PROCESSING

Dokumentation

Programming Basics

Alexander Häberlin

Kurs BDE-VIAD-V-1040 Programming Basics
Dozent - Max Rheiner
17.09.2013 bis 27.09.2013

Inhaltsverzeichnis

1. Intro

Diese Dokumentation dient nicht nur als Zusammenfassung der letzten zwei Wochen, aber auch als Nachschlagewerk für mich, daher wird sie relativ detailliert sein. Ich werde oft nur ein Beispiel für eine Referenz oder Befehl nennen um die Anwendung zu erläutern, alles weitere kann auf der Processing Website nachgeschaut werden.
Bis an hin konnte ich leider noch keine Erfahrung mit Processing oder Ähnlichem machen, jedoch bin ich positiv überrascht über die Vielfalt der Möglichkeiten welche einem dargelegt werden, wenn man die Basics des Processing versteht. Ich will mich in Zukunft weiter in dieses Programm einarbeiten und dieses Nachschlagewerk auch stetig erweitern.

Alle Wörter welche in der folgenden Zusammenfassung mit einem * versehen sind, werden am Schluss der Dokumentation in der Bibliothek genauer beschrieben und erläutert.

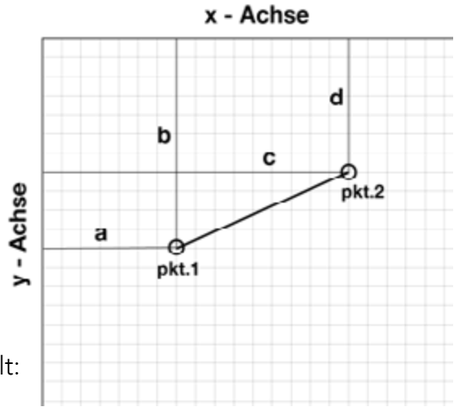
Processing ist eine simple Version von Java, welches einem eine integrierte Entwicklerumgebung bietet und das erstellen von Grafiken, Animationen und Simulationen stark vereinfacht.

2. Basics

Grundsätzlich definiert man, in den meisten Fällen, als erstes die Grösse und die Hintergrundfarbe des Fensters in dem der Code dargestellt werden sollte. Dies kann mit dem Befehl „size()“* und „background()*“ festgelegt werden.
Um dann eine Zeichenausgabe zu befehlen gibt es auf der Processing Website eine Liste mit vielen nützlichen Abkürzungen welche einem das Codieren wesentlich vereinfachen.
Der Spielraum welcher einem zur Verfügung steht besteht immer aus einem Koordinatensystem mit einer x- und einer y-Achse.
Hier ein kleines Beispiel, wie man eine Linie zeichnen kann:
(Das Zeichen „ **;** “ muss nach jedem Befehl gesetzt werden, um dem Computer zu sagen, dass er auf dieser Zeile nicht mehr weiter schauen muss.)

line(a,b,c,d);

Die Buchstaben stehen für die folgenden Standorte:



a = x-Achse Pkt.1
b = y-Achse Pkt.1
c = x-Achse Pkt.2
d = y-Achse Pkt.2

Bei einer Ellipse wären die Punkte folgendermassen verteilt:

ellipse(a,b,c,d)

a = x-Achse vom Mittelpunkt
b = y-Achse vom Mittelpunkt
c = Breite der Ellipse
d = Höhe der Ellipse

Mit dieser Art kann man beliebig viele Formen in das Koordinatensystem setzten. Jedoch sind die Formen immer leer, also ohne Farben. Die Füllfarbe des Objektes kann mit dem Befehl „fill(r,g,b,t)*“ definiert werden. Diese wird vor der Form platziert und mit dem RGB Farbschema (r,g,b) und der Transparenz (t) beschrieben.
Die Stärke und Farbe der Konturen werden im Setup-Bereich* mit dem Befehl „stroke()*“ (Farbe) und „strokeWeight()*“ (Stärke) gesetzt.

3. Variablen

Man kann jedoch auch Farben, Stärken oder Längen mit einem Platzhalter, einer sogenannten Variablen festlegen. Diese Variable können am Anfang (globale Variable), oder in einer Funktion (lokale Variable) definiert werden, um dann mit einem Platzhalter (meinst ein Buchstabe) eingesetzt zu werden.

Hier ein Beispiel:

Ich will eine Länge öfters in verschiedenen Figuren nutzen und definiere daher eine Variable.

int laenge1 = 10;
int laenge2 = 5;

ellipse(x-position, y-position, laenge1, laenge 2)

„int“ legt fest, dass es sich um eine ganze Zahl handelt. Wenn Dezimalzahlen eingesetzt werden, müsste die Variabel-Bezeichnung „float*“ eingesetzt werden.

Somit ergibt sich eine Ellipse mit der Breite 10 und der Höhe 5. Diese Variablen eignen sich besonders wenn man beispielsweise mit dem „random*“ Befehl arbeiten will. Praktisch ist auch, wenn die Grössen von mehreren Figuren geändert werden müssen, muss lediglich die Variable geändert werden und alle Figuren mit dieser Variable werden geändert.
Es gibt noch viele weitere Bezeichner für Variablen mit verschiedenen Funktionen. (Siehe Bibliothek).

4. Funktionen

Eine Funktion dient zur Vereinfachung des Programmieren und um gewisse Schritte zu bündeln. Zwei Schritte gehören zu einer Funktion; Zum einen gibt es die Funktion-Deklaration, welche bestimmt wie die Funktion genannt wird, was sie macht und was sie beinhaltet. Zum anderen gibt es den Funktions-Aufruf, dort wird dann die vorgegebene Funktion abgerufen.

Hier ein Beispiel:

Ich will einen Kreis definieren um es später einfach abrufen zu können.

```
void Ball()              //void()* definiert, dass „Ball“ kein Rückgabewert erzeugen soll.
{
ellipse(x-achse,y-achse,50,50);
fill(0);
}

// Diese Funktion kann dann z.B in einem Zeichenbefehl eingesetzt werden.

void draw()
{
Ball();
}
```

Somit wird ein Kreis am definierten Ort gezeichnet und muss nicht immer wieder abgetippt werden.

5. Interaktiv

Sobald man ein Programm interaktiv oder animiert machen will, müssen gewisse Programmevents als Ereignisse welche abgespielt werden sollten festgelegt werden.
Processing erkennt diese Ereignisse wenn sie mit ihrem festgelegten Namen beschrieben werden; Bsp:

„setup*“ --> Programmstart
„draw*“ --> Zeichenausgabe
„keyPressed*“ --> Tastatureingabe
usw..

Hier z.B einen gesamten Code. Hier folgt der Kreis dem Mauszeiger.

```
void setup()
{
  size(600,600);                    // Grösse des Bildes
  background(255);                  //Hintergrundfarbe
  stroke(0);                        //Strichfarbe
  strokeWeight(5);                  //Strichdicke
}
void kreis()                        //ich benenne mein Objekt „kreis“
{
background(255);
//Bei jedem Frame wird ein weisser Hintergrund gezeichnet, somit sieht der Kreis animiert aus und
man sieht die Zeichnung des letzten Frames nicht.

ellipse(mouseX,mouseY,50,50);             //zeichne ellipse(x-position Maus,y-position Maus, breite
                                          50, höhe 50) }
void draw() {                       //Funktion „zeichne“
kreis(); }
```

Processing „spielt“ den gesamten Code 60 mal pro Sekunde ab, dass bedeutet wenn eine Koordinate nicht fix sondern veränderbar ist wie z.B die mouseposition X, wird das Programm animiert. Oft wird auch eine Variabel für eine Koordinate eingesetzt welche sich selber nach jedem Frame vergrössert. Bsp: x = x+1 -> Somit wird x nach jedem Frame um 1 grösser.
Es können auch Kontrollstrukturen eingesetzt werden in welcher definiert wir wie oft ein gewisser Block wiederholt werden sollte. Hierfür wird meistens die „for-schleife*“ genutzt:

```
for([Variablen Init.],[Bedingung],[Fortsetzung])
{}
Bsp.:
for(int i=0; i > 100; i++)
{}
--> (int i=0)    = „i“ ist am Anfang = 0
--> (i > 100)    = solange „i“ kleiner als 100 ist, dann..
--> i++          = addiere “i“ bei jedem Frame mit 1
```

6. Bedingungen

Es kann definiert werden, dass gewisse Aktionen nur ausgeführt werden sollten wenn festgelegte Bedingungen erfüllt sind.
Diese werden meistens mit der „if /else*“ Funktion definiert.

Hier ein Beispiel: (Ich möchte das Processing mir sagt ob die Zahl über 10 ist oder nicht)

```
int x = 9;

if(x == 10)                    // wenn x=10 ist, dann..
{
println(“x ist gleich 10“);    // „println*“ druckt im unteren Fensterbereich „x ist gleich 10“
}
else                           // wenn x ungleich 10, dann
{
println(“x ist nicht gleich 10“);
}
```

Wenn man mehr als zwei Möglichkeiten festlegen will, kann man einfach anstatt „else“, „else if“ einsetzen um noch weitere Möglichkeiten durch-testen zu lassen. Beim letzten muss dann einfach wieder „else“ geschrieben werden damit Processing weis, dass nicht weiter geprüft werden muss. Ähnlich ist auch der Befehl „switch*“, es ermöglicht auch mehrere Antworten:

```
int x=10;
switch(x)
{
case 10:
  println(„x ist gleich 10“);
  break;
case 9:
  println(„x ist gleich 9“);
  break;
case 8:
  println(„x ist gleich 8“);
  break;
default:
  println(„x ist nicht 10,9 oder 8“);
  break;
}
```

7. Bilder

Zu beachten ist:

PNG = Das File wird als Bild abgespeichert mit all seinen Eigenschaften, einfach als Bild.

Jpeg = Reduziert das Bild auf das wesentliche und daher gehen relativ viele Eigenschaften verloren. (Dateigrösse wird klein)

SVG = Speichert das Bild mit allen Eigenschaften und Vektoren. (Dateigrösse wird gross)

Ich möchte eine Vektordatei in mein Programm rein-laden. Dafür muss ich zuerst meine Dateien benennen, z.B: PShape* baum;

Nun ist die Form als „baum“ benennt. Jetzt muss das Bild noch definiert werden; (Wichtig: Diese Datei muss sich zwingend im gleichen Ordner befinden wie der Code)

```
baum = loadShape(„baum.svg“);
```

Jetzt ist die Form definiert. Nun kann sie z.B in der draw Funktion gezeichnet werden:

```
void draw()
{
shape(baum, a, b, c, d);
}
```

a = x-kordinate
b = y- Koordinate
c = breite des Bildes
d = Höhe des Bildes

Wenn man jedoch mehrere Bilder laden möchten, erstellt man am besten eine „array*“. Wenn ich nur Bilder habe welche ich platzieren möchten, dann nehme ich die array Bezeichnung „PImage[]“, bei Vektordateien (svg) nehme ich die Bezeichnung „PShape[]“

8. Vektoren

PVector ist ein Punkt im Koordinatensystem welches mit einem Schrank verglichen werden kann. Am Anfang des Codes wird definiert, dass eine neue Schublade erstellt werden soll;

```
PVector  p1;                    // neue Schublade namens „p1“ welche Vektordateien
                                beinhalten kann.

p1 = new PVector(40, 20);       // in der Schublade ist ein Vektorpunkt. (X-koor. = 40, Y-ko
                                or.20)
```

Wenn ich den Punkt aus dieser Schublade anwenden will, muss ich den Namen der Schublade nennen und welcher Wert ich möchte: z.B

```
Ellipse(p1.x, p1.y, 12, 12);
```

Mit „p1.x“ sag ich, dass ich Schublade p1 möchte und die Koor. x ,also 40 möchte.
Die Vektorschublade „p1“ kann am Anfang auch noch keinen Inhalt haben, dieser kann später mit der .add* Funktion hinzugefügt werden. Es gibt noch unzählige Optionen wie Punkte hinzugefügt, entfernt oder multipliziert werden können.

9. Sound

Um Geräusche und Soundtracks bei einer Animation oder Spiel in Processing zu implementieren, muss die Minim Library heruntergeladen werden und in den Processing Ordner gestellt werden. Dies erlaubt es einem mp3, wav und weitere Audiodateien in das Processing zu laden. Es funktioniert grundsätzlich ähnlich wie mit dem upload von Bildern und Vektordateien.
Zuerst muss die Library von Minim importiert werden:

```
import ddf.minim.*;
```

Danach muss der Audiotyp definiert und benannt werden:

```
Minim minim;
AudioPlayer soundtrack;     // „AudioPlayer“ benutzt man mehr für ein Lied welches im
                            Hintergrund läuft und relativ gross ist.
AudioSample shoot;          // „AudioSample“ wird genutzt für kurze Audiodateien wie z.B
                            ein Schuss.
```

Dann muss die Dateien ins Processing geladen werden. Dies wird direkt im Setup gemacht:

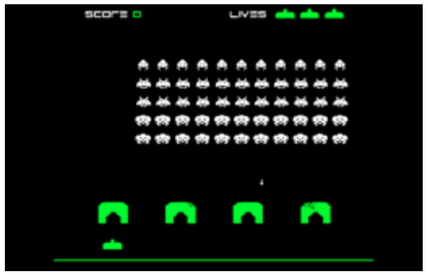
```
 shoot  = minim.loadSample(„shoot.wav“);

soundtrack = minim.loadFile(„soundtrack.mp3“);
soundtrack.play();                  // Den Soundtrack kann auch direkt mit
soundtrack.loop();                  dem „play();“ Befehl gestartet werden, da es immer lau
                                    fen soll solange das Fenster geöffnet ist.
```

Audiodateien welche erst nach einem gewissen Ereignis abgespielt werden sollten, können mit dem „.trigger();“ Befehlt gestartet werden.

10. Endprojekt

Für mein Endprojekt wollte ich mich zuerst langsam an den vorgegebenen Code herantasten. Meine Grundidee hatte ich schon von Anfang an im Kopf. Ich wollte eine abgeänderte Version von Spaceinvaders kreieren. Jedoch in meiner Version würden die Aliens die Menschen angreifen und nicht umgekehrt. Das ganze habe ich „Alien Invasion“ getauft. Hier ein Beispiel vom Klassischem Spaceinvader Spiel:



Wichtig war mir die Gameerfahrung welche dem Spieler damals geboten wurde, in mein Spiel zu implementieren. Das heisst übertriebene Themenmusik, laute Abschussgeräusche, einfache Gamesteuerung und verpixelte Aliens.

10.1 Bewegung der Schiffe

Angefangen habe ich mit der Änderung der „bricks“, die sollten Raumschiffe werden welche sich von links nach rechts bewegen. Das Aussehen des Schiffes habe ich auch von Spaceinvaders kopiert und ins Processing geladen. Damit die Schiffe sich bewegen, habe ich zu Ihrer X-Position einen zufällig steigender Faktor multipliziert, damit sich nicht alle Schiffe gleich schnell bewegen. Schiffe welche noch alle Leben haben (mehr zu den Leben im nächsten Abschnitt) erscheinen wieder am linken Bildrand wenn sie den Rechten überschreiten. Wenn sie jedoch einmal getroffen werden, beginnen sie langsam in Richtung unterer Bildrand zu sinken, beim zweiten Treffer wird die Fallgeschwindigkeit noch erhöht. Wenn sie den unteren Bildrand überschreiten sind sie „besiegt“ und tauchen nicht mehr im Spiel auf.

10.2 Leben der Schiffe

Die Raumschiffe können auf zwei Arten besiegt werden; Entweder sie werden drei mal getroffen und sind somit ganz zerstört oder sie fallen unter den unteren Bildrand. Dies kann aber nur nach zwei Treffern passieren.
Wie schon oben erwähnt wird bei jedem Treffer dem Schiff ein Leben abgezogen. Dies zeigt sich in der fallenden Richtung, wie auch im beschädigten Aussehen des Schiffes. Jedes Schiff hat am Anfang des Spieles drei Leben. In den Funktion „updateScene“ habe ich festgelegt, wenn ein Schiff mehr als null Leben hat, wird ihm bei einem Treffer eines abgezogen. Jedoch wenn seine Leben auf Null fallen ist die „BrickList[i]._life = 0“ und somit ist das Schiff zerstört und aus dem Spiel.

10.3 Aliens

In meinem Spiel sind die Angreifer wie schon erwähnt die Aliens. Die Schiffe werden vom „Mutter-Alien“ mit Hilfe von kleineren Aliens abgeschossen. Die kleinen Aliens verhalten sich wie ein Bouncingball, das heisst sie prallen von den Wänden wie auch von der Mutter ab. Falls erwünscht kann der kleine Alien mit einem Mausklick wieder zu der Mutter gezogen werden, um sie dann wieder „abzuschiessen“. Die Mutter lässt sich nur in der X-Koordinate bewegen.

Raumschiff mit 3 Leben

Raumschiff mit 2 Leben

Raumschiff mit 1 Leben

Mutter-Alien

Alien

10.4 Nächste Schritte

Was ich gerne noch ins Spiel integrieren möchte, wäre wie im ersten Beispiel vom Original Space invaders, dass die Leben vom Spieler sowie ein Punktestand angezeigt wird. Cool wäre natürlich auch mehrere Levels zu erstellen um die Spielerfahrung noch spannender zu gestalten.

10.5 Code des Spiels

Der gesamte Code meines Spiels kann unter
http://blogs.iad.zhdk.ch/codingspace/2013/10/01/code-endprojekt-alien-invasion/
angesehen werden.

12. Bibliothek

Ich werde hier nur ein paar Basic Befehle auflisten, eine detailliertere Ausführung ist unter folgendem Link zu finden: Processing language

size();	Definiert die Grösse des Bildschirms. Die angegebene Grössen kann später width und height abgerufen werden.
int();	Definiert z.B Buchstaben als ganze Zahlen ohne Dezimalstelle.
float();	Definiert z.B Buchstabe als Zahl mit Dezimalstellen.
background();	Legt die Hintergrundfarbe fest. (RGB Farben).
fill();	Wird dieser Befehl vor einem Objekt gesetzt, kann die Füllfarbe definiert werden. Mit dem Befehl noFill() kann ein Objekt ohne Farbe gesetzt werden.
stroke();	Legt die Farbe der Konturen fest.
strokeWeight();	Definiert die Stärke der Konturen.
void();	Legt den Namen einer Funktion fest und sagt das sie keinen Rückgabewert generieren sollte. Hier ein paar Beispiele was auf den Befehlt folgen könnte: setup() = Hier werden die Grundinfos des Programmes festgelegt. draw() = Was im Programm gezeichnet werden sollte. keyPressed() = Was folgen sollte wenn eine Taste gedrückt wurde.
println();	Zeigt das Ergebnis des Codes im unteren Balken an.
if() / else()	Eine einfache wenn/dann Funktion.
case();	Ähnlich wie if/else jedoch mit mehreren „wenn“ Möglichkeiten.
PShape();	Definition von einem Vektor. Für Bilder wird PImage(); benutzt.
array	Verhaltet sich wie eine Schubladen in welcher man mehrere Informationen stecken kann um sie später wieder aufzurufen. Details hier: http://processing.org/reference/Array.html
for();	Kann eine Abfolgung von Wiederholungen kontrollieren. Aufbau: for (init; test ; update) { „was wiederholt werden soll“}
Random();	Wählt auf zufälligerweise eine Zahlen zwischen zwei definierten.