koordinaten der Maus vom letzten Frame wiedergibt.

Ansonsten war es allgemein eine gute Übung sich wieder ein wenig mit dem Programmieren zu beschäftigen und sich schrittweise and Processing heranzutasten.

Weiteres Wichtiges

http://blogs.iad.zhdk.ch/coding-space/2013/09/19/random-circle/

Nach der ersten Aufgabe folgten weitere grundlegende Programmier-Infos sowie weitere Aufgaben.
Das Prinzip von zufällig generierten Zahlen und wie man dieses nutzen kann, um zufällige visuelle Outputs zu generieren, wurde eingeführt. Die gestellte Aufgabe drehte sich um zufällig generierte Ellipsen und eine von deren Grösse abhängige Einfärbung.
Diese Aufgabe war relativ leicht umzusetzen, da für mich eigentlich keine neuen Infos dazugekommen waren. Trotzdem, wie auch schon zuvor angedeutet, bin ich der Meinung, dass man – vor allem auch beim Programmieren – nur durch Anwendung wirklich dazulernt.

Schleifen im Wald

http://blogs.iad.zhdk.ch/coding-space/2013/09/20/magic-wood/

Die nächste Aufgabe wurde uns nach der Behandlung des Themas Schleifen gestellt.
Diesmal wurde uns jedoch viel mehr Zeit gewährt, um die Aufgabe zu erfüllen und eben dies ermöglichte einem auch über das minimalistische Ziel hinaus zu gehen.
Einen kleinen Wald mithilfe von Schleifen zu generieren hatte ich innert kürzester Zeit geschafft, aber dass der Wald dann auch einigermassen ansprechend aussieht, dies stellte mich vor eine neue Herausforderung.

Ich beschloss, mich im Internet zu bedienen; ich lud von openprocessing.org den Code herunter, der das Zeichnen eines mehr oder weniger real aussehenden Baumes übernahm. Der Code dazu befand sich innerhalb einer Klasse. Dies bedeutete, dass ich mich mit Klassen auseinandersetzen musste. Das tat ich zwar nicht zum ersten Mal, trotzdem war – und ist es wohl teils noch immer – relativ unerforschtes Terrain für mich. Dem zum Trotz fand ich mich ziemlich schnell damit zurecht und ich hatte in Bälde eine Baumreihe auf die Zeichenfläche gebracht, indem ich mehrere Instanzen dieser Baum-Klasse aufrief.
Die nächste Schwierigkeit bestand darin, die Instanzen in einer Schleife zu generieren und diese gleichzeitig in einem Array zu speichern. Mit etwas Recherche, logischem Denken und Probieren gelang auch dies.

Danach programmierte ich, dass sich die Baumreihen von hinten nach vorne aufbauen und per if-Abfrage geprüft wurde ob die Anzahl Bäume pro Reihe erreicht war, um dann das Koordinatensystem nach unten zu verschieben und somit das Zeichnen einer Baumreihe vor der anderen zu

ermöglichen. Dieser Prozess enthielt einige mir bis anhin unbekannte Dinge. Erstens wollte ich eine Abfrage generieren, die true zurückgibt, wenn die Anzahl gezeichnete Bäume einem Vielfachen einer von mir vorbestimmten Zahl entspricht und zweitens die Verschiebung des Koordinatensystems.
Das erste Problem konnte ich mithilfe von Suchmaschine und Foren lösen (%-Operator) und die zweite Angelegenheit war interessant zu erfahren. Jedoch war die pushMatrix-Sache auch für spätere Probleme verantwortlich. Mehr dazu weiter unten.

Im Arbeitsprozess kam ich noch in den Kontakt mit der PGraphic-Klasse, die, wie mir erklärt wurde, dazu da war, einen Teil des vom Code generierten visuellen Output, in einem Zwischenspeicher abzuspeichern und diesen wie eine Art eingebundenes Bild zu behandeln.
Ich wollte damit einen Teil des Bildes per Blur verschwimmen lassen, während der Rest unbetroffen blieb. Das erreichte ich zwar, jedoch war das Ergebnis nicht ansprechend und verlangsamte nur die Erstellung meiner vom Code vorgegeben Zeichnung.

Um den Wald weiter zu verfeinern, baute ich an einigen Stellen Instanzen einer weiteren Klasse ein. Diese war dafür verantwortlich, dass Kirschblüten-Bäumchen spriessten. Auch den Code dazu habe ich auf openprocessing.org gefunden. Zwar hatte ich zuerst noch Probleme damit, dass alle Elemente, die nach dem ersten Kirschblüten-Bäumchen erstellt wurden, pink und mit einer komischen Konturdicke erschienen. Ja, es wäre da schon hilfreich gewesen die Befehle pushStyle und popStyle zu kennen. Diese kreuzten jedoch erst später meinen Weg.

Als letztes Feature wollte ich dem Betrachter ermöglichen weiter und näher am Wald zu stehen. Dies wollte ich durch einen Zoom des Koordinatensystems erreichen.
Das ganze sollte an einen Slider gekoppelt sein, den ich mit der controlP5-Library erstellt hatte.
Doch dieser letzte Schritt liess mich Stunden verschwenden.

Durch meine allzu oft getätigten Matrix-Veränderungen kam auch mein Slider durcheinander und ein reibungsloses Funktionieren wurde verunmöglicht.
Bis ich jedoch herausfand, was das Problem verursachte und dass ich die controlP5 Slider korrekt erstellte, hatte ich versucht ca. vier Beispiele der controlP5 Controller von openprocessing zu analysieren, zwei verschiedene Versuchsdateien gebaut und zig mal die Erläuterungen zur Library gelesen.
Trotzdem wusste ich schlussendlich, wie ich controlP5 erfolgreich nutzen konnte, einzig die Anwendung bei meinem inzwischen recht unübersichtlichen Programm gelang mir nicht.

Fraktale, Satelliten, Smileys, Bilder und Vektoren

Je länger die Beschreibung zur Waldaufgabe ausfiel, umso kürzer ist hingegen der nächste Teil wieder.
Der Stoff beinhaltete Koordinatensystem-Manipulationen, Winkelfunktionen und Vektorgeometrie, die ich lediglich etwas auffrischen musste, das Einfügen von Bildern und Mappen von Zahlenreihen.

All dies war im Ausmass der Aufgaben gut nachvollziehbar, relativ leicht zu lösen und teilweise habe ich die Themen sogar schon zuvor angetroffen gehabt.

Ich stellte dabei fest, dass ich mich besser in eine Sache stürzen kann, wenn ich ein klares Ziel vor Augen habe. Das war beim ‹Probieren› mit dem zur Verfügung gestellten Fraktalfunktionen weniger der Fall.

Deshalb habe ich nach kurzem Probieren mehr Zeit ins Analysieren interessanter Codebeispiele von openprocessing.org und schliesslich in die Installation vom Android Software Development Kit und Integration derer in die Programmierumgebung Processing.
Dies gelang mir jedoch auch nur beschränkt, weil die Sache viel aufwändiger ist, als ich mir gedacht habe.

Deswegen freute ich mich auch auf die Endaufgabe, wo ich wieder mehr Potential erkennen konnte.

http://blogs.iad.zhdk.ch/coding-space/2013/09/20/satellite-face/

http://blogs.iad.zhdk.ch/coding-space/2013/09/24/lessons/

http://blogs.iad.zhdk.ch/coding-space/2013/09/24/planeten-2/

http://blogs.iad.zhdk.ch/coding-space/2013/09/24/vectory/

Finale Aufgabe: Pong und Verwandte

Als Abschlussaufgabe wurde uns der Code zum sehr alten Spiel ‹Break Out› zur Verfügung gestellt. Zuerst nahm ich mir die Zeit diesen Code zu analysieren mit dem Ziel jeden Teil des Codes zu verstehen.
Bei dieser Tätigkeit sah ich neue Möglichkeiten um an Probleme heranzugehen, die ich womöglich ohne diesen Code nicht hätte lösen können.

Mit diesem neuen Wissen und dem Code als Hilfe und Nachschlagewerk machte ich mich ans neue Projekt:
Ich wollte das Spiel ‹Space Impact›, das früher auf vielen Nokia Handys gespielt werden könnte, nachahmen und es mit dem ‹Break Out› verbinden.

http://blogs.iad.zhdk.ch/coding-space/2013/10/01/final-destination/

Ich hatte auch vor den Code von Beginn weg selbst zu schreiben und möglichst keine Abschnitte zu kopieren, da ich dadurch Probleme jeweils nochmals ganz anders durchdenken muss und mein Lernfortschritt noch etwas grösser ist.

Dynamische Arrays, oder eben ArrayLists, die meine Schüsse, die feindlichen Ufos und deren Position speichern, die Iteration solcher ArrayLists, Bewegungen des Schiffes per Tastatureingabe, Parallax-Bewegungen der verschiedenen Hintergrundebenen, nahtloses Aneinanderreihen von Bildern, Unterklassen, ‹public› Funktionen, Abspielen von Sound Samples und Tracks per minim Library, Timer und Tweening sind nur einige Themen die mich während dieser Aufgabe kurz oder eben auch länger beschäftigten.

Durch meine klaren Ziele, die ich von Beginn weg erreichen wollte, und deren zielstrebigen Verfolgung musste ich mich unvermeidlich mit oben genannten Themen auseinandersetzen. Trotz Problemen fand ich bei Allem einen Weg, um das von mir Geforderte zu erreichen.

Schlussendlich bin ich mit dem Produkt mehr oder weniger zufrieden, da ich die Features, die ich darin enthalten haben wollte, einbauen konnte. Ebenfalls festhalten möchte ich, dass das Projekt mich dazu bewegte, sehr viel Neues zu sehen und einzubauen. Demzufolge habe ich damit auch sehr viel gelernt. Für komplexere Ausführungen oder gar ganz neue Ansätze, wäre ich wahrscheinlich nun bereit.
Zum Zeitpunkt vor dem Projekt, denke ich, dass ich mein Können relativ gut ausschöpfen konnte ohne mich dabei zu übernehmen.
Deshalb bin ich auch mit meiner Selbsteinschätzung vor der Ausführung des Projektes zufrieden.

Gleichzeitig möchte ich aber anfügen, dass ich auch das Potential für Weiterführendes klar erkenne und auf jeden Fall motiviert bin, das mächtige Werkzeug ‹Programmieren› beim Erreichen ganz anderer Projekte einzusetzen.
Was ich hingegen bei zukünftigen Projekten stark verbessern muss, ist das Beibehalten von Übersicht und Struktur. Trotz dem Versuch diesen Vorsatz möglichst zu erfüllen, war mir dies nicht hundertprozentig möglich.

Aber beim nächsten Mal, werde ich wohl noch einmal ganz anders an ein solches Problem herangehen.