S E M I N A R A R B E I T

im Seminar
„Softwareentwicklung von Computerspielen“
(Leitfach Informatik)

Drum-Machine

Verfasser: Caspar Raap

Seminarleiter: OstR Seiler

Abgabetermin: 7.11.2017

Erzielte Punkte schriftlich: ..... in Worten: .....
(0 – 15, einfache Wertung)

Erzielte Punkte mündlich: ..... in Worten: .....
Präsentation / Prüfungsgespräch
(0 – 15, einfache Wertung)

.....
Unterschrift des/der Seminarleiter/in

Inhaltsverzeichnis

1. Extreme Programming

1.1. Allgemein

Extreme Programming ist eine agile Methode der Softwareentwicklung.[1] Das Ziel von agilen Methoden ist es, eine kooperative, flexible Art der Softwareentwicklung in die Tat umzusetzen.[2] Agile Methoden hängen stark von den sie anwendenden Teams ab, so schreibt Don Wells: „*The most important thing to know about Agile methods or processes is that there is no such thing. There are only Agile teams.*“[3] Auch Extreme Programming setzt als eine agile Methode einen Schwerpunkt auf Teamwork. Das Team sollte sich, um möglichst effizient zu sein, selbst organisieren.[4]

Extreme Programming beinhaltet 5 Grundwerte[5]:

- Einfachheit: Das Team setzt alle nötigen und alle angefragten Features um, aber nicht mehr.
- Kommunikation: Jeder ist Teil des Teams und es wird an allen Teilen des Projektes zusammen gearbeitet. Deswegen ist regelmäßige Kommunikation zwischen den Teammitgliedern essentiell.
- Rückmeldung: Es wird darauf geachtet, die Software früh und häufig vorzuführen und auf die daraus resultierenden Rückmeldungen zu reagieren.
- Respekt: Gegenseitiger Respekt ist eine der Grundlagen für eine funktionierende und ertragreiche Zusammenarbeit.
- Mut: Es wird die Wahrheit über den Fortschritt und Einschätzungen gesagt. Es wird davon ausgegangen, dass das Projekt gelingt. Man braucht vor nichts Angst zu haben, da niemand alleine arbeitet. Es wird sich an Veränderungen angepasst, sobald sie eintreten.

Ein Extreme Programming Projekt beginnt bei den „User-Stories“. In den User-Stories beschreiben die Nutzer auf „Storycards“ in jeweils ungefähr drei Sätzen Dinge, die die Software für sie beinhalten muss. Im Zuge dessen werden gleichzeitig die „Acceptance-Tests“ erstellt, mit denen man überprüfen kann, ob die Storycards erfüllt wurden.[6] Als Nächstes wird beim „Release Planning“ ein Release Plan erstellt, in dem die Grundzüge des Projektes dargelegt werden. Der wichtigste Teil des Release Planning ist es, die Zeit, die man für die Umsetzung der verschiedenen Storycards braucht, einzuschätzen und diese daraufhin in eine Reihenfolge zur Bearbeitung in Abhängigkeit der Wichtigkeit und der benötigten Zeit einzuordnen.[7] Ein wichtiger Einfluss dabei ist die “Projekt Velocity“[8], welche angibt, wie viele

1 Wells, Don : http://www.extremeprogramming.org
2 Wells, Don : http://www.agile-process.org
3 Wells, Don : http://www.agile-process.org
4 Wells, Don : http://www.extremeprogramming.org
5 Wells, Don : http://www.extremeprogramming.org/values.html
6 Wells, Don : http://www.extremeprogramming.org/rules/userstories.html
7 Wells, Don : http://www.extremeprogramming.org/rules/planninggame.html
8 Wells, Don : http://www.extremeprogramming.org/rules/velocity.html

Storycards in einer bestimmten Zeit implementiert werden können. Als Nächstes wird mit der Arbeit begonnen. Alle ein bis drei Wochen wird eine neue Iteration der Software mit den neuesten Errungenschaften herausgegeben, die daraufhin mithilfe der Acceptance-Tests auf Funktionalität überprüft wird. Die Rückmeldungen, die von den Nutzern kommen, werden bei der nächsten Iteration mit in den Entwicklungsprozess einbezogen. Versionen, die die Zustimmung der Nutzer bekommen, werden in kleinen Releases veröffentlicht.[9] Ein wichtiger Kniff in der technischen Entwicklung ist, dass die „Unit-Tests“ - Code der prüft, ob ein Softwareteil das tut, was es soll - vor dem eigentlichen Code programmiert werden. Dies ermöglicht, den simpelsten Code zu schreiben, der die gewünschten Aufgaben erfüllt.[10]

1.2. In diesem Projekt genutzte Elemente

In meinem Projekt habe ich nicht alle Elemente des Extreme Programming eingesetzt, da dies wegen den Dimensionen meines Projektes unpraktikabel gewesen wäre (siehe *5.Bewertung von Extreme Programming / Fazit*). Ich beschränkte mich deshalb auf einige wenige, die auf die Größe meines Projektes passen.

Umgesetzt wurden:

- Die 5 Grundwerte:
  - Ich habe den Code darauf beschränkt, nur das zu tun, was zur Erfüllung der Storycards notwendig ist.
  - Ich habe mich mit so vielen Leuten wie möglich konstruktiv über mein Projekt ausgetauscht.
  - Ich habe die Rückmeldungen der Testnutzer und der Kommunikationspartner ernst genommen und berücksichtigt.
  - Ich habe Mitwirkende (Testnutzer usw.) respektiert, egal, wie viel Ahnung sie von der Materie haben beziehungsweise zu diesem Zeitpunkt hatten.
  - Ich habe nicht gezögert, mir für dieses Projekt eventuell nötiges Zusatzwissen anzueignen und schwierige Probleme oder Features offen anzugehen.
- Das Grundmodell der Planung inklusive Storycards in Verbindung mit der Entwicklung und des seperaten Testens einzelner Iterationen

9 Wells, Don : http://www.extremeprogramming.org/map/project.html
10 Wells, Don : http://www.extremeprogramming.org/rules/testfirst.html

2. Mein Thema

2.1. Allgemeine Informationen

Mein Projekt ist ein virtuelles Instrument mit dem Namen „Drum-Machine“, in dem man mithilfe von vier Klängen eines Schlagzeuges Rhythmen frei erstellen kann.

2.2. Storycards

Storycard	**Testsituation**
Die Software soll vier Klänge (eines Schlagzeuges) besitzen.	Es gibt vier Schlagzeuggeräusche.
Das Abspielen der einzelnen Töne soll programmierbar sein.	Man kann frei bestimmen, wann ein Geräusch im Takt abgespielt wird.
Die Länge des programmierbaren Bereiches soll zwei Takte betragen.	Es gibt zwei Takte, in denen man die Geräusche setzen kann.
Jeder Takt soll in acht Noten aufgeteilt sein (Achtelnoten) die einzeln ansteuerbar sind.	In jedem Takt gibt es acht voneinander gleich entfernte Stellen an denen man Geräusche platzieren kann.
Die Anzahl der Viertelnoten pro Minute (”BPM“) soll frei wählbar sein.	Es wird die angegebene Anzahl an Viertelnoten pro Minute in gleichmäßigem Abstand gespielt.
Das Abspielen der Töne soll pausierbar sein.	Das Abspielen der Töne kann gestoppt und wieder gestartet werden.

Im Umsetzungsprozess ist in einer Testphase offenkundig geworden, dass ein Feature praktisch wäre, welches aber noch nicht in den Storycards vorhanden war. Daraufhin wurde eine weitere Storycard formuliert und implementiert.

Die Stelle im Takt, die gerade zu hören ist soll erkennbar sein	Es gibt eine Anzeige, die anzeigt, wo im Takt man sich momentan befindet

2.3. Verortung in der Literatur

Man kann „Drum-Machine“ als einen Softwaresynthesizer einordnen, da es ein Programm zur Erzeugung von Ton ist.[11] Spezifischer ist das Programm als ein Software Sampler zu betrachten, also eine Software zur Emulation der Funktionen eines Samplers.[12] Ein Sampler ist ein Instrument, welches aufgenommene Töne abspielen kann.[13] Diese können in einer Reihenfolge programmiert und anschließend abgespielt werden.[14] Des Weiteren ist das Programm wie die meisten Sampler polyphon.[15] [16]

11 https://en.wikipedia.org/wiki/Software_synthesizer
12 https://en.wikipedia.org/wiki/Software_sampler
13 https://en.wikipedia.org/wiki/Sampler_(musical_instrument)
14 https://en.wikipedia.org/wiki/Sampler_(musical_instrument)
15 https://en.wikipedia.org/wiki/Sampler_(musical_instrument)
16 https://de.wikipedia.org/wiki/Polyphonie_(Elektrophon)

3. Storycards

3.1. Storycard „BPM“

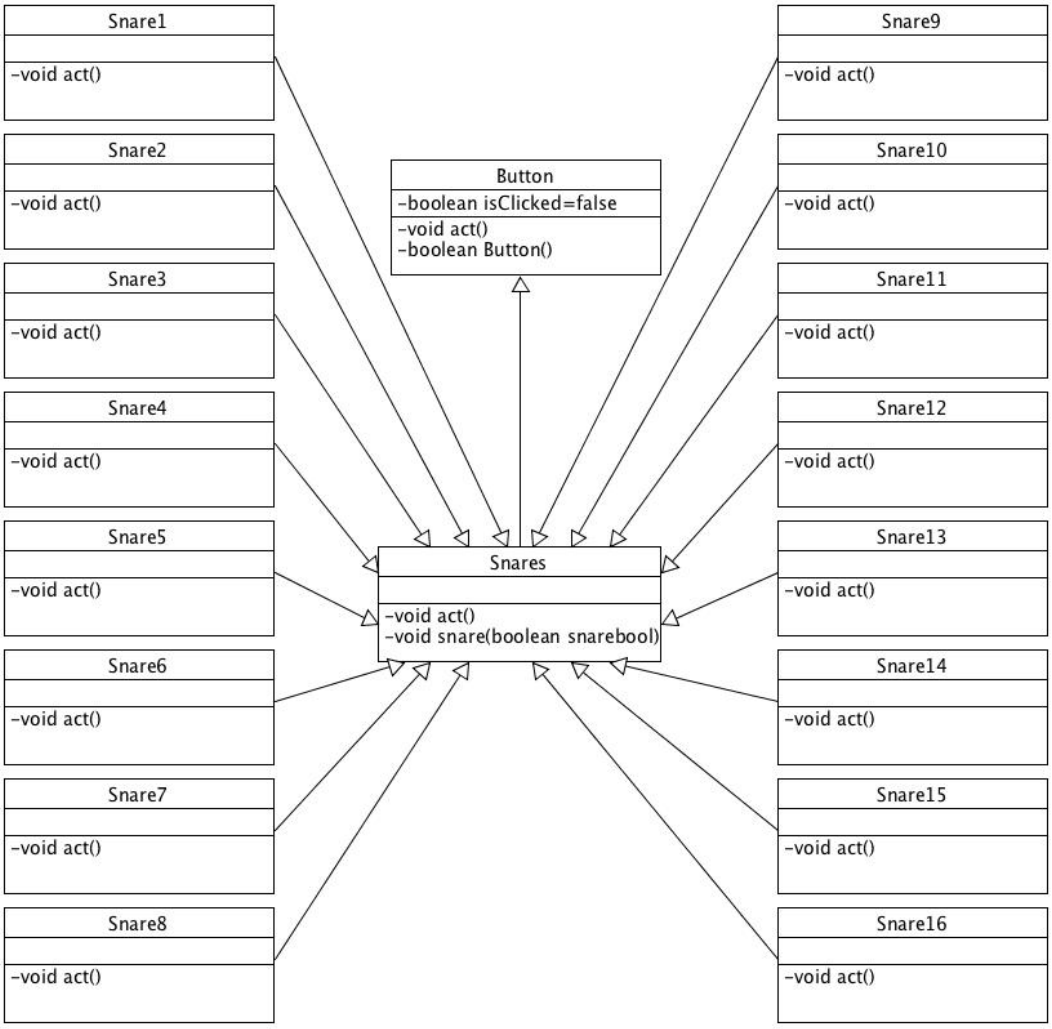
a) Beschreibung

Die einzelnen Achtelnoten werden in gleichem Abstand voneinander abgespielt. Dieser Abstand ist so gewählt, dass eine vom Nutzer veränderbare Anzahl an Viertelnoten pro Minute (auch bekannt als „BPM“ also „Beats Per Minute“) abgespielt wird.

b) Implementierung

Grundsätzlich haben wir vier Instrumente, die durch jeweils eine Klasse modelliert werden. Von diesen vier Klassen hat jede 16 Unterklassen, die je eine Achtelnote ihres „Instrumentes“ modellieren.

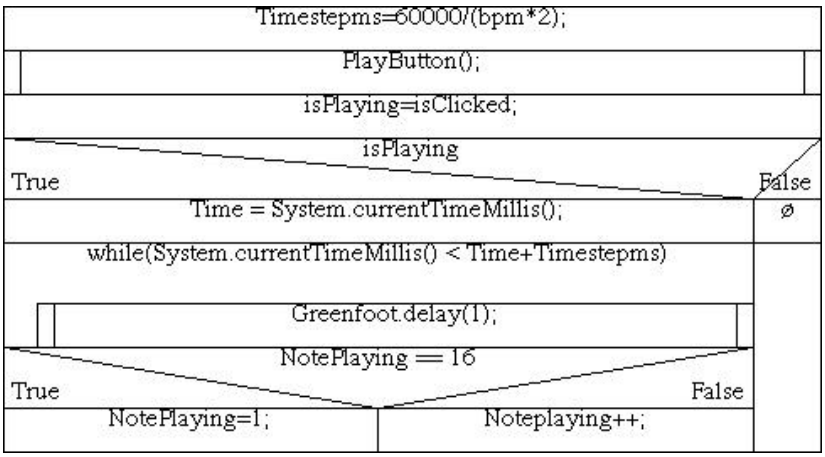
Dazu hier ein Klassendiagramm des Instrumentes „Snaredrum“:



Jede Note jedes Instrumentes fragt dieselbe Variable ab, die die aktuelle Position im Takt angibt. Diese Variable wird in besagten regelmäßigen Abständen inkrementiert

beziehungsweise auf den Anfangswert zurückgesetzt. Dieselbe Klasse, die diese Aufgabe übernimmt, rechnet auch die Abstände aus und bemisst diese. Im Grunde fragt die Klasse am Anfang eines Intervalls die aktuelle Systemzeit in Millisekunden ab, wartet einen kleinen Moment, fragt erneut die Systemzeit ab, berechnet die Zeit die seit dem ersten Zeitpunkt vergangen ist, vergleicht diesen Wert mit der Anzahl an Millisekunden, die zwischen zwei aufeinanderfolgenden Achtelnoten liegen muss und wiederholt diesen Vorgang solange, bis die Zeit, die seit dem ersten Zeitpunkt vergangen ist, der Zeit gleicht, die zwischen zwei aufeinanderfolgenden Achtelnoten liegen muss, oder größer als jene ist.

Zum besseren Verständnis hier ein Struktogramm der „act()“ Methode der Klasse „Taktgeber“:

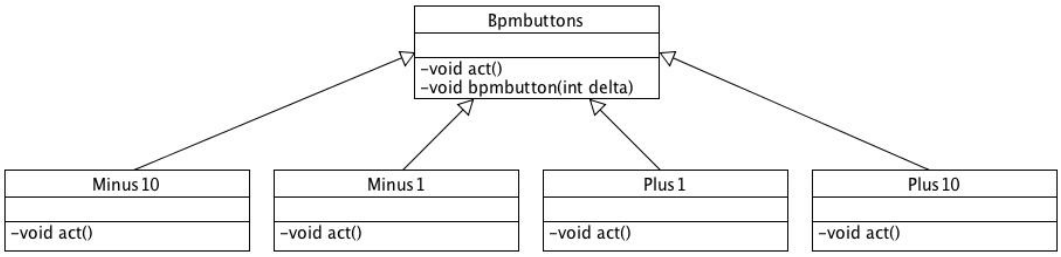


Quellcode Anlage 1: die act() Methode der Klasse „Taktgeber“

Da der Abstand zwischen zwei aufeinanderfolgenden Achtelnoten nur die Hälfte des Abstandes zwischen zwei aufeinanderfolgenden Viertelnoten beträgt, wird er mit der Anzahl an Millisekunden pro Minute (60000), geteilt durch die Anzahl an Viertelnoten pro Minute, mal zwei, berechnet.

Die Variable „bpm“ ist eine Variable der Art „public static int“, das heißt, dass man von anderen Klassen auf sie zugreifen und sie verändern kann. Der Nutzer soll die BPM frei wählen können. Für die Implementation dieses Features sind die Klasse „Bpmindi“ sowie die Klasse „Bpmbuttons“ mit ihren Unterklassen zuständig. Die Klasse „Bpmindi“ zeigt die aktuelle Anzahl an BPM an, denn um etwas frei wählen zu können muss man es kennen. Sie greift auf die statische Variable „bpm“ zu und zeigt mithilfe einer Greenfoot eigenen Funktion diese auf dem Bildschirm an. Die Unterklassen von „Bpmbuttons“ sind eine Reihe von Bedienelementen, welche auf die Variable „bpm“ zugreifen und diese um jeweils den Wert eins beziehungsweise zehn steigern beziehungsweise verringern.

Hierzu ein Klassendiagramm der Klasse „Bpmbuttons“:



c) Quellcode

Quellcode Anlage 1:

```
Timestepms=60000/(bpm*2);
PlayButton();
isPlaying=isClicked;
if(isPlaying){
   long Time  = System.currentTimeMillis();
   while(System.currentTimeMillis() < Time+Timestepms){
      Greenfoot.delay(1);
   }
   if(NotePlaying==16){
      NotePlaying=1;
   }
   else{
      NotePlaying++;
   }
```

3.2. Storycard „Indikator“

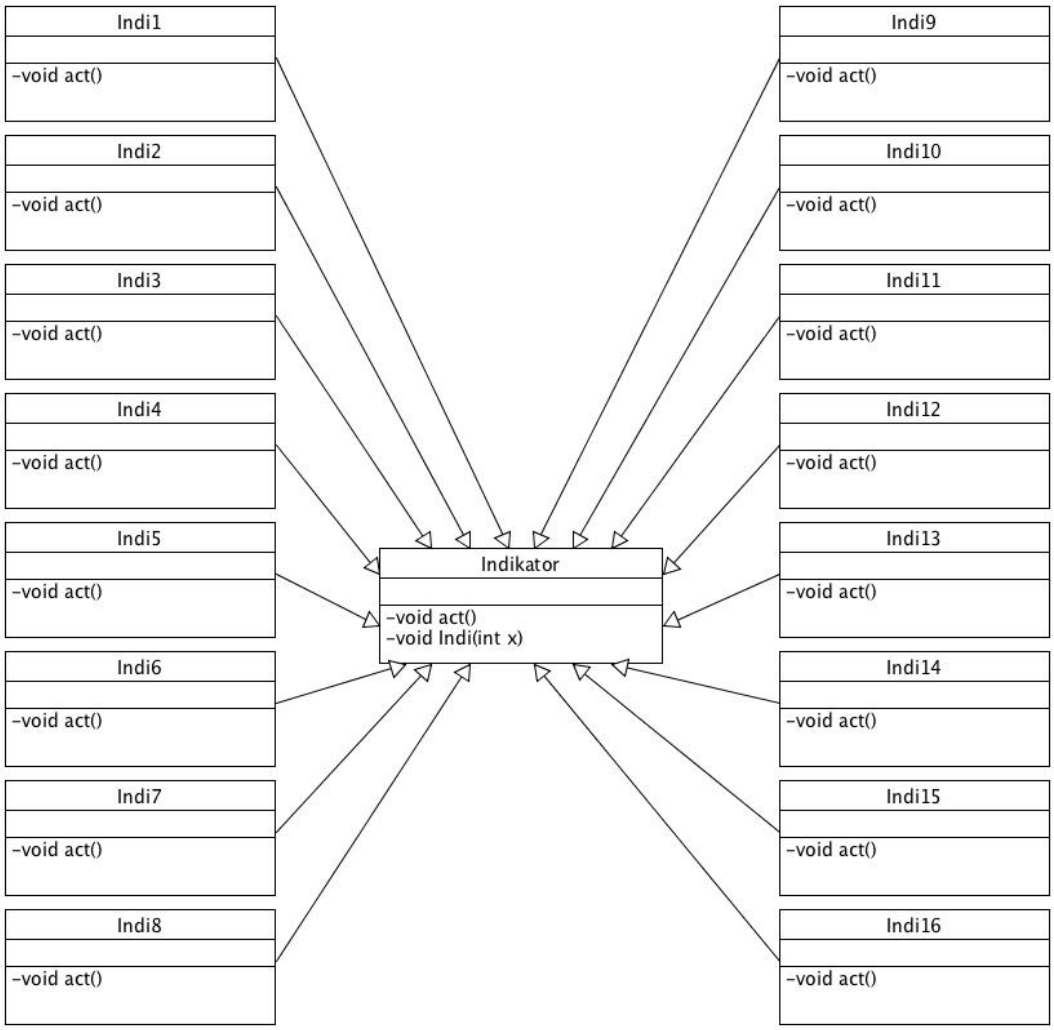
a) Beschreibung

Eine Anzeige, welche die aktuelle Abspielposition darstellt.

b) Implementierung

Ähnlich wie bei den Instrumenten ist hier ein Klasse mit 16 Unterklassen, für jede Achtelnote eine, vorhanden, welche die Variable „NotePlaying“ abfragen und an genau einer Stelle in den zwei Takten ihre Aktion(en) durchführen.

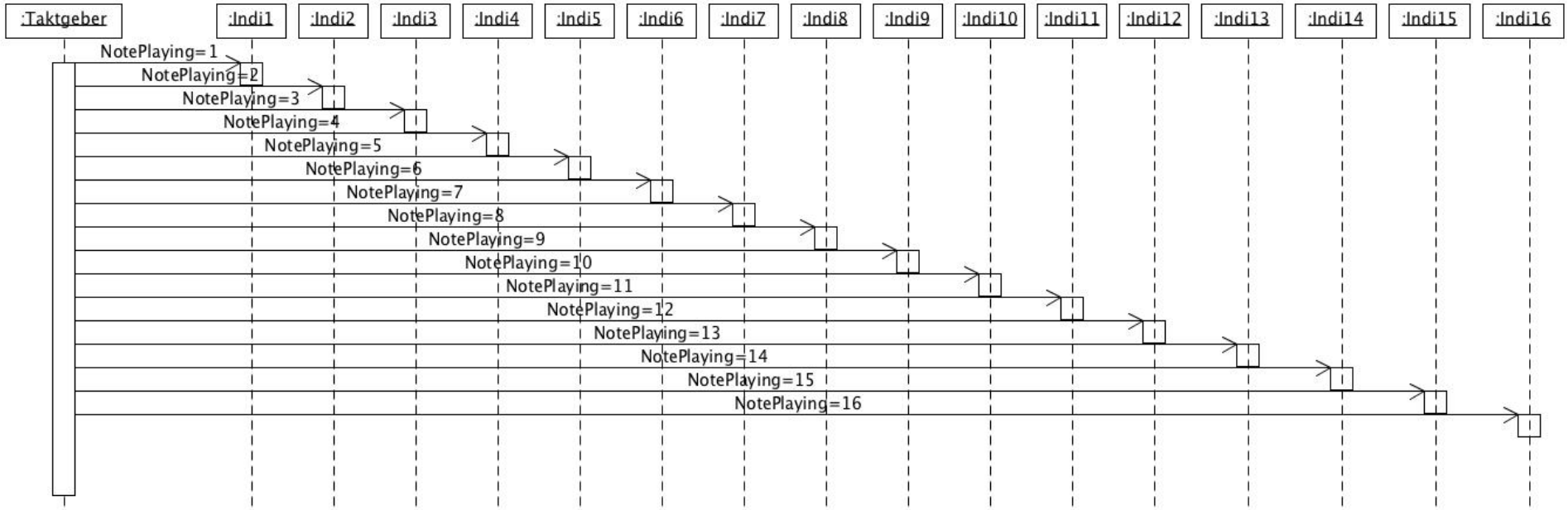
Klassendiagramm der Klasse „Indikator“:



Die Unterklassen von „Indikator“ fragen die Variable „NotePlaying“ ab und leuchten sozusagen auf, wenn „NotePlaying“ ihrer Stelle im Takt entspricht.

Zeitlich läuft das wie im folgenden Sequenzdiagramm dargestellt ab:

c) Sequenzdiagramm



Quellcode Anlage 2: die „act()“ Methode der Klasse „Taktgeber“, welche die Variable „NotePlaying“ updated

Quellcode Anlage 3: der Quellcode der Klasse „Indikator“, welche die Methode „Indi(int x)“ für ihre Unterklassen „Indi1-16“ bereitstellt

Quellcode Anlage 4: exemplarisch der Quellcode der Klasse „Indi1“, welcher derselbe ist wie der der restlichen Klassen „Indi2-16“ bis auf das Detail, dass in den einzelnen Klassen die Methode „Indi(int x)“ jeweils mit der genauen Stelle in den zwei Takten aufgerufen wird, welche sich im Namen der jeweiligen Unterklasse widerspiegelt

d) Quellcode

Im Folgenden werden Kommentare teilweise außer Acht gelassen!

Quellcode Anlage 2:

```
Timestepms=60000/(bpm*2);
PlayButton();
isPlaying=isClicked;
if(isPlaying){
        long Time  = System.currentTimeMillis();
        while(System.currentTimeMillis() < Time+Timestepms){
        Greenfoot.delay(1);
        }
        if(NotePlaying==16){
                NotePlaying=1;
        }
        else{
                NotePlaying++;
        }
```

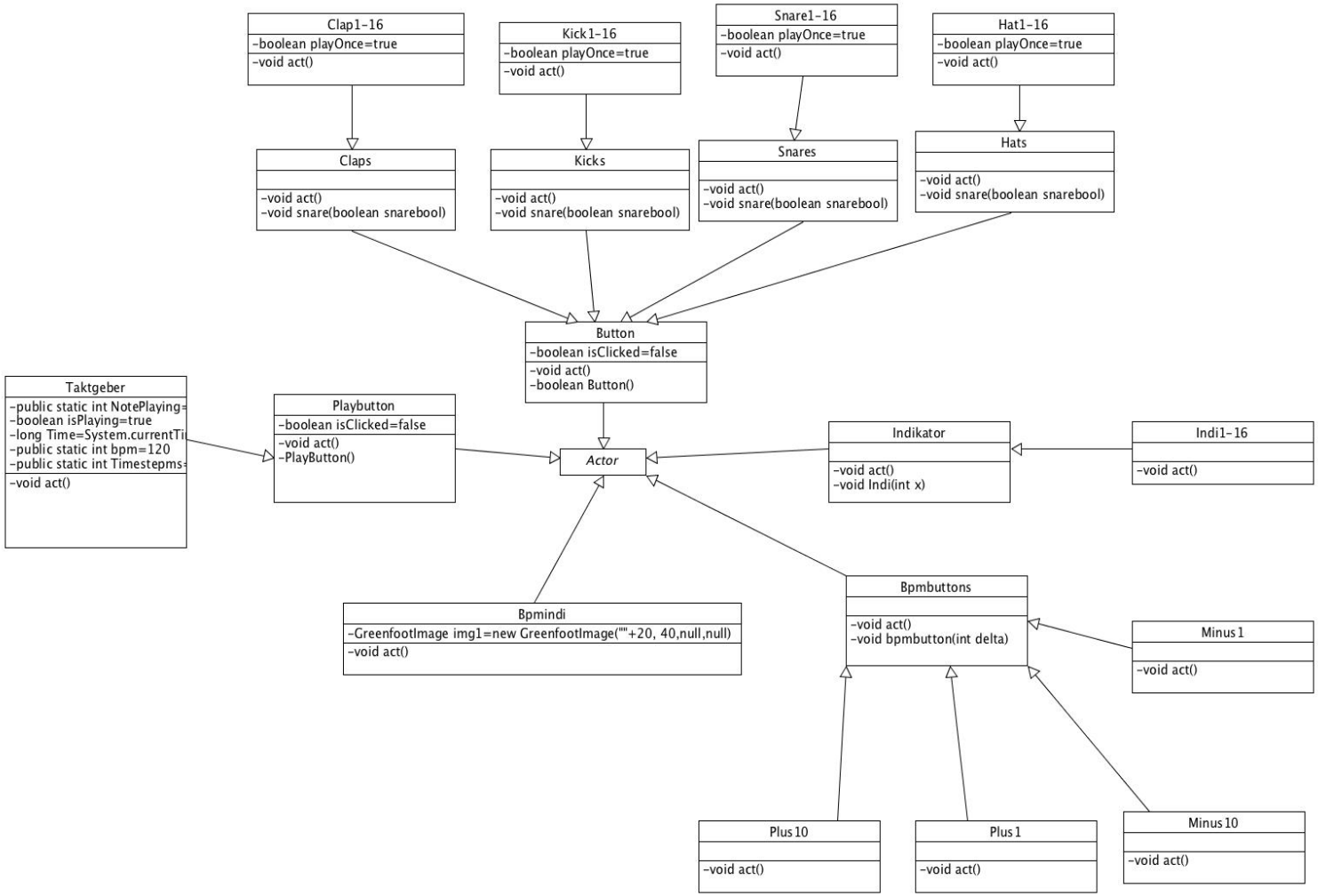
Quellcode Anlage 3:

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)
public class Indikator extends Actor
{
public void act() {}
public void Indi(int x){
        if(Taktgeber.NotePlaying==x){
                setImage("button-blue.png");
        }
        if(Taktgeber.NotePlaying!=x){
                setImage("button-black.png");
        }
}
}
```

Quellcode Anlage 4

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)
public class Indi1 extends Indikator
{
public void act() {
                    Indi(1);
          }
}
```

4. Klassendiagramm des gesamten Projektes



5. Bewertung von Extreme Programming / Fazit

Extreme Programming ist eine Arbeitsweise, die für Entwicklerteams geschaffen wurde. Da ich hingegen meine Software alleine programmiert habe und es sich außerdem um ein, im Gegensatz zu den Projekten, aus denen diese Arbeitsweise entstanden ist, relativ überschaubares Projekt handelte, ist diese Arbeitsweise für meinen Anwendungsfall überdimensioniert gewesen. Ich habe mir im Vorfeld jedoch überlegt, welche Teile des Extreme Programming für mein Projekt nützlich sein könnten und damit welche umgesetzt werden sollten, und konnte deswegen sehr effizient arbeiten. Einen großen Vorteil, welchen Extreme Programming für mich hatte, war der gut durchstrukturierte Ablauf von Planung, Umsetzung, Testen und eventueller darauffolgenden Änderungen an der Software beziehungsweise den Storycards. Durch die Verwendung von Storycards konnte ich die Anforderungen an meine Software ohne viel Aufwand, aber dennoch sehr präzise modellieren. Des Weiteren verhalfen mir diese, den Entwicklungsprozess in klare Abschnitte einzuteilen, welche sich einzeln besser umsetzen ließen, als wenn ich das Projekt als ganzes angegangen wäre. Unter anderem an dieser Stelle habe ich aber auch gemerkt, dass Extreme Programming ursprünglich für größere Vorhaben konzipiert wurde, da mein Projekt sich auch durchaus als Ganzes hätte durchführen lassen können, auch wenn mir die Einteilung sehr geholfen hat. Weitere Elemente, welche sich auf die Durchführung meines Projektes ausgewirkt haben, sind das kontinuierliche Testen und die systematische Kommunikation mit Außenstehenden. Diese zwei in Kombination verhalfen mir im Laufe des Projektes zu sinnvollem Feedback, welches ich in der weiteren Entwicklung berücksichtigte, sowie zu einer weiteren Storycard, welche zusätzlich zu den ursprünglichen implementiert wurde. Auch das Entwickeln von einzelnen Modultests hat dazu geführt, dass Probleme leichter aufzufinden und zu beheben waren.

Obwohl ich mir die Hintergründe sowie die genauen Gedanken und Ideen hinter Extreme Programming selber erarbeitet habe und deswegen vermutlich im Stande gewesen wäre Extreme Programming ohne Anleitung in meinem Projekt umzusetzen, war es dennoch hilfreich ein paar Elemente, namentlich die Erstellung und die Umsetzung von Storycards, im Voraus im Unterricht praktiziert zu haben. Der Einsatz des Lehrers war meiner Meinung nach genug, um die gröbsten Prinzipien zu verstehen und zu üben, aber nicht ausreichend, um tiefer in die Materie einzudringen. Mich persönlich störte dies jedoch nicht, da dadurch freies Arbeiten ermöglicht wurde und der Lehrer auch außerhalb des Unterrichts für etwaige Fragen zur Verfügung stand.

Ich würde bei meiner nächsten Softwareentwicklung dieser Art darauf achten, sie, soweit wie möglich, an einem Stück fertigzustellen, was bei diesem Projekt aufgrund von anderen, vor allem schulischen, Verpflichtungen nicht möglich war. Außerdem würde ich mir im Vorfeld eine Arbeitsweise aussuchen, welche womöglich besser auf ein Projekt dieser Größe passen würde.

6. Verwendete Software

Greenfoot V. 3.0.4: https://www.greenfoot.org/home

→IDE basierend auf der Java 1.8.0_92

7. Quellen

Wells, Don : http://www.extremeprogramming.org , zuletzt abgerufen am 20.5.17

Wells, Don : http://www.agile-process.org , zuletzt abgerufen am 20.5.17

https://en.wikipedia.org/wiki/Software_synthesizer , zuletzt abgerufen am 21.5.17

https://en.wikipedia.org/wiki/Software_sampler , zuletzt abgerufen am 21.5.17

https://en.wikipedia.org/wiki/Sampler_(musical_instrument) , zuletzt abgerufen am 21.5.17

https://de.wikipedia.org/wiki/Polyphonie_(Elektrophon) , zuletzt abgerufen am 21.5.17

8. Erklärung

Ich erkläre hiermit, dass ich die Seminararbeit selbstständig angefertigt und nur die im Literaturverzeichnis angeführten Quellen und Hilfsmittel verwendet habe.

....., den ....

Ort Datum Unterschrift des Schülers