Mobile Musik-Welten – Spielerische Exploration von digitalen Musiksammlungen

Endbericht zur Projektgruppe 554

Tim Delbrügger
Ümit Güler
Jannic Hartwecker
Annika Jordan
Matthias Kuchem
Jan Lahni
Dimitri Scheftelowitsch
Nils Vortmeier
Mirko Walter-Huber

13. Juni 2012

Betreut durch:
Mike Preuß
Igor Vatolkin

Lehrstuhl 11
Fakultät für Informatik

technische universität
dortmund

Inhaltsverzeichnis

1 Motivation und Ziele der Projektgruppe **7**
- 1.1 Ziele der PG . . . 8
  - 1.1.1 Verbindliche Ziele . . . 8
  - 1.1.2 Optionale Ziele . . . 9
- 1.2 Hilfestellungen . . . 9
  - 1.2.1 Zeitliche Abfolge . . . 9

2 Entwicklungsplattformen **11**
- 2.1 Java als Basis . . . 11
- 2.2 Plattformunabhängige Lösung . . . 12

3 Projektmanagement mit SCRUM **15**
- 3.1 Auswahl des Verfahrens . . . 15
- 3.2 Was ist Scrum? . . . 16
- 3.3 Scrum in der PG . . . 17
- 3.4 Erfahrungen mit Scrum . . . 18

4 Spielkonzept **21**
- 4.1 Die Spielidee . . . 21
  - 4.1.1 Das Spielbrett . . . 21
  - 4.1.2 Die Minispiele . . . 22
  - 4.1.3 Mehrspieler . . . 23
- 4.2 Reflektion . . . 23

5 Architektur der Engine **25**
- 5.1 Überblick . . . 25
  - 5.1.1 Anforderungen . . . 25
  - 5.1.2 Struktur . . . 26
- 5.2 Funktionen . . . 28
  - 5.2.1 Datenverwaltung . . . 28

5.2.2 Grafik . . . 29
5.2.3 Sound . . . 30
5.2.4 Eingaben . . . 31
5.2.5 Grafische Benutzeroberfläche . . . 32
5.2.6 Netzwerk . . . 32
5.2.7 Sonstiges . . . 34
5.3 Tools . . . 35
5.3.1 ShapeViewer . . . 35
5.3.2 Continuous Integration System . . . 35
5.3.3 AIClient . . . 36
5.4 Reflektion . . . 36

6 Arbeit mit musikalischen Merkmalen 37
6.1 Generelles Vorgehen . . . 37
6.2 Konstruktion von neuen Merkmalen . . . 38
6.2.1 FeatureVisualizer . . . 38
6.2.2 Einsatz des FeatureVisualizers . . . 40
6.3 Konstruktion des Spielbretts . . . 42

7 Entwicklung des Spiels 45
7.1 Tap by Tap . . . 45
7.2 TowerDefense . . . 46
7.2.1 Spielfeld . . . 47
7.2.2 Gegner . . . 47
7.2.3 Gegnerwellen . . . 48
7.2.4 Türme . . . 48
7.3 MusicFighter . . . 48
7.3.1 Steuerung und Spielmechanik . . . 49
7.3.2 Gegner . . . 50
7.3.3 Items . . . 51

8 Tests am lebenden Menschen 53
8.1 Feature-Studie . . . 54
8.1.1 Teilnehmer . . . 54
8.1.2 Testprozedur . . . 54
8.1.3 Ergebnisse . . . 55
8.1.4 Statistische Analyse . . . 55
8.1.5 Diskussion . . . 55
8.2 Game-Balancing . . . 56

9 Ausblick 59
9.1 Verbesserung der Minispiele . . . 59
9.2 Neue Minispiele . . . 59
9.3 Vermarktung . . . 60
9.3.1 Benutze nur „freie“ Musik . . . 60
9.3.2 Musik-Bezahl-Modell . . . 60
9.3.3 Spieler zahlt für Features . . . 60

Literaturverzeichnis **61**

KAPITEL 1

Motivation und Ziele der Projektgruppe

Annika Jordan

Der Markt für Mobiltelefone hat sich in den vergangenen Jahren deutlich verändert. Die modernen „Smartphones“ sind mobile Rechner geworden und vereinigen Kommunikationsmittel mit Unterhaltungselektronik. Neben den „althergebrachten“ Kommunikationsformen Telefon und SMS bieten Smartphones heutzutage bezahlbaren und relativ schnellen Zugang zum Internet sowie Verbindung untereinander über Bluetooth oder ähnliche Technologien. Weiterhin kann eine große Anzahl von Musikstücken mitgeführt und abgespielt werden und der Funktionsumfang der Geräte kann durch Applikationen („Apps“) aus dem Internet erweitert werden.

Mobile Geräte haben allerdings immer noch den Nachteil, dass ihnen im Vergleich zu festen Rechnern nur stark begrenzte Ressourcen zur Verfügung stehen, insbesondere Speicherplatz und Rechenkraft. Weiterhin sollen sie möglichst klein sein, haben somit einen kleinen Bildschirm und begrenzte Möglichkeiten der klassischen Eingabe über Tastatur oder Maus. Andererseits stehen ihnen oft weitere Eingabemöglichkeiten wie Bewegungssensor oder GPS zur Verfügung.

Alle diese Bereiche sollen im Rahmen dieser Projektgruppe 554, „Mobile Musik-Welten: Spielerische Exploration von digitalen Musiksammlungen“ miteinander verbunden werden. Die Projektgruppe ist eine Lehrveranstaltung im Hauptstudium der Diplom- sowie der Masterstudiengänge der Fakultät für Informatik der Technischen Universität Dortmund. Ein Projekt, welches in der Regel aus einem Forschungsbereich der Betreuer der Projektgruppe stammt, soll von einer Gruppe von acht bis zwölf Studenten über einen Zeitraum von zwei Semestern durchgeführt werden. Dies soll nicht nur dazu dienen, die im Studium erlernten Methoden in die Praxis umzusetzen, auch sollen die Teilnehmer Erfahrungen im Bereich Projektmanagement erlangen und lernen, sich in einem größeren Team zu organisieren.

1.1 Ziele der PG

Die Ziele der Projektgruppe werden im Großen und Ganzen durch die Vorgaben im Projektgruppenantrag bestimmt. Nachfolgend werden daher zunächst diese Ziele aufgelistet und dazu Stellung genommen. Des Weiteren werden aber auch die Ideen und Wünsche, die von der Projektgruppe selbst hinzugefügt wurden, dargestellt. Im Laufe des vorliegenden Endberichts kann so nachvollzogen werden, inwiefern die gesteckten Ziele erreicht und umgesetzt werden konnten.

1.1.1 Verbindliche Ziele

Kernziel der Projektgruppe ist die Entwicklung eines Spiels, welches die Navigation durch große Sammlungen von Musikdateien auf Smartphones der Marke Blackberry vom Typ Torch 9800 ermöglicht. Diese Zielplattform hat dabei einige bedeutende Einschränkungen: Die Rechenleistung und Speichergröße ist ebenso beschränkt wie die Netzwerkbandbreite. Gerade in Hinblick auf die Navigation durch große Datensammlungen ist auch die geringe Bildschirmgröße eine wichtige Limitierung. Aber auch die positiven Eigenschaften dieser Plattform sollen nach Möglichkeit voll ausgenutzt werden. So gibt es etwa einen Beschleunigungssensor, GPS-Navigation, Kamera und Mikrofon. Um den Mitgliedern der Projektgruppe Zugang zu dieser Plattform zu geben, stellt die Firma Research in Motion (RIM) vier Blackberry Torch 9800 Geräte.

Der Aspekt der Navigation durch die Musiksammlung stellt die Eigenständigkeit des Produktes sicher, denn so etwas gibt es bisher nach Kenntnis der Organisatoren und Teilnehmer noch nicht.

Um diese Navigation umzusetzen, soll die Projektgruppe Musikmerkmale verwenden. Diese sind in der Literatur vielfältig vorhanden, aber bisher nur auf leistungsfähigen Computersystemen ohne gravierende Leistungseinschränkungen berechnet worden. Ob und inwiefern dies auch auf mobilen Geräten möglich ist, kann von der Projektgruppe erforscht werden. Bei der Berechnung soll auch, wenn möglich, auf existierende Frameworks wie etwa AMUSE zurückgegriffen werden. Diese Merkmale sind allerdings zu vielfältig, um direkt in eine Spielkarte umgesetzt zu werden. Deshalb sollen viele Merkmale kombiniert werden und mithilfe einer Methode zur Dimensionsreduktion auf einen bespielbaren Raum abgebildet werden (zum Beispiel mit zwei Dimensionen). Als zwei Möglichkeiten werden der Projektgruppe die Nutzung selbst-organisierender Karten (Self-Organizing feature Maps, SOMs) oder Multi-Dimensional Scaling empfohlen.

Als weitere Schwierigkeit kommt der Mehrspieler-Aspekt hinzu. Mindestens zwei Spieler sollen das Spiel gegeneinander spielen können.

Wegen der Komplexität der Problemstellung und der zu nutzenden Techniken ist die Einarbeitung in die Thematik eine der Hauptaufgaben der Projektgruppe, welche auch über die anfängliche Seminarphase hinausgeht. Zusätzlich soll das Arbeiten im Team und das Präsentieren von Ergebnissen die sozialen Kompetenzen der Teilnehmer trainieren.

1.1.2 Optionale Ziele

Als besondere Funktionen kann das Spiel Gruppen verwalten, etwa über Lokalität (erkannt durch GPS-Ortung) oder Freundschafts-Stati in sozialen Netzwerken wie Facebook. Mehr als zwei Spieler sind erwünscht, erfordern aber deutlich größeren Aufwand bei der Synchronisation und sind deshalb optional. Ein Client-Server-Modell erscheint hier sinnvoll, vor allem weil der Server auch die schwierigen Berechnungen der Musikmerkmale durchführen könnte.

1.2 Hilfestellungen

Bei der Umsetzung dieser Ziele soll die Projektgruppe durch die Beratung durch Wolfgang Theimer (RIM) unterstützt werden. Zwei Mitglieder, Annika Jordan und Tim Delbrügger, besuchen einen Programmierkurs für mobile Geräte an der Ruhr-Universität Bochum. Zudem stellt RIM vier Blackberry Torch Geräte bereit. Außerdem stehen die erfahrenen Betreuer Mike Preuss und Igor Vatolkin der Projektgruppe zur Seite. Der Lehrstuhl unterstützt die PG durch einen Server, welcher durch die Projektgruppe selbst verwaltet wird und eine Versionsverwaltung und ein Ticketsystem bereitstellt. Des weiteren fungiert er als Spielserver in der für das entwickelte Spiel wichtigen Client-Server-Architektur. Für das zweite Semester stehen der PG weiterhin fünf Galaxy Nexus S Geräte vom Lehrstuhl zur Verfügung, um die Entwicklung für Android (siehe Kapitel 2) zu unterstützen.

1.2.1 Zeitliche Abfolge

In [PV11] wird der Projektgruppe empfohlen, zunächst einzelne Komponenten, etwa zur Merkmalsextraktion aus Musikstücken, zu entwickeln, die dann im zweiten Semester zu einem fertigen Gesamtpaket zusammengefasst werden. Die Projektgruppe hat sich jedoch zu Beginn einstimmig dazu entschlossen, direkt mit dem Spiel zu beginnen und erst nach und nach fortgeschrittene Komponenten wie Netzwerkmodus und Musikmerkmale zu integrieren. Hierfür gab es zwei wichtige Beweggründe: Zum einen ist die Komplexität der Anwendung bei der Verbindung all dieser unbekannten Technologien sehr hoch (und damit fehleranfällig), zum anderen soll agil entwickelt werden. Bei der agilen Softwareentwicklung soll möglichst schnell ein Prototyp des Produktes entstehen, was dem vorgeschlagenen Prinzip widerspricht. Natürlich soll die Architektur des Spiels aber von Anfang an so erweiterbar gehalten werden, dass die noch fehlenden Komponenten später leicht hinzugefügt werden können. Die Entscheidung für die modulare, agile Entwicklung soll der PG helfen, effizient und parallel zu arbeiten.

Im vorliegenden Endbericht soll nun das Spiel und der Entwicklungsprozess näher vorgestellt werden. Dazu werden im Kapitel 2 zunächst die beiden unterstützten Entwicklungsplattformen näher betrachtet. Im dritten Kapitel (3) wird das von der Projektgruppe gewählte Vorgehensmodell zur Softwareentwicklung SCRUM kurz vorgestellt und notwendige Anpassungen betrachtet. Kapitel 4 befasst sich dann mit dem Spielkonzept, dass von der PG entwickelt wurde. Dabei wird auf technische Details verzichtet. In Kapitel 5 wird dann die eigenständig entwickelte Game Engine detailliert vorgestellt.

Dabei wird auf die einzelnen Funktionen eingegangen und die getroffenen Entscheidungen reflektiert.

Ein wesentlicher Bestandteil der Arbeit der Projektgruppe ist der Umgang mit musikalischen Merkmalen und deren Einbindung ins Spiel. Im 6. Kapitel wird deshalb erläutert, welche Tools eingesetzt und entwickelt wurden, um Musikmerkmale sinnvoll zu verwenden. Auf diesen Informationen aufbauend werden dann die drei Minispiele einzeln und im Detail vorgestellt (Kapitel 7).

Um herauszufinden, ob die Ideen der PG Anklang finden und ob die Nutzung von Features zur Generierung der Spielinhalte funktioniert, wurde eine kleine Studie durchgeführt, auf die in Kapitel 8 näher eingegangen wird. Das letzte Kapitel 9 bietet einen Überblick darüber, was noch zu tun wäre und welche Möglichkeiten das Spiel für eine weitere Entwicklung und eventuelle Vermarktung bietet.

Weitere Informationen zur Arbeit der Projektgruppe finden sich auch im Zwischenbericht [DmGH+11].

KAPITEL 2

Entwicklungsplattformen

Annika Jordan

Eines der Ziele der PG war es, den Kern des Programmcodes plattformunabhängig zu gestalten, um so eine verhältnismäßig einfache Portierung des entwickelten Spiels auf verschiedene Plattformen zu ermöglichen. Im Falle der PG handelt es sich dabei konkret um Android und Blackberry OS6. Da beide Betriebssysteme gewisse Eigenheiten und Besonderheiten aufweisen, sollen sie im nachfolgenden Kapitel kurz vorgestellt werden. Dabei sollen Unterschiede aber auch Gemeinsamkeiten aufgezeigt werden, um die im weiteren Verlauf des Berichts beschriebenen Entwicklungstätigkeiten besser einschätzen zu können.

2.1 Java als Basis

Sowohl *Blackberry OS* als auch *Android* nutzen als Programmiersprache *Java*. Da würde es für eine plattformunabhängige Entwicklung nahe liegen, auf den *Java ME* Standard für mobile Geräte zurückzugreifen. Diese werden in „.jar“ Dateien gespeichert und können von jedem Gerät mit einer *Java VM* ausgeführt werden. Hierbei muss aber berücksichtigt werden, dass *Android*, obwohl es *Java* nutzt, nicht auf den *Java ME* Standard aufsetzt und **keine** *Java VM* besitzt, also **nichts** mit „.jar“ Dateien anfangen kann. Bei *Blackberry* steht dagegen eine *Java VM* zur Verfügung.

Android setzt statt einer *Java VM* auf eine *Dalvik VM*, welche speziell vorkompilierten *Dalvik*-Bytecode ausführt. Dieser nutzt nach [BP10] schon in der virtuellen Prozessorarchitektur die Existenz von Registern aus, was viele Berechnungen stark beschleunigen kann. Hierzu wird der *Java*-Bytecode nach dem normalen *javac* Compiler noch in einem weiteren Schritt in *Dalivik*-Bytecode übersetzt.

Neben der Performance gab es wohl noch einen weiteren Vorteil der *Dalvik VM* für Google gegenüber einer *Java VM*: Für die Verwendung einer *Java VM* fallen gegenüber Sun bzw. nun Oracle Lizenzgebühren an, nicht aber bei reiner Verwendung der Sprache *Java*.

Nun könnte man denken, man muss nur einen weiteren Kompilierungsschritt durchführen, um eine *Java ME* Anwendung auf *Android* ausführen zu können. Dies ist aber **falsch**, weil *Android* nur sehr wenige Klassenbibliotheken der *Java ME* kennt.

Um dennoch *Java ME* Anwendungen in auf einem *Android* Gerät auszuführen, gibt es verschiedene Ansätze, welche im Test unterschiedlich schlecht funktioniert haben. Wir konnten nicht alle davon testen, für eine ernsthafte Anwendung scheinen diese Methoden jedoch allesamt ungeeignet.

Überblick über die Funktionen

Blackberry

Blackberry unterstützt im Gegensatz zu *Android* die meisten Klassen der *Java Microedition*. Zudem liefert *Blackberry* viele eigene Pakete, mit denen man zum Beispiel das *Blackberry* Look& Feel bei Benutzeroberflächen erhalten kann. Eine Übersicht über alle unterstützten Bibliotheken findet sich unter `http://www.blackberry.com/developers/docs/6.0.0api/category-summary.html`.

Android

Android bietet Zugriff auf die meisten Klassen der *Java SE* und erweitert diese noch um spezifische wie etwa für die Oberflächengestaltung oder die integrierte *SQLite*-Datenbank. Insbesondere genannt seien *Java SE* Features wie *Generics*, *Enums* oder *for-each* Schleifen. **Wichtig:** Es gibt keinen Zugriff auf die Pakete *java.awt* (enthält *Java2D*) oder *javax.swing*. Sämtliches Zeichnen von Oberflächen wird mit *Android*-Paketen realisiert.

2.2 Plattformunabhängige Lösung

Um beiden Plattformen gerecht zu werden und redundanten Code zu vermeiden, entschied sich die Projektgruppe für die Entwicklung einer eigenen, plattformunabhängigen Game Engine, die den kleinsten gemeinsamen Nenner beider Plattformen zu Grunde legt. Dabei handelt es sich lediglich um einige grundlegende Java Pakete. Da die *JavaME* in etwa der Java-Version 1.3 entspricht, musste im gesamten Spiel auf alles verzichtet werden, was erst später in die Java-Bibliothek aufgenommen wurde. Viele der benötigten Funktionalitäten wurde daher von Hand implementiert. Zudem gibt es für jede unterstützte Plattform eine Engine-Teil, der die Schnittstelle zum Gerät darstellt. Hier, und nur hier, wird auf die plattformspezifischen Funktionen zurückgegriffen und so das Darstellen von Inhalten, der Zugriff auf den Speicher, das Abfragen von Sensorwerten

und so weiter realisiert. Kapitel 5 informiert ausführlich über die Funktionalität der Game Engine.

Um neben *Android* und *Blackberry* noch weitere Betriebssysteme zu unterstützen, müsste genau dieser spezifische Teil für die neue Plattform implementiert werden. Damit ist eine Portierung der Game Engine und des gesamtem Spiels zumindest für *Java*-basierte Systeme mit wenig Aufwand zu realisieren.

KAPITEL 3

Projektmanagement mit SCRUM

Annika Jordan

Dieser Abschnitt befasst sich mit dem Thema Projektmanagement in der Projektgruppe. Die PG hat sich für die agile Entwicklung mit Scrum entschieden. Dazu mussten jedoch einige Anpassungen vorgenommen und das Modell an die Bedürfnisse der Projektgruppe angepasst werden. Dieses Kapitel beleuchtet den Auswahlprozess und beschreibt Scrum kurz im Allgemeinen, bevor näher auf unsere Änderungen eingegangen wird. Im nachfolgenden Abschnitt wird von den Erfahrungen mit Scrum berichtet und dieses Vorgehensmodell im Hinblick auf Produktivität und Teamarbeit kritisch hinterfragt.

3.1 Auswahl des Verfahrens

Eine wichtige Aufgabe im Rahmen eines Projektes ist die Auswahl eines für das Team geeigneten Projektmanagementverfahrens. Schnell war klar, dass für die PG nur ein modernes, agiles Verfahren in Frage kommt, nach den Prinzipien des Manifest für Agile Softwarentwicklung [BBvB+]. So soll vermieden werden, dass das erste Semester ausschließlich aus Planung und Analyse besteht und noch nichts implementiert wird. Die Idee dahinter ist es, die Motivation dadurch zu steigern, dass man bereits frühzeitig erste Erfolge sieht.

Ein statisches, unflexibles Verfahren wie das klassische Wasserfallmodell schien für die Aufgabe auch deswegen ungeeignet, da nur sehr geringe Vorkenntnisse im Bereich der Programmierung für mobile Endgeräte vorhanden waren und auch ziemlich unklar war, was überhaupt realisierbar ist. Außerdem hatte die Projektgruppe viele Freiheiten für die konkrete Gestaltung des Endprodukts und so entwickelte sich die gemeinsame Vision davon mit Fortschreiten des Projekts und nicht zu Beginn. Mit einem agilen Vorgehen

konnten früh Dinge ausprobiert und so viele Probleme schon zu Beginn erkannt und in der nächsten, zeitnahen Planungsphase berücksichtigt werden.

3.2 Was ist Scrum?

Scrum ist ein bekanntes und bewährtes Verfahren, das oft in großen Unternehmen zum Einsatz kommt. Bei Scrum gibt es drei Rollen:

1. Den **Product Owner**, der vor allem die Auftraggeber, aber auch sonstige Stakeholder vertritt.
2. Der **Scrum Master**, der die Philosophie und die Regeln von Scrum ins Projekt trägt und unterstützend tätig wird um die Produktivität zu steigern.
3. Die **Teams**, bestehend aus den Entwicklern, die für die Umsetzung verantwortlich sind und während der Implementierungsphasen autonom handeln.

Wie in den meisten Projektmanagementverfahren gibt es auch bei Scrum verschiedene Phasen. Dabei wird zu Beginn das **Product Backlog** erstellt, eine priorisierte Liste von Anforderungen an das Produkt. Diese werden idealerweise in Form von *User Stories* formuliert. User Stories sind in Alltagssprache verfasste Anforderungen an das Produkt. Hierbei wird aus der Sicht des Anwenders beschrieben, wie sich die Software verhalten soll. Insbesondere ist das Product Backlog also keine Liste technischer Anforderungen wie im klassischen Projektmanagement. Für die Erstellung und Priorisierung dieser Liste ist der Product Owner zuständig. Dabei bekommen die wichtigen Anforderungen, welche dann auch als erstes implementiert werden sollen, eine hohe Priorität.

Ein **Sprint** ist eine Entwicklungsphase in Scrum und dauert typischerweise 2 oder 4 Wochen. Nach der Erstellung des Product Backlog geht es dann in die **Sprint-Planung**. Im ersten Schritt entscheidet das Team dabei, welche Einträge aus den Product Backlog in diesem Sprint umgesetzt werden können. Sie sind dabei angehalten die Einträge mit hoher Priorität zu bevorzugen. Dabei soll das Team genau so viele Enträge auswählen, wie es in der Zeit des Sprints schafft. Anschließend wird im **Sprint Backlog** alles festgehalten, was getan werden muss, um die aus dem Product Backlog gewählten Anforderungen umzusetzen. Das kann allerdings auch Projektunabhängige Tätigkeiten, wie das Einrichten von Entwicklungstools einschließen. Es wird bei der Erstellung kein Wert auf Vollständigkeit gelegt. Der Sprint Backlog soll das Team unterstützen, nicht binden. Deshalb sind die Aufgaben bei Erstellung typischerweise sehr grob gefasst und werden dann bei der Umsetzung in mehrere Aufgaben für die einzelnen Teilschritte zerlegt.

Ist das Sprint Backlog fertiggestellt, kann der Sprint beginnen. In der nachfolgenden Phase arbeitet das Team autonom und ohne Beeinflussung von außen an der Erfüllung der Sprintziele. Es steht dem Team also frei sich zu organisieren wie es möchte. Der ScrumMaster steht lediglich bereit, um Probleme im Team oder auch technischer Natur zu lösen. Um die Kommunikation im Team hoch zu halten, werden täglich **Daily Scrums** abgehalten. In diesen Meetings, die typischerweise zu Beginn eines Tages stattfinden, muss jedes Teammitglied kurz drei Fragen beantworten:

1. Was hast du seit dem letzten Daily Scrum gemacht?

2. Was machst du bis zum nächsten Daily Scrum?
3. Behindert dich etwas in deiner Arbeit?

Die Fragen werden vom ScrumMaster gestellt, der diese Treffen moderiert. Dieser notiert sich auch mögliche Probleme, um sie später lösen zu können. Er ist dafür verantwortlich, eine möglichst hohe Produktivität des Teams zu gewährleisten, indem er Schwierigkeiten zeitnah löst.

Am Ende eines Sprints steht der **Sprint Abschluss**. Dabei präsentiert das Team alle Entwicklungen. Alle Stakeholder sind zu diesem Treffen eingeladen. Sie sollen dem Team Feedback geben und müssen die Arbeit bewerten. Jeder Punkt aus dem Product Backlog der umgesetzt werden sollte, wird akzeptiert oder abgelehnt. Außerdem besteht in der Zeit zwischen den Sprints die Möglichkeit den Product Backlog zu ändern.

Im **Sprint Rückblick** bespricht das Team was gut funktioniert hat und was verbessert werden kann. Der Entwicklungsprozess wird so immer wieder reflektiert und verbessert. Nach diesem Treffen beginnt dann typischerweise die nächste Sprint Planung, wenn der Product Owner das Projekt nicht für beendet erklärt hat.

Details über Scrum kann man am besten der Fachliteratur entnehmen. Für Scrum Einsteiger eignet sich da besonders [Glo11]. Für Fortgeschrittene gibt es eine Sammlung von typischen Problemen bei dem Einsatz von Scrum in Form von Erfahrungsberichten in [Sch04].

3.3 Scrum in der PG

In den meisten Beschreibungen von Scrum wird davon ausgegangen, dass die Entwickler Vollzeitangestellte sind, oder zumindest täglich in einem Daily Scrum zusammen kommen können. Um Scrum in einer Projektgruppe nutzen zu können, bei der jedes Mitglied zu unterschiedlichen Zeiten und auch deutlich weniger am Projekt arbeitet, mussten wir es an unsere Bedürfnisse anpassen.

Ein Problem in dem Zusammenhang waren zeitliche Abstände. Innerhalb der PG mussten vor allem die Fragen geklärt werden, wie lange ein Sprint dauern soll und wie oft ein Daily Scrum sinnvoll ist. In Anbetracht der Gesamtdauer der PG und eines Semsters einigte sich die PG auf eine Sprintdauer von 4 Wochen. Diese Zeit sollte ausreichen, um das Projekt sinnvoll voranzubringen und ist zugleich auch noch kurz genug, um mehrere solche Sprints in einem Semester unterzubringen. Für die Daily Scrums einigten wir uns darauf, in jedem der Treffen der PG eine solche Fragerunde abzuhalten, sodass es zwei Daily Scrums pro Woche gab. Die Vor- und Nachbereitungstreffen eines Sprints wurden zeitlich an die Länge einer PG-Sitzung angepasst.

Eine weitere Abweichung vom Originalvorschlag für Scrum ist die Tatsache, dass der ScrumMaster selbst Teil eines Teams ist und mit entwickelt, statt nur helfend zur Seite zu stehen. Da wir innerhalb der PG die Aufgaben des ScrumMasters als eher gering einschätzten, entschieden wir, ihn in ein Team zu integrieren. Diese Lösung erschien auch auf Grund der geringen Gruppengröße als sinnvolle Anpassung, da so ein Entwickler mehr zur Verfügung stand.

Die Teams für die einzelnen Sprints wurden für jeden Sprint neu nach Interessen und Fähigkeiten zusammengestellt. Die Teamgrößen variierten zwischen zwei und fünf Entwicklern.

Die PG Betreuer übernahmen gemeinsam die Rolle des Product Owners.

3.4 Erfahrungen mit Scrum

1. Sprint Nachdem das Modell an die Bedürfnisse der PG angepasst wurde, startete die PG mit der Erstellung des Product Backlogs für das Musikspiel. Nachdem die Betreuer die Prioritäten vergeben hatten, konnte der erste Sprint starten.

Für diesen Sprint hatte sich die PG viel vorgenommen. So sollte neben der Game Engine auch ein erstes Minispiel entstehen. Schnell stellte sich allerdings heraus, dass die Erwartungen nicht erfüllt werden konnten. Die gesteckten Ziele waren viel zu hoch, die Aufgaben zwischen den Teams ungleich verteilt und die Produktivität in den Teams zu unterschiedlich.

Während das selbstorganisierte Arbeiten im „Engine“ Team gut funktionierte, gab es im Team, welches sich um das Minispiel kümmerte, viele Probleme mit der Verteilung der Zuständigkeiten. Hier hätte das Team einen Teamleiter bestimmen können, der die Koordination der Aufgaben übernimmt.

Einen positiven Eindruck hinterließen von Anfang an die Daily Scrums. Im ersten Sprint wurden diese in den getrennten Teams durchgeführt, um die Regeln von Scrum einzuhalten. Die Teams sollten sich dadurch nicht gegenseitig beeinflussen. Fragen an das jeweils andere Team wurden dann danach in der Sitzungen gestellt und geklärt.

2. Sprint Nachdem der erste Sprint für gescheitert erklärt wurde, beschloss die PG noch einige weitere, kleinere Anpassungen an Scrum vorzunehmen. Es wurde der Wunsch geäußert, ein drittes Daily Scrum einzuführen, um die Kommunikation zu verbessern. Da der Originalansatz sogar ein tägliches Treffen vorsieht und die Zeitspanne vor allem von Donnerstag bis Montag recht lang ist, einigte sich die PG auf ein zusätzliches Meeting im Teamspeak am Samstagmittag. Allerdings wurde aufgrund des ungünstigen Termins am Wochenende von der Anwesenheitspflicht abgesehen.

Auch wurde beschlossen, dass die Teams ihre Daily Scrums nicht mehr für sich abhalten, sondern sich die komplette Gruppe trifft. So sollen Missverständnisse und Unklarheiten in der Kommunikation zwischen den Teams früher auffallen. Zu dem wurde die "Macht"des ScrumMasters erweitert. Er darf und soll nun auch führend eingreifen, wenn es Probleme mit der Produktivität einzelner Mitglieder oder sonstige Schwierigkeiten gibt.

Neben dieser rein technischen Anpassung des Projektmanagementverfahrens wurde auch die komplette Sprintplanung überdacht. Insgesamt wurden deutlich weniger Ziele gesetzt, als noch im ersten Sprint. Dafür wurde deutlich mehr Zeit und Sorgfalt in das Erstellen des Sprint Backlogs investiert. Es wurden kleinschrittige ToDos formuliert und wenn möglich sofort die Zuständigkeiten vergeben. Außerdem sollte das Ticketsystem

vermehrt benutzt werden, um einfach eine bessere Übersicht über die Aufgaben zu haben.

Der Sprint selbst verlief dann erfolgreicher als der erste. Vor allem hatte die verbesserte Kommunikation über die Teamgrenzen hinweg positive Auswirkungen auf das Projekt. Der ScrumMaster hatte vor allem mit technischen Schwierigkeiten zu kämpfen, konnte aber immer eine Lösung präsentieren, sofern dies in seiner Macht stand.

Insgesamt kann die PG mit dem Verlauf des zweiten Sprints zufriedener sein als mit dem Verlauf des ersten.

Das zweite Semester Da nach anfänglichen Schwierigkeiten im ersten Semester viele Änderungen von Nöten waren, die PG zu Beginn des zweiten Semesters aber ein funktionierendes Modell gefunden hatte, werden hier nicht mehr die einzelnen Sprints beleuchtet sondern lediglich zusammenfassend Änderungen beschrieben.

Aufgrund der vielen Terminschwierigkeit mit dem Daily Scrum am Samstag entschied die PG erst einmal wieder mit zwei Daily Scrums auszukommen und nur bei Bedarf zusätzliche Meetings anzusetzen. Die Beteiligung aller Teams wurde jedoch beibehalten. Zwischenzeitlich entwickelten sich während der Daily Scrums immer wieder Diskussionen, die die eigentlich vorgesehene Zeit von fünfzehn Minuten weit überschritten. Aus diesem Grund waren die Sitzungsleiter und der Scrum Master angehalten, die Diskussionen zu unterbinden und entstehende Fragen auf später zu verschieben. Dies erwies sich als wichtig und richtig, um die Funktion des Daily Scrums, nämlich alle Beteiligten über den aktuellen Stand zu informieren und mögliche Probleme aufzudecken, nicht aus den Augen zu verlieren.

Des weiteren wurde auch das eingesetzte Ticketsystem stärker benutzt. Die Teams organisierten auf diese Weise ihre Sprint Backlogs. Hierbei wurde vor allem darauf geachtet, dass jedem Ticket auch ein PG-Teilnehmer zugewiesen wurde und zusätzlich ein Enddatum innerhalb des Sprints festgelegt wurde.

Um gegen Ende der Projektgruppe die vielen kleinen Baustellen zu schließen, wurde in bis zu vier kleinen Teams gearbeitet. Diese Zweier- und Dreiergruppen stellten sich als sehr effizient heraus, auch wenn sie aus Scrum Sicht zu klein waren.

Zusammenfassend kann man sagen, dass die PG im zweiten Semester ihr Projektmanagementverfahren gefunden hatte und damit vernünftig arbeiten konnte.

Fazit Abschließend wollen wir ein Fazit ziehen und den Einsatz von Scrum in der Projektgruppe bewerten.

Nach den vielen Anpassungen, die im Laufe des ersten Semesters durchgeführt wurden, lässt sich aus unseren Erfahrungen nur sehr schwer etwas über die Tauglichkeit vom originalen Scrum-Ansatz ableiten. Es stellte sich heraus, dass das Arbeiten in einer Projektgruppe sich doch sehr von der Arbeit in einem Unternehmen unterscheidet. Aus diesem Grund war der Einsatz des Ursprungsverfahrens nicht realisierbar. Bei dem Projektmanagementverfahren, welches die PG angewendet hat, handelt es sich mehr um ein dynamisches, agiles Verfahren, welches die Elemente von Scrum enthält, die für uns passend erschienen und die uns gefallen haben.

Ein wichtiges Element davon sind die Daily Scrums. Unserer Erfahrung nach stellen sie nicht nur eine regelmäßige Kommunikation sicher sondern erhöhen die Produktivität auch direkt. Da jeder Teilnehmer bei diesen Treffen berichten muss, was er seit dem letzten Treffen am Projekt getan hat, steigert das die Motivation, tatsächlich auch etwas vorweisen zu können.

Die Arbeit in kleinen zweier oder dreier Teams stellte für uns die produktivste Lösung dar, da die Einteilung der Aufgaben gut gelang und jeder genug Verantwortung für einen Bereich übernehmen konnte und musste. Dennoch war es wichtig die Kommunikation zwischen den Teams hoch zu halten. Die kurzen Sprints mit klar abgesteckten Zielen sorgten zusätzlich für eine höhere Motivation.

Insgesamt muss man feststellen, dass der ursprüngliche Scrum Ansatz für uns und vermutlich auch allgemein für Projektgruppen nur eingeschränkt geeignet ist. Nach einigen Anpassungen an unsere Wünsche und Bedürfnisse, sind wir jedoch mit dem Ergebnis zufrieden und konnten einige Elemente gewinnbringend einsetzen.

KAPITEL 4

Spielkonzept

Annika Jordan

In diesem Kapitel soll das Spielkonzept des von der PG entwickelten Spiels „BeatThe-Beat“ vorgestellt werden. Dabei sollen sowohl der reine Ablauf eines Spiels, als auch die Ideen, die dahinter stehen aufgezeigt werden. Im Laufe der PG wurden viele Ideen zum Spielablauf und -design entwickelt und verworfen. Nachfolgend werden die wichtigsten Entscheidungen präsentiert und diskutiert, um ein Verständnis für das Spiel zu schaffen. Dabei geht es (hier) nur hintergründig um die technische Realisierung.

4.1 Die Spielidee

„Beat the Beat“ ist ein asynchrones Multiplayer-Brettspiel nach dem Conquer-Prinzip, bei dem verschiedene Gebiete verschiedenen Spielern gehören können. Neue Gebiete können dabei erobert, andere an Mitspieler verloren werden. Der stärkste Spieler ist der, der die meisten Punkte hat. Jeder Spieler kann zu jeder Zeit spielen, ohne das andere Spieler in ihrem eigenen Spiel beeinflusst werden.

Nachfolgend werden die einzelnen Teile des Spiels kurz erläutert und die grundlegenden Spielregeln erklärt.

4.1.1 Das Spielbrett

Das Spielbrett besteht, wie in Abbildung 4.1 zu erkennen, aus hexagonalen Feldern, die aneinander grenzen aber auch einzelne Inseln bilden können. Jedem Feld ist ein Lied

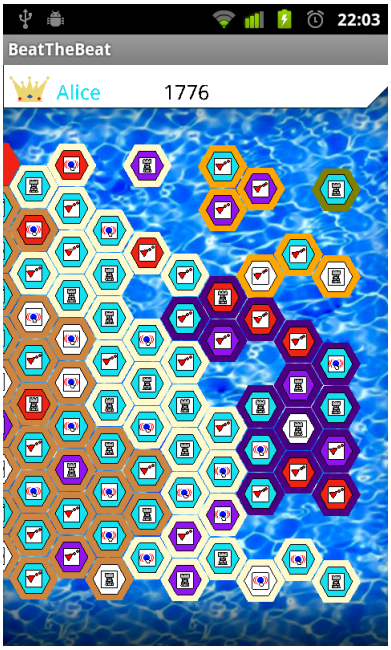


Abbildung 4.1: Das Spielbrett

zugeordnet. Dabei liegen Lieder, die einander ähneln, nah beieinander. Wie die Zuteilung der Lieder zu den einzelnen Feldern funktioniert, wird im Abschnitt 6.3 erläutert. Außerdem wird jedem Feld ein festes Minispiel (4.1.2) zugeordnet. Die Farbe eines Feldes zeigt an, wem das Feld momentan gehört. Jeder Spieler spielt auf seinem eigenen mobilen Endgerät. Dort kann er für sich selbst neue Felder erobern, in dem er ein Feld auswählt, das zugehörige Minispiel spielt und den momentan bestehenden Highscore überbietet. Gelingt dies, so gehört das Feld von nun an ihm und erhält seine Farbe. Andere Spieler müssen nun ihrerseits den Spieler überbieten, um das Feld zu erobern. Der Spieler darf jedes Feld auf dem Spielfeld frei wählen und versuchen es zu erobern. Ein Gesamtspielstand zeigt an, wer momentan führender Spieler auf dem Spielbrett ist. Dabei gibt es für ein einzelnes Feld einen Punkt, für zusammenhängende Gebiete berechnet sich die Punktzahl als $(\#\text{Felder im Gebiet})^2$. Durch geschickte Auswahl der Felder können also eigene Gebiete vergrößert oder zusammenhängende Gebiete anderer Spieler aufgeteilt werden.

4.1.2 Die Minispiele

Wie bereits erläutert, ist jedem Feld ein Minispiel zugeordnet. Diese Zuordnung geschieht beim initialisieren des Spielfeldes zufällig und ist danach fest. Momentan gibt es drei verschiedene Spiele. „TapByTap“ ist ein Rhythmusspiel nach dem Vorbild des erfolgreichen „Guitar Hero“, „Musicfighter“ ein Musik-gesteuertes Shoot'em up Game und „MusicTowerDefense“ greift das bekannte Tower Defense-Konzept auf. Die einzelnen Spiele werden im Abschnitt 7 genau vorgestellt. Allen gemeinsam ist, dass der Spielablauf durch das zugeordnete Lied gesteuert und vorgegeben wird.

Innerhalb eines Spiels gilt es, den bestehenden Highscore des Feldes zu schlagen. Die Punktzahl gilt nur für das aktuelle Feld und ist unabhängig von den oben erwähnten Gesamtspielpunkten. Ein Minispiel dauert genauso lange wie das abgespielte Lied. Anhand einer Fortschrittsanzeige, kann Spieler verfolgen, wie viel Zeit ihm noch bleibt, um die Höchstpunktzahl zu erzielen. Gelingt ihm dies, so gewinnt er das Feld. Er kann nun sofort ein neues Spiel auf einem anderen Feld beginnen oder auf dem gleichen Feld versuchen einen noch besseren Highscore zu erzielen. Letzteres gilt natürlich auch, wenn man das Minispiel verlieren sollte.

Der Ablauf eines Minispiels, z. B. welche Gegner auftauchen wird durch Musikfeatures gesteuert. Dabei gehört zu jedem Lied eine Datei mit den aus dem Lied extrahierten Merkmalen. Dies führt dazu, dass das gleiche Spiel zu verschiedenen Liedern verschiedene Schwierigkeitsgrade und Ausprägungen haben kann.

4.1.3 Mehrspieler

„Beat the Beat“ ist ein Spiel für mehrere Spieler. Die Anzahl der Spieler pro Brett wird hauptsächlich durch die Größe der Karte begrenzt. Theoretisch können beliebig viele Spieler zusammen spielen, jedoch ist eine Begrenzung aus Gründen des Spielspaß sicher empfehlenswert. Jeder Teilnehmer spielt auf seinem eigenen mobilen Endgerät. Synchronisiert wird das Spiel über einen Server, der die Spielstände und Spielbretter verwaltet. Wie bereits erwähnt, ist das Spiel nicht rundenbasiert oder an einen festen Ablauf gebunden. Jeder Spieler kann zu jeder Zeit beliebig viele Minispiele spielen und so seinen Gesamtpunktestand verbessern. Es ist sogar möglich ohne Internetverbindung zu spielen. Eine Synchronisation wird dann erst vorgenommen, wenn wieder eine Datenübertragung möglich ist.

4.2 Reflektion

In diesem Abschnitt sollen kurz die wichtigsten Entscheidungen erläutert und offenen Fragen genannt werden.

Minispiele Zu Beginn der PG wurden mehrere mögliche Spielkonzepte vorgeschlagen und diskutiert. Die Vorteile des umgesetzten Konzepts liegen in der Modularität. Die Möglichkeit mehrere, kleine Minispiele zu entwickeln statt eines großen, zusammenhängenden Projekts, erlaubte es, schnell erste Versuche mit der GameEngine zu unternehmen und auch Ergebnisse zu sehen. Auch konnte so effizient parallel gearbeitet werden, ohne dass sich einzelne Teams behinderten. Das Spiel und auch die darunter liegende Architektur wurden so gestaltet, dass leicht weitere Minispiele hinzugefügt werden können. Dadurch soll der Spielspaß möglichst lange hochgehalten werden. Zudem bieten die Minispiele die Möglichkeit verschiedene Musikmerkmale auszuprobieren und verschieden Ansätze zu verfolgen. Dies ermöglicht den Entwicklern in Zukunft neue Ideen zur Content-Generierung in einem eigenen Minispiel in relativ kurzer Zeit umzusetzen.

Casual Gaming Das Konzept des asynchronen, nicht rundenbasierten Ablaufs hat ebenfalls mehrere Vorteile gegenüber verschiedener Alternativen, wie zum Beispiel eines klassischen Brettspiels, bei dem die Spieler abwechselnd Züge ausführen. Ein Vorteil ist es, dass keine permanente Internetverbindung von Nöten ist, so dass auch Spieler, die über keine mobile Datenflatrate verfügen, aber WLAN nutzen, teilnehmen können. Auch führt die in einigen Gebieten unzureichende Netzabdeckung zu keinen Verzögerungen im Spiel, so dass im schlimmsten Falle andere Spieler lange Wartezeiten in Kauf nehmen müssten. Dies bedeutet des weiteren, dass Spieler die gerne viel Zeit in ein Spiel investieren genauso auf ihre Kosten kommen wie diejenigen, die nur unregelmäßig teilnehmen möchten.

Spielfigur Eine weitere Entscheidung, die die PG getroffen hat, war die gegen eine Spielfigur, die das Ziehen und Auswählen der Felder einschränkt. Ein trivialer Grund ist, dass Verwaltungsaufwand eingespart wird, der das Vorhalten der Positionen der Spieler und vor allem die Überwachung erlaubter Züge beinhalten würde. Wichtiger jedoch ist, dass es dem Spieler möglich sein soll, die Musik zu hören, die gerade zu seiner Stimmung passt und er dementsprechend frei in seiner Entscheidung sein muss, welches Feld er auswählt.

Spielende Eine offene Frage ist die nach einem Spielende. Momentan läuft das Spiel in einer Welt als Endlosspiel. Je nach Spielstärke und Spielintensität kann das jedoch dazu führen, dass einzelne Spieler große Teile des Feldes einnehmen und die Highscores so hoch liegen, dass sie nicht mehr überboten werden können. Zwar sollen die Minispiele so konzipiert sein, dass das Erreichen einer nicht mehr zu überbietenden Punktzahl fast unmöglich ist, jedoch leidet der Spielspaß auch schon unter allgemein sehr hohen Highscores. Aus diesen Gründen ergibt es Sinn, dass Spiel entweder nach einer bestimmten Zeit, oder bei Erreichen eines Spielziels, wie z. B. einer bestimmen Anzahl an besetzten Feldern, neu zu starten. Wie bereits oben erwähnt muss man sich auch Gedanken machen, wie viele Spieler an einem Spiel teilnehmen sollen.

Unter Berücksichtigung dieser Aspekte haben sich die Designentscheidungen der Projektgruppe bewährt und „Beat The Beat“ ist sicherlich ein gelungenes Spiel mit viel Potenzial.

KAPITEL 5

Architektur der Engine

Tim Delbrügger, Dimitri Scheftelowitsch

In diesem Abschnitt soll klar werden, was die Engine leistet, warum sie genau diese Aufgaben übernimmt und wie sie dies umsetzt.

Hierzu sollen zunächst aus den Zielen der PG Anforderungen an die Engine hergeleitet werden, um dann den groben Aufbau mit den unterschiedlichen Projekten zu erläutern. Im nächsten Abschnitt werden die Funktionen der Engine einzeln vorgestellt und in Bezug zu den Anforderungen gesetzt. In einem eigenen Abschnitt werden Tools wie der ShapeViewer, das CI-System und der AIClient vorgestellt. Zum Abschluss wird reflektiert, welche Ziele die Engine erreicht hat und wo noch Verbesserungspotenzial zu finden ist.

5.1 Überblick

Aus den Zielen der Projektgruppe in Kapitel 1 ergeben sich in Abschnitt 5.1.1 einige konkrete Anforderungen. Anschließend wird in Abschnitt 5.1.2 die von der PG gewählte Struktur näher erläutert.

5.1.1 Anforderungen

So ist klar, dass die Engine zum größten Teil plattformunanhägig arbeiten soll und eine vollständige Plattformunabhängigkeit des Spiels ermöglichen muss. Wegen der anfangs sehr unklaren Ziele muss die Engine außerdem von vornherein sehr modular aufgebaut

sein, damit ständig neue Anforderungen umgesetzt werden können, ohne die alte Struktur aufzubrechen. Die von uns gewählte Struktur wird in Abschnitt 5.1.2 erklärt.

Aufgrund der relativ schwachen Hardware der mobilen Geräte sowie der schlechten Umsetzung der Garbage Collection auf Blackberry-Geräten muss die Engine um fast jeden Preis die Erstellung von Objekten zur Laufzeit vermeiden. Auch muss sie kritische Aufgaben wie das Rendering sehr effizient erledigen, damit das Endprodukt noch spielbar bleibt. Zwei Ideen dazu sind das Caching von häufig benutzten Werten und die Verwendung von mehreren Detailstufen bei der Anzeige, wie in Abschnitt 5.2.7 beschrieben.

Zu guter Letzt muss für den Mehrspieler-Aspekt eine Client-Server-Kommunikation ermöglicht werden. Eine Beschreibung unseres Ansatzes findet sich in Abschnitt 5.2.6.

5.1.2 Struktur

Zunächst soll dem Leser eine Übersicht über die Engine gegeben werden. Dazu bietet sich die Einteilung in verschiedene Schichten an, wie in Abbildung 5.1 zu erkennen ist. Danach wird eine funktionale Einteilung in Module gezeigt, welche ein wichtiger Aspekt für die Erweiterbarkeit ist.

Schichten der Engine

Auf der untersten Schicht (grau) liegen die plattformabhängigen Teile, welche jeweils komplett die Eigenheiten der Plattform kapseln. Dazu gehören etwa das Erkennen von Eingaben, Anzeige von Grafiken oder die Ausgabe von Musik. Von dieser Schicht benötigt man pro unterstützter Plattform eine Instanz, hier also eine Blackberry- und eine Android-Implementierung.

Darüber liegt der größte Teil der Engine (blau), welcher die gesamte Logik und High-Level-Funktionalitäten bietet. Hier liegt beispielsweise die gesamte Verwaltung der Grafikobjekte und die Logik des Renderns. Für die konkrete Anzeige eines bestimmten Objekts an einer speziellen Bildschirmposition wird dann auf die Funktionen der Plattformschicht zurückgegriffen. Auch Kollisionen und Picking (siehe 5.2.7) liegen in dieser Schicht.

Die Minigame-Engine (grün) kapselt Gemeinsamkeiten der unterschiedlichen Minispiele, ist also streng genommen nicht Teil der Engine. Dennoch könnte sie in vielen Teilen auch für andere Spiele interessant sein.

Module der Engine

Unabhängig von den Schichten besteht die Engine aus vielen Modulen (siehe Abbildung 5.2), welche zum großen Teil nichts voneinander wissen und die sich zum Teil über mehrere Schichten erstrecken. So wird gewährleistet, dass einzelne Teile geändert oder gar ausgetauscht werden können, ohne dass andere Module beeinflusst werden. Beim Hinzufügen neuer Funktionen wurden im Zweifel eher neue, sehr kleine Module erstellt, als ein bestehendes, schon erfolgreich getestetes und in der Praxis erprobtes Modul zu ändern.

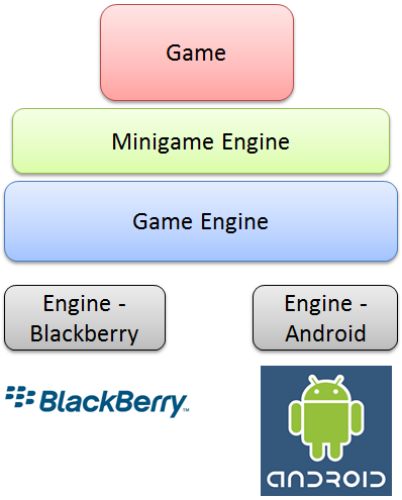


Abbildung 5.1: Schichten der Engine

Bindeglied ist die Klasse `GameEngine`. Sie bildet eine Art Telefonzentrale, über die sich der Benutzer mit den *Managern* verbinden lassen kann. Mithilfe des Entwurfsmusters Singleton wird sichergestellt, dass diese einmalig ist und dass die Funktionen der Engine überall zur Verfügung stehen.

Einige Module, wie etwa der `GameDataManager`, stellen auch Funktionen für andere Module bereit. Der Grafikbereich nutzt diesen beispielsweise, um die Daten einer `Shape`-Datei zu laden.

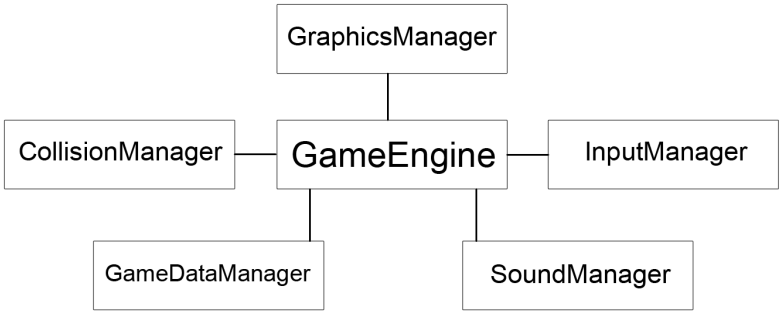


Abbildung 5.2: Module der Engine

Game Loop

Ein sehr wichtiger Bestandteil der Engine ist die *game loop*. Dies ist eine Endlosschleife, welche die Funktionen der Engine in einer festen Reihenfolge immer wieder nacheinander aufruft und somit eine Art „Herzschlag“ des Systems aus Spiel und Engine darstellt.

So wird der `CollisionManager` aufgefordert, nach Kollisionen zwischen Spielobjekten zu suchen, der `InputManager` soll Eingaben an das Spiel weiterreichen und der

`SoundManager` bearbeitet Ereignisse wie das Ende eines Liedes.

Als nächstes wird die Spiellogik ausgeführt, wodurch der nächste Zustand der simulierten Welt erzeugt wird. Dieser neue Zustand wird daraufhin auf dem Bildschirm angezeigt, man spricht hier von *rendering*. Danach beginnt die Schleife wieder von vorn.

Die Anzahl der Durchläufe dieser Schleife pro Sekunde ergibt somit auch die Anzahl der angezeigten Bilder pro Sekunde. Um dem Benutzer eine möglichst flüssige Darstellung von Bewegungen zu bieten, sollte die Schleife also sehr oft durchlaufen werden.

5.2 Funktionen

Nun gehen wir nacheinander auf die einzelnen Module der Engine ein und erläutern die jeweiligen Features sowie deren Umsetzung.

5.2.1 Datenverwaltung

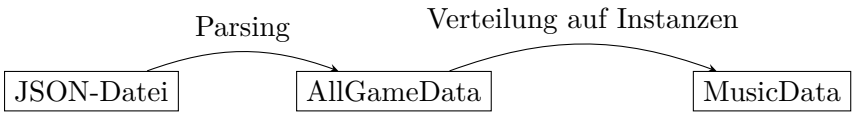


Abbildung 5.3: Die Pipeline für die JSON-Daten am Beispiel der Musikdaten

Wichtige und häufig verwendete Spielelemente wie z.B. Formen, Musikmerkmale und Musikinformationen werden in Konfigurationsdateien abgelegt. Als Format haben wir JSON-Dateien gewählt, da diese sehr intuitiv aufgebaut und gelesen werden können und sie etwa gegenüber XML wesentlich kleiner sind. Die Verarbeitung dieser Konfigurationsdateien findet im `GameData`-Bereich statt, eine grobe Skizze der Verarbeitungsschritte findet sich in Abbildung 5.3. Es gibt insgesamt vier abstrakte Grundtypen von GameData, die beim Auslesen der Konfigurationsdateien erzeugt werden können: `MusicData`, `FeatureData`, `SoundEffectData` und `ShapeData`. Aus diesen abstrakten Typen werden dann später echte Spielelemente erzeugt, aus `MusicData` wird `Music`, aus `ShapeData` `Shape` usw.. Das Auslesen der Konfigurationsdateien und umwandeln in echte Spielelemente findet im `FileManager` statt. Der gesamte `GameData`-Bereich wird allerdings vom `GameDataManager` verwaltet, dieser initialisiert die Spielelemente über den `FileManager` und stellt nach außen hin die Funktion `getByName()` bereit, womit die Engine bestimmte Spielelemente anfordern kann.

Aber nicht nur die Engine benutzt `GameData`-Objekte, sehr wichtig ist dies auch für die einzelnen Spiele. So werden die speziellen Eigenschaften der Spielobjekte alle in JSON-Dateien abgelegt, die dann über den `GameDataManager` ausgelesen werden. Dazu registrieren die Minispiele bei der anfänglichen Initialisierung eigene `GameDataFactory`-Objekte, welche dann die speziellen `GameData`-Objekte des Spiels laden können. Insbesondere beim Balancing ergeben sich so Vorteile, weil nicht immer der Java-Code geändert und neu kompiliert werden muss, nur weil sich die Geschwindigkeit eines Gegners geändert hat.

5.2.2 Grafik

Scenegraph

Jede Grafikengine muss die darzustellenden Objekte in irgendeiner Datenstruktur verwalten. In der aktuellen Spielentwicklung ist dies meist ein Baum, in welchem Bewegung, Rotation oder Skalierung der Eltern stets alle Kinder ebenfalls skaliert, rotiert oder bewegt. Diesen Baum nennt man in der Literatur *scene graph*. Als Referenzimplementierung haben wir dabei die Open-Source-Bibliothek *Ogre3d* angesehen (siehe [Ker10]).

Dazu wurde einerseits ein Teil der Mathematik-Bibliothek (lineare Algebra und einen Teil der trigonometrischen Funktionen) selbst geschrieben, weil das Java 1.3-API diese Funktionalität nicht liefern kann, und dann ein *scene graph* implementiert. Für die Umsetzung einiger trigonometrischer Funktionen wurde dabei zugunsten einer höheren Ausführungsgeschwindigkeit auf statisch berechnete Wertetabellen zurückgegriffen.

Rendering

Bei einem Rendering Schritt müssen nun die im Szenengraph gespeicherten Objekte auf den Bildschirm gezeichnet werden.

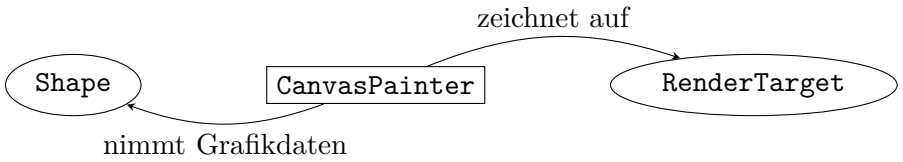


Abbildung 5.4: Die Grafik-Pipeline, im groben Überblick

Ein großer Teil der Rendering-Funktionalität wurde in den plattformunabhängigen Teil der Engine ausgelagert: So ist etwa die Entscheidung, welche Objekte in welcher Reihenfolge auf dem Bildschirm erscheinen, vollkommen unabhängig davon, auf welchem Gerät das Programm letztendlich läuft.

Weiterhin wurde vom plattformspezifischen Verständnis des Koordinatensystems auf dem Bildschirm, wie in modernen 3D-APIs, etwa OpenGL, wegabstrahiert: Das Spiel gibt vor, welcher Ausschnitt aus den „virtuellen“ Koordinaten gezeichnet werden soll: So kann das Spiel etwa angeben, dass nur die Punkte innerhalb des (virtuellen) Rechtecks $((0,0),(3,0),(3,4),(0,4))$ gezeichnet werden; damit kann der Entwickler die Koordinaten nach seinen eigenen Vorstellungen beschreiben. Die einzige Einschränkung unserer API ist die Konvention, dass die positive Richtung der x-Achse rechts und die der y-Achse unten ist.

Im ersten Rendering-Schritt bestimmt die Engine die Knoten im Szenegraphen, die gezeichnet werden müssen (also diejenigen, die im aktuellen Bildausschnitt sichtbar sind) und deren Reihenfolge anhand ihrer z-Koordinate (Tiefe im Bild). Im nächsten

Schritt werden diese Knoten mit dazugehörigen Grafiken (`Shapes`) an die plattformspezifische Rendering-Klasse weitergereicht (siehe Abbildung 5.4); im Falle der BlackBerry-Plattform ist dies der `BlackberryCanvasPainter`. In dieser Klasse werden nun nach der Umrechnung in Bildschirmkoordinaten Blackberry-eigene Methoden verwendet, um die Grafik auf dem Bildschirm des Geräts anzuzeigen.

Koordinatensysteme

Die Engine arbeitet mit zwei Koordinatensystemen: einem „virtuellen“ und einem „realen“. Das „virtuelle“ Koordinatensystem ist das, was für den Benutzer der Engine verfügbar ist. Dem Benutzer steht es frei, sich ein passendes virtuelles Koordinatensystem zu wählen und dieses mit Objekten des Szenegraphen abzubilden.

Das reale Koordinatensystem wird beim Rendering verwendet: Es ist das gerätespezifische Koordinatensystem des Bildschirms. Um zu wissen, welcher Teil der Szene auf dem Bildschirm sichtbar sein sollte, muss der Benutzer zur Laufzeit ein Rechteck im virtuellen Koordinatensystem angeben, welches in jedem Fall sichtbar sein soll. Die Engine berechnet dann die Abbildung zwischen den virtuellen Koordinaten und Pixeln, und benutzt diese Abbildung, um die virtuellen, benutzerspezifischen Koordinaten aus dem Szenegraphen auf die Koordinaten auf dem Bildschirm abzubilden.

Die zweite Anwendung des Koordinatensystems des Bildschirms findet sich bei der Verarbeitung von Benutzereingaben: Die Ereignisse, die an das Bildschirm gebunden sind, wie etwa Touch-Events, müssen in das Koordinatensystem des Szenegraphen interpretiert werden; dazu wird die zu der oben beschriebenen rückwärtige Transformation berechnet und verwendet, um die Pixelwerte auf die virtuellen Koordinaten abzubilden.

Mehrere Detailstufen

Durch das Ändern des dargestellten Bildschirmausschnitts kann das Spiel die Größe der dargestellten Objekte auf dem Bildschirm ändern. Manche Objekte sollen sehr detailliert dargestellt werden, wenn sie gut sichtbar sind. In weiter Entfernung sind diese Details aber nicht sichtbar und somit überflüssig. Deshalb bietet die Engine die Möglichkeit, in Abhängigkeit von der Fläche, die ein Objekt auf dem Bildschirm einnimmt, nur bestimmte Teile anzuzeigen (*levels of detail*). Die Konfiguration geschieht allein über die schon erwähnten JSON-Dateien.

5.2.3 Sound

Das Interface `Sound` beinhaltet die Standard-Funktionen, die für das Interagieren mit Sound-Dateien wichtig sind: Das Abspielen, Beenden und Pausieren einer Sound-Datei, aber auch das Hinzufügen und Entfernen von `SoundListenern`. Mit ihrer Hilfe können wir Ereignisse abfangen, die vom Musik-Player des Gerätes z.B. bei Ende eines Musikstücks ausgelöst werden. Das Interface `Sound` kann entweder einen Sound-Effekt oder ein Musikstück repräsentieren.

Die Klasse `Music` implementiert und erweitert das Interface `Sound` um die Funktion der Verarbeitung von Musikmerkmalen. Für ein Musikstück und ein Musikmerkmal (*feature*) kann man einen `FeatureListener` registrieren. Wird nun zu dem Merkmal an einem bestimmten Zeitpunkt im Lied ein Ereignis erkannt (etwa eine Erhöhung der Lautstärke), so wird der `FeatureListener` darüber benachrichtigt. Wie in Abbildung 5.5 zu sehen ist, testet jedes `Music` Objekt selbst, ob zu einem Merkmal zur Zeit Ereignisse vorliegen. Der `SoundManager` hält lediglich eine Liste aller Musikstücke vor, um diese regelmäßig (mit der *game loop*, siehe 5.1.2) aufzufordern, die `Listener` zu benachrichtigen.

Manchmal ist es erforderlich, dass Spielelemente schon benachrichtigt werden, bevor in der Musik ein bestimmtes Ereignis eintritt. Für diesen Fall wurde die Möglichkeit eines zeitlichen Versatzes (*offset*) eingebaut.

Nähere Informationen zu den Musikmerkmalen finden sich in Kapitel 6.

Beispielhaft sei hier noch einmal das Zusammenspiel der unterschiedlichen Schichten der Engine erklärt: Allgemeine Logik wie das Testen auf vorhandene Ereignisse in Musikmerkmalen ist in der plattformunabhängigen Klasse `Music` untergebracht. Die davon erbende Klasse `AndroidMusic` spezialisiert diese dann mit Zugriffen auf den Mediaplayer von *Android*.

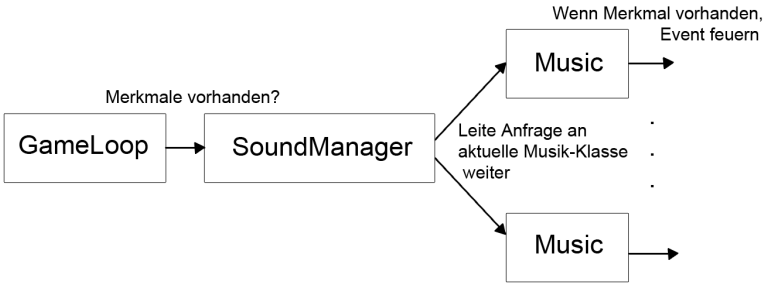


Abbildung 5.5: Merkmalsabfrage

5.2.4 Eingaben

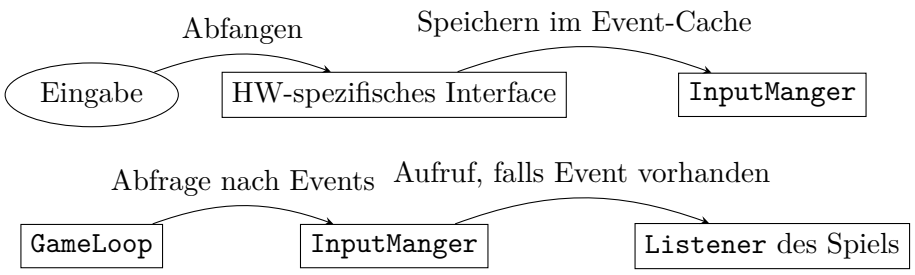


Abbildung 5.6: Die Pipeline für die Verarbeitung der Eingaben

Die Verarbeitung von Eingaben erfolgt über einen Übersetzer aus den plattformabhängigen Konsumenten der möglichen Eingaben in einen plattformunabhängigen Teil, der eine einheitliche Beschreibung für einzelne Eingabearten (etwa: Accelerometer, Tastatur, Touchscreen) hat und diese dann an verschiedene vom Spiel implementierte Listener für verschiedene Teile der Eingabe weiterleitet. Konkret werden die auftretenden Events zunächst im `InputManager` gespeichert, bis die `GameLoop` explizit nach bisher vorgekommenen Events nachgefragt hat. Dann werden alle entsprechenden Listener mit eventspezifischen Angaben aufgerufen.

5.2.5 Grafische Benutzeroberfläche

Um eine grafische Benutzeroberfläche zu erstellen, reichen die bisher vorgestellten Mittel bereits aus: Die Oberfläche kann (wie normale Spielgrafik auch) mit dem Grafik-Teil der Engine gezeichnet werden und auch auf Benutzereingaben kann reagiert werden. Dennoch wurde es im Verlauf der Entwicklung des Spiels immer klarer, dass überall ähnliche Elemente benötigt werden wie etwa Buttons, Textfelder oder beschriftete Icons; diese sollen natürlich überall im Spiel gleich aussehen. Deshalb bekam die Engine noch ein weiteres Modul, die *GUI*. Dies ist im wesentlichen eine Sammlung der verschiedenen *GUI* Elemente, erweitert um größere Komponenten wie etwa ein Fenster mit einer Liste von Anzeigeelementen und automatischer Blätterfunktion. Das Ergebnis lässt sich in Abbildung 5.7 gut erkennen.

Die Realisierung direkt in der Engine erleichtert insbesondere die Einbindung von Funktionen wie dem Aktivieren der Bildschirm-Tastatur des Betriebssystems auf mobilen Geräten, die keine Hardware-Tastatur besitzen.

5.2.6 Netzwerk

Um das Spiel für mehrere Spieler gleichzeitig nutzbar zu machen, wurde eine Netzwerkschicht implementiert. Beim Design der Netzwerkschicht standen zwei grundsätzliche Überlegungen im Vordergrund: Erstens sollte die Netzwerkbibliothek möglichst wenig spielspezifisch sein, zweitens sollte die Kommunikation über das Netzwerk möglichst asynchron ablaufen. Die zweite Forderung erscheint uns im mobilen Spielkontext zwingend, weil man als Entwickler zu jedem Zeitpunkt mit Veränderungen der Netzwerkkonnektivität rechnen muss und Nutzer ständiges Feedback zu ihren Eingaben benötigen.

Konkret besteht die Netzwerkschicht aus mehreren Komponenten. Auf der niedrigsten Ebene, die in der Übersicht in der Abbildung 5.8 nicht abgebildet ist, wird die API zur Netzwerk-Kommunikation der entsprechenden Plattformen in ein gemeinsames Interface gekapselt und zur Übertragung von Daten als Byte-Strings verwendet. Die Ebene darüber implementiert ein Protokoll für den Aufruf von Funktionen: Auf der Client-Seite werden die Aufrufe an den Server geschickt, und der Server führt die aufgerufene Funktion aus. Weiter benutzt die Server-Software zur Speicherung der Daten die Java-Bibliothek Hibernate, die Java-Objekte direkt auf Einträge in der Datenbank (wie etwa MySQL oder Postgres) abbildet, ohne dass es nötig wäre, die Abbildung Java $\leftrightarrow$ SQL selbst zu schreiben.

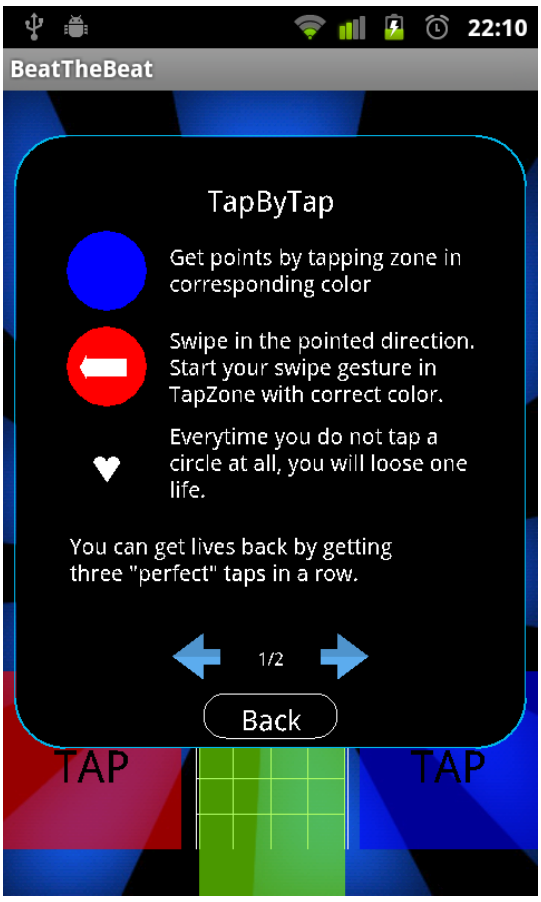


Abbildung 5.7: Fenster der grafischen Benutzeroberfläche mit diversen beschrifteten Icons, einem Button und Blätterfunktion

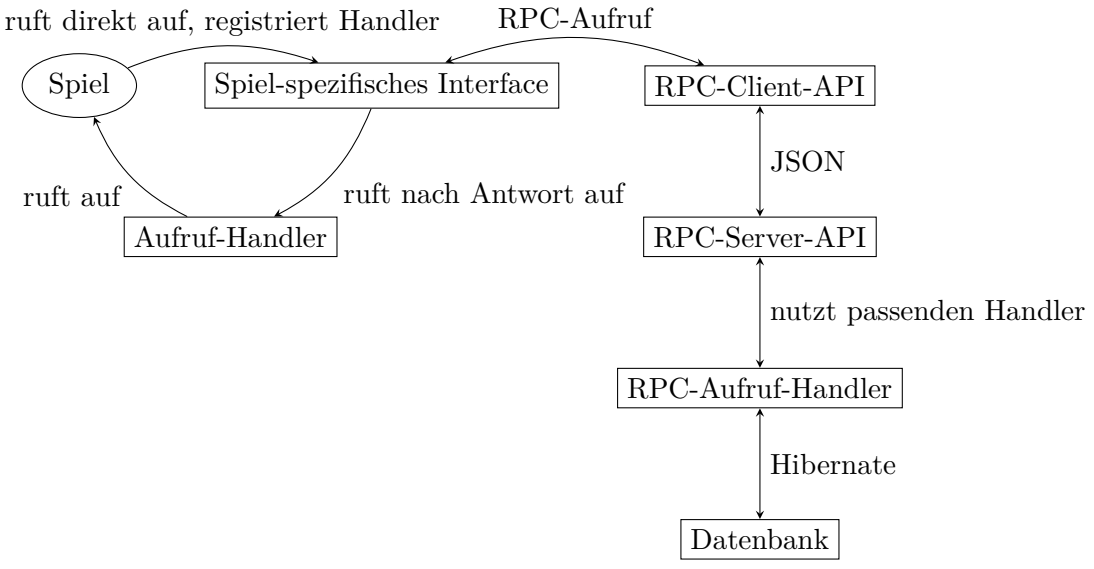


Abbildung 5.8: Die Netzwerk-Architektur

Auf der Seite des Clients steht dem Spiel ein asynchrones Interface zur Verfügung, mit dem man spielspezifische Aufrufe an den Server schicken kann, ohne eine spezielle Syntax zu verwenden: Das Spiel ruft eine „serverseitige“ Funktion als normale Funktion mit Parametern in Java-Datentypen auf und gibt eine (zum Aufruf passende) Funktion an, die aufgerufen werden soll, wenn der Server antwortet.

5.2.7 Sonstiges

Kollisionen

Die Kollisionsbehandlung wurde so konstruiert, dass der Benutzer der API Objekten verschiedene Kollisionsgruppen zuweisen kann und über einen eigenen Listener Kollisionen zwischen beliebigen zwei Gruppen abfangen kann, um sie weiterhin gemäß Spiel-Logik zu behandeln. Wir organisieren Objekte in Kollisionsgruppen, weil das Spiel Kollisionen zwischen Objekten gleicher Klasse auch meistens gleich behandelt; durch die Unterstützung dieses Konzeptes direkt in der API lässt sich Code im Spiel sparen. Der einfache Brute-Force-Ansatz zur Kollisionserkennung, der einfach für alle kollidierbaren Objektpaare die Grenzen der Objekte auf Überschneidungen testet, hat sich bei wenigen kollidierbaren Objektpaaren als schnell genug erwiesen. Gut spielbar auf den von uns benutzten Geräten waren ca. hundert Paare, also etwa zwei Kollisionsgruppen mit zehn und zehn Kollisionsobjekten.

Wichtig für die Performanz ist natürlich nicht nur die Anzahl der getesteten Objektpaare, sondern auch die Geschwindigkeit der Kollisionserkennung bei einem einzelnen Paar. Deshalb wird zunächst die *axis-aligned bounding box* ($AABB$) der beiden Objekte auf Kollisionen verglichen und erst dann ein echter Polygon-Vergleich herangezogen.

Picking

Oft möchte das Spiel wissen, welches Objekt sich gerade an einer bestimmten Stelle am Bildschirm befindet, etwa wenn der Benutzer dort den Bildschirm angetippt hat. Hier hilft der `PickingManager`, wo das Spiel zunächst Objekte als `Pickable` registriert und danach abfragen kann, welches davon sich an einer gegebenen Position befindet. Analog zu den Kollisionen wurde ein simpler Ansatz gewählt, welcher einfach für alle auswählbaren Objekte testet, ob der gegebene Punkt innerhalb ihrer Grenzen liegt. Das erste gefundene Objekt wird dann zurückgegeben.

Pooling

Die Erzeugung von neuen Objekten beeinträchtigt die Effizienz im Spielverlauf stark negativ. Damit war es naheliegend, eine Methode zu bieten, möglichst viele Spieldaten vor dem Spielbeginn zu laden und bis zum Ende des Spiels nicht wieder freizugeben. Benötigt ein Spiel also etwa viele Gegner, so werden diese Objekte nach ihrem Tod nicht zerstört, sondern beiseite gelegt und wiederverwendet, sobald neue Gegner erscheinen sollen. Damit diese Funktionalität nicht unnötig im Code jedes Spiels wiederholt implementiert werden muss, wurde sie ein Teil der Engine-API.

Caching

Zur Optimierung des Speicherverbrauchs (und damit auch des Laufzeitverhaltens) sollte noch weiter vermieden werden, neue Objekte zu erzeugen. Leider ist bei der Java-Entwicklung häufig das genaue Gegenteil der Fall. Insbesondere bei nicht veränderlichen Datentypen war es sinnvoll, diese Objekte wiederzuverwenden, anstatt für jedes neue Vorkommen auch neue Instanzen anzulegen. Beispielsweise für Objekte vom Typ `Float`, die eine als Java-Objekt gekapselte (auch *boxed* genannt) Version von 32-Bit-Gleitkommazahlen sind, wurde ein Cache erzeugt, der diese Objekte vorher berechnet und speichert. Da nicht alle Gleitkomma-Objekte gespeichert werden konnten, musste ein Kompromiss auf Kosten der Genauigkeit eingegangen werden: Die Genauigkeit ist $\pm\frac{1}{100}$ für Zahlen im Intervall $[0, 1]$ und $\pm\frac{1}{4}$ im Intervall $[1, 50]$. `Float`-Objekte im Rest des Zahlenbereichs wurden hinreichend selten benutzt, dass man sich auf den Garbage Collector verlassen konnte.

Ab Version 1.5 bietet Java eine ähnliche Funktionalität übrigens mittels der Methode *Float.valueOf(float)*[1] selbst an. Wie bereits in Abschnitt 2 erwähnt, kann die Projektgruppe herauf jedoch nicht zugreifen.

5.3 Tools

Für die Verwendung einer Engine sind Tools und Wartbarkeit sehr wichtig, weshalb nun noch auf ShapeViewer, AIClient und CI-System eingegangen wird.

5.3.1 ShapeViewer

Der ShapeViewer ist ein Programm für Android, mit dem man die JSON-Grafiken des Spiels betrachten kann. Geplant als Proof-of-Concept, erwies sich der ShapeViewer nützlich bei dem Design der Grafiken im Spiel. Ursprünglich war der ShapeViewer der erste Schritt in Richtung des Android-Ports gewesen; damit konnte man das Grafik- und das Dateien-Subsystem für Android testen.

5.3.2 Continuous Integration System

Mit der Einführung des Servers schien es ratsam, eine Test-Infrastruktur aufzubauen, mit der die jeweils aktuelle Version der Engine schnell eingesetzt und überprüft werden kann. Aus dieser Motivation heraus wurde *Jenkins* auf dem PG-Server aufgesetzt. Dieses Programm beobachtet die Versionsverwaltung und kann somit bei Aktualisierungen im Quellcode direkt den Server mit der neuen Version starten. So stand der Projektgruppe stets die aktuelle Version des Servers zur Verfügung. Zusätzlich führt *Jenkins* automatisch die bestehenden JUnit-Tests aus.

[1] Siehe http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/Float.html

5.3.3 AIClient

Zum Testen der Netzwerk-Infrastruktur, insbesondere der Interaktion verschiedener Clients mit dem Server, war es notwendig, mehrere Clients auf dem PG-Server nebeneinander spielen zu lassen. Aus vielen verschiedenen Gründen wurde die Unterstützung von verschiedenen Spielernamen im eigentlichen Spiel nur spät implementiert. Deswegen wurde ein kleiner „Client“ entwickelt, dessen einzige Aufgabe es war, sich mit einem eindeutig verschiedenen Benutzernamen auf dem Spielserver anzumelden und den Spielprozess zu imitieren. Der Spielprozess wurde insofern imitiert, als dass auf zufälligen Spielfelder der Highscore erhöht wurde und diese Felder in den Besitz der KI übergingen. Damit wurde ein Eindruck erzeugt, dass im Spiel permanent etwas passiert; das Feld hat sich regelmäßig geändert und dies wurde schnell auf Endgeräten sichtbar.

5.4 Reflektion

Im Rückblick lief die Entwicklung der Engine im Großen und Ganzen aus der Perspektive des Engine-Teams sehr gut. Hervorzuheben sind die recht hohe Geschwindigkeit der Entwicklung und die Qualität des Codes: Durch eine umfassende, transparente Planung der Entwicklungsprozesses war es nie notwendig, große Teile des Codes neu zu implementieren oder auf neue Interfaces anzupassen, weil die Anforderungen an den Code durchdacht waren. Ein Indiz hierfür ist auch, dass die Engine innerhalb eines einzigen Sprints auf die Android Plattform portiert werden konnte.

Allerdings gab es auch einige Problemstellen bei der Entwicklung, etwa bei der Kommunikation mit den Benutzern der Engine. So waren am Anfang der Entwicklung die Prioritäten nicht genau geklärt und neue Features der Engine wurden nicht so schnell eingesetzt wie erhofft. Es wäre sinnvoller gewesen, die neuen Features in schriftlicher Form festzuhalten und etwas deutlicher zu kommunizieren (etwa in Form einer Rundmail, wie es später praktiziert wurde).

Ein weiterer Nachteil des ganzen Entwicklungsprozesses, der aber nicht von der Projektgruppe als solche verschuldet ist, ist die Tatsache, dass die Engine für die Java-Version 1.3 geschrieben wurde. Damit ist die Engine nicht bequem in neueren Umgebungen einsetzbar; es wäre sinnvoll, einen großen Teil des äußeren Interfaces auf Features der neueren Java-Plattformen (etwa 1.5) aufzurüsten.

Insgesamt lässt sich feststellen, dass die Entwicklung der Engine in sehr guten Bedingungen abgelaufen ist und wenig an dem Entwicklungsprozess auszusetzen ist. Rückblickend könnte man sich noch mehr Produktivität wünschen, damit mehr Features möglich wären, aber auch das Ergebnis, welches jetzt präsentiert werden kann, ist ein großer Erfolg.

KAPITEL 6

Arbeit mit musikalischen Merkmalen

Jannic Hartwecker, Nils Vortmeier

Aufbau und Ablauf des Gesamtspiels und der einzelnen Minispiele sollen in hohem Maße von den Eigenschaften der verwendeten Musik abhängen: Das Spielbrett arrangiert die zur Verfügung stehende Musiksammlung, so dass ähnliche Lieder gruppiert werden; Aktionen und Ereignisse in den Minispielen werden von den Eigenschaften des zugehörigen Musikstücks gesteuert.

6.1 Generelles Vorgehen

Als ersten Schritt zur Umsetzung der genannten Anforderungen benutzen wir das Programm AMUSE (siehe [VTB10]) zur Merkmalsextraktion. Einige der enthaltenen Merkmale sind in [TVE08] erklärt, weitere Informationen enthält [Set98]. Dieses Programm liefert pro Musikstück 78 Dateien von Merkmalen, aus denen im nächsten Schritt die für das Spiel wichtigen Informationen ausgewählt und in eine für die Spiele-Engine lesbare Form gebracht werden müssen. Zu diesem Zweck hat die Projektgruppe das Tool „FeatureExtractor“ entwickelt. Es liest die Musiksammlung, die von AMUSE bereitgestellten Merkmalsdateien und generiert die für die Engine nötigen Informationsdateien zu vorhandener Musik, den jeweiligen Merkmalen und dem Aufbau des Spielbretts.

Die Datei `music_data_list.json` beinhaltet für jedes Musikstück die Informationen zu Künstler und Titel, die aus den jeweiligen MP3-Tags ermittelt werden, sowie die Pfadangabe zur MP3-Datei und zur generierten Feature-Datei. Diese zu jedem Musikstück erstellte Feature-Datei besteht nicht direkt aus den von AMUSE erhaltenen Daten. Vielmehr werden sie weiter verarbeitet und erst dann exportiert. FeatureExtractor stellt zu diesem Zweck Methoden zur Glättung, Filterung, Normalisierung und Ableitung der

Daten zur Verfügung, durch Kombination von Daten werden neue Merkmale abgeleitet, die als neue Klassen eingebunden werden können. Der folgende Abschnitt 6.2 erläutert die Konstruktion dieser Merkmale. Im Anschluss in Abschnitt 6.3 beschreiben wir die Konstruktion des Spielbretts.

6.2 Konstruktion von neuen Merkmalen

Für eine reibungslose Entwicklung sind praktische Software-Werkzeuge unerlässlich. Ein solches Software-Werkzeug ist „FeatureVisualizer“, mit dessen Hilfe die rein numerischen Musik-Features visuell aufbereitet werden und damit der Entwickler unterstützt wird, neue Merkmale zusammenzustellen. FeatureVisualizer setzt auf FeatureExtractor auf und stellt eine grafische Oberfläche zu einigen seiner Funktionen zur Verfügung.

6.2.1 FeatureVisualizer

FeatureVisualizer ermöglicht es dem Benutzer Lieder im MP3-Format und die zugehörigen Merkmalsdateien zu laden. Dem Benutzer ist es hierbei selbst überlassen, welche und wie viele der von FeatureExtractor bereitgestellten Merkmale er in der GUI darstellen möchte.

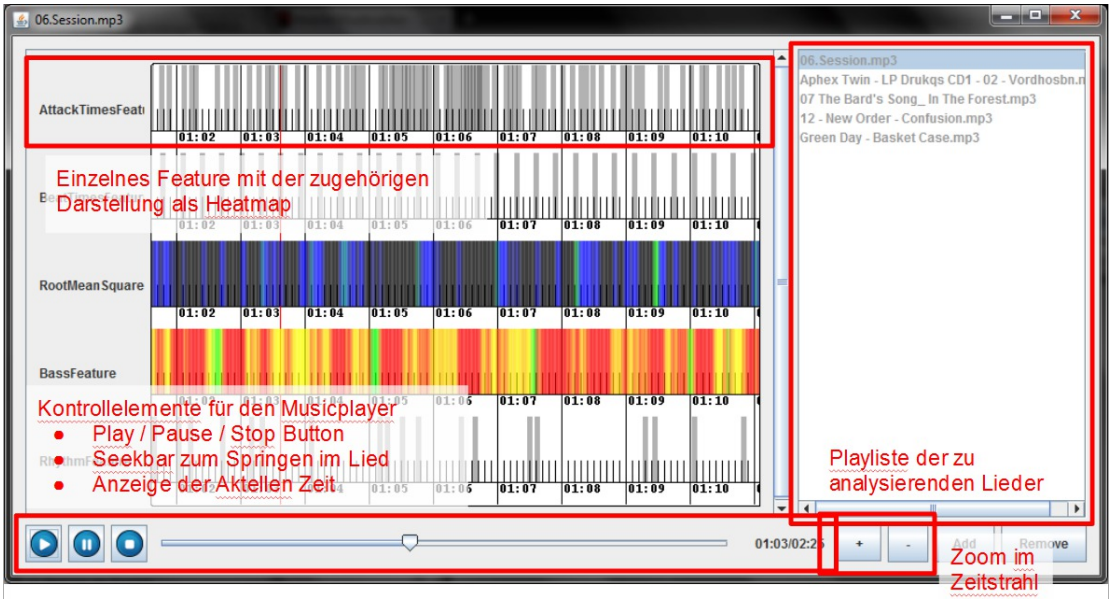


Abbildung 6.1: FeatureVisualizer: Hauptfenster

Die grafische Oberfläche (6.1) wurde mit den grundlegenden Merkmalen zur Musikwiedergabe ausgestattet (Abspielen, Pause, Stopp, Positionsanzeige und (Rest-)Zeit).

Manche Merkmale (6.2) besitzen verschiedene Wertreihen, die selektiert werden müssen und in der rechten Liste mit Zahlen gekennzeichnet sind. Bei der Auswahl können für jedes Merkmal Parameter ausgewählt werden. Durch Mehrfachauswahl eines Merkmals

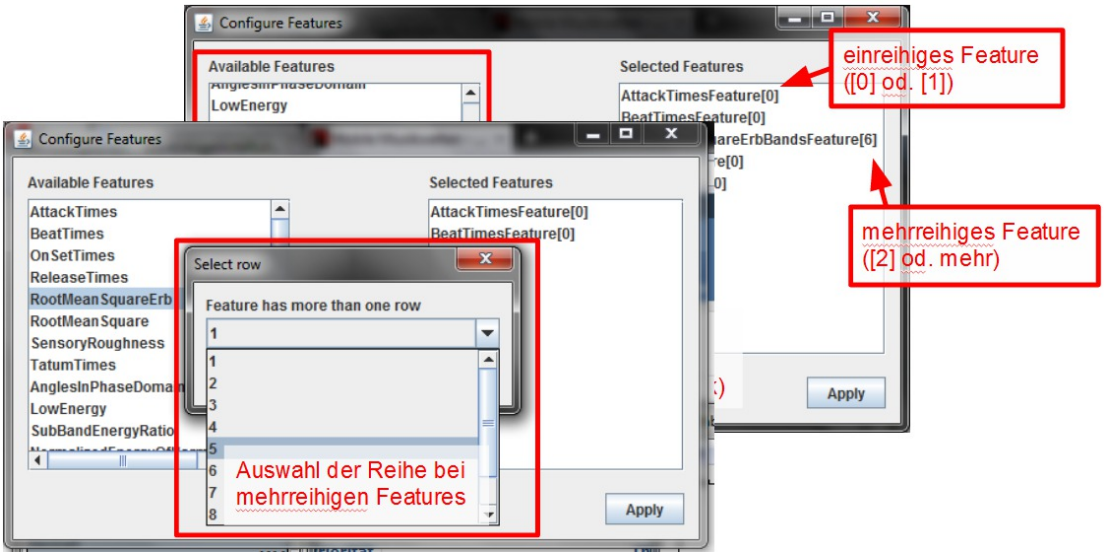


Abbildung 6.2: FeatureVisualizer: Liste von Features

ist so auch ein Vergleich zwischen unterschiedlichen Konfigurationen dieses Merkmals möglich.

Die Konfiguration (6.3) umfasst die folgenden Möglichkeiten:

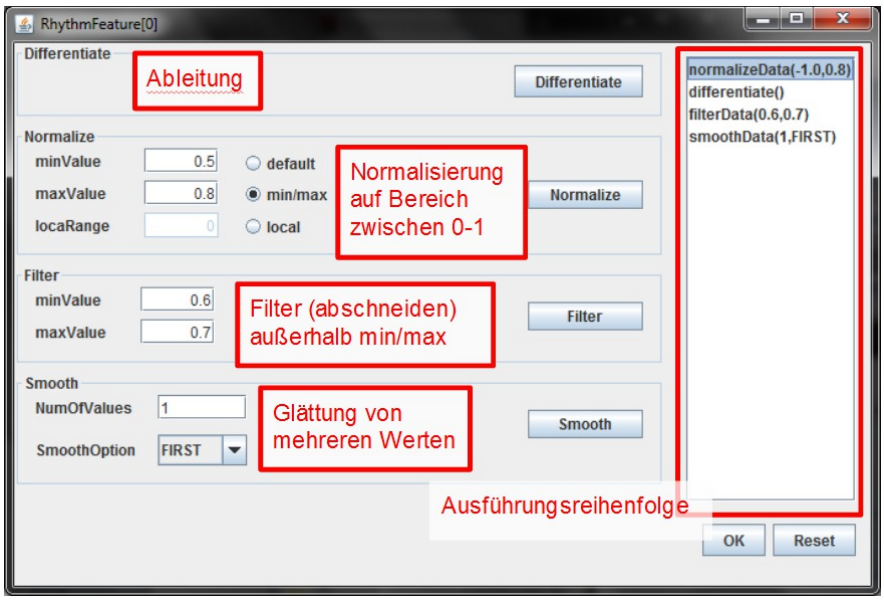


Abbildung 6.3: FeatureVisualizer: Konfiguration

Differentiate Ableitung über die gegebenen Werte

Normalize Normierung der Werte auf drei Arten

default zwischen 0.0 und 1.0 bezüglich der bestehenden Minima und Maxima

minMax zwischen 0.0 und 1.0 bezüglich `minValue` und `maxValue`

local zwischen 0.0 und 1.0 bezüglich der lokalen Minima und Maxima im Bereich `localRange`

Filter Nach der Filterung existieren nur Werte im Intervall zwischen `minValue` und `maxValue`

Smooth Glättung von je `NumOfValues` Werten. Mit `SmoothOption` werden die Verschmelzungen konfiguriert

FIRST wähle den ersten Wert als Repräsentant des Blocks

MIN wähle den kleinsten Wert als Repräsentant des Blocks

MAX wähle den größten Wert als Repräsentant des Blocks

MEDIAN wähle den Median als Repräsentant des Blocks

MEAN wähle den Mittelwert als Repräsentant des Blocks

Durch Klicken der entsprechenden Schaltfläche werden die Anpassungen in die Konfigurationsliste übernommen und mit einem Klick auf `Apply` in der Reihenfolge der Liste angewendet. Insbesondere ist auch eine mehrfache Nutzung der selben Anpassung möglich (z. B. mehrfache Ableitung).

Die so gewählten Merkmale erscheinen nun im Hauptfenster. Horizontal werden so die Merkmalswerte für den jeweiligen Zeitpunkt dargestellt und farbig auf einer Skala von 0.0 (schwarz) bis 1.0 (rot) gekennzeichnet.

Für eine bessere Übersicht kann der Benutzer mit Hilfe der Knöpfe `+` und `-` in die Merkmale hineinzoomen bzw. herauszoomen und so den sichtbaren Ausschnitt verkleinern oder vergrößern.

Der Benutzer kann zwischen verschiedenen Liedern wechseln, die ihm in einer Wiedergabeliste auf der rechten Seite der grafischen Oberfläche präsentiert werden. Hierbei bleiben die ausgewählten Merkmale für das aktuell gewählte Lied bestehen, wodurch ein Vergleich der Merkmale zwischen verschiedenen Lieder möglich wird, ohne diese erneut konfigurieren zu müssen.

6.2.2 Einsatz des FeatureVisualizers

Durch experimentellen Einsatz (6.4) sind im Rahmen der Projektgruppe mehrere Features „generiert“ worden: `RhythmusFeature`, `BassFeature` und `SopranoFeature`. Das Rhythmus-Feature ist ereignisbasiert und soll ausschlagen, wenn im Musikstück ein neuer Ton der Hauptmelodie gespielt wird, idealerweise korrespondiert dieses Feature also mit dem Rhythmus, den man mitklatschen oder mitklopfen würde. Die Umsetzung kann natürlich nur eine Näherung an dieses Ziel sein.

Das Rhythmus-Feature benutzt das RootMeanSquare-Feature, also die Energie des Stückes in den jeweiligen Zeitfenstern. So ergibt sich der Wert für ein Fenster, in unserem Fall bestehend aus $N = 512$ Amplitudenwerten, aus der Wurzel der Summe der Amplitudenquadrate.

$$x_{RMS} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} x^2} \tag{6.1}$$

Die Werte werden in einem lokalen Bereich von 200 Fenstern normalisiert: Für jedes Fenster werden die vorigen und folgenden 200 Fenster betrachtet, der eigentliche Fensterwert wird bezüglich der dort gefundenen Minima und Maxima normalisiert. Dadurch sollen sehr leise und sehr laute Abschnitte innerhalb eines Liedes aneinander angeglichen werden. Da wir uns für die Änderung der Energie interessieren, wird dann die Ableitung des Werte gebildet und das Ergebnis wiederum normalisiert. Zu jedem Fenster, dessen berechneter Wert größer als 0.55 ist, wird ein Rhythmus-Event ausgelöst. In einer Nachbearbeitung werden Ereignisse gefiltert, so dass sie einen Mindestabstand von 0.1 Sekunden haben.

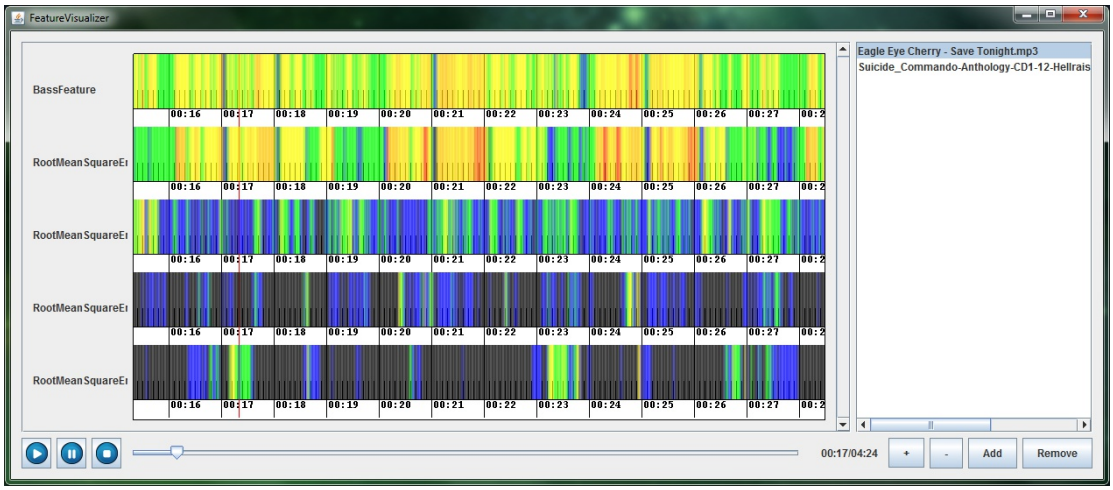


Abbildung 6.4: FeatureVisualizer: `BassFeature` und seine Komponenten

Das Bass-Feature soll zu einem Zeitpunkt die Intensität des Basses beziehungsweise der tiefen Töne angeben. Dazu wurde das `RootMeanSquareErbBandsFeature` genutzt, welches ein mehrbändiges Feature auf verschiedenen Frequenzbereichen ist und auf [MG96] basiert. Es wurden die ersten vier Bänder gewählt (die vier tiefsten Frequenzen). Um vor allem die tiefsten Frequenzen zu berücksichtigen wurden die einzelnen Bänder verschieden gewichtet (6.1).

Das Soprano-Feature ist sehr ähnlich zum Bass-Feature, soll aber die Intensität der hohen Töne angeben. Die Bänder 4 bis 10 wurden dazu in dieser Reihenfolge mit den Werten $0.1, 0.3, 0.6, 0.9, 1, 1, 1$ gewichtet.

Band	Gewichtung
1	1
2	1
3	0.7
4	0.1

Tabelle 6.1: Gewichtung der `RootMeanSquare` Bänder

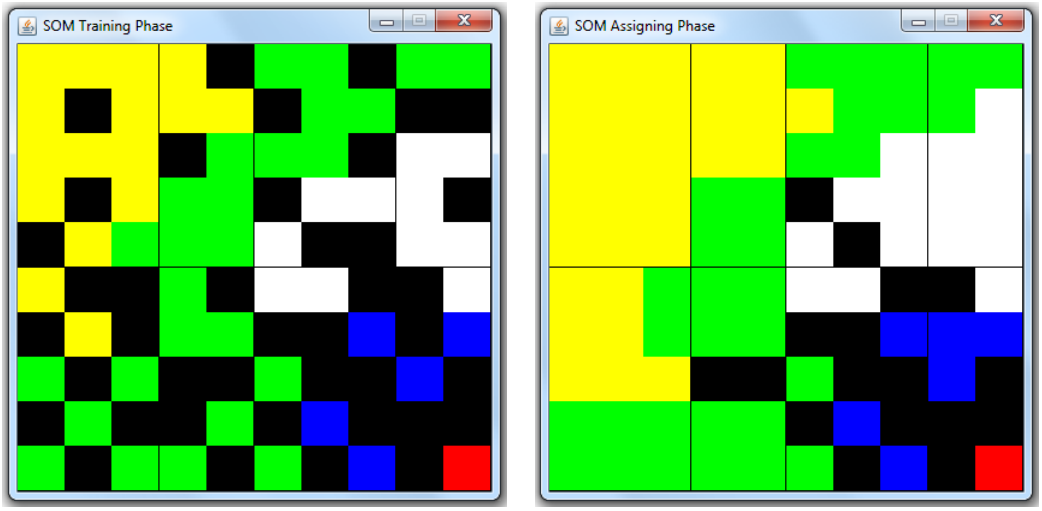
6.3 Konstruktion des Spielbretts

Das Spielbrett soll die auf dem Gerät befindliche Musiksammlung möglichst intuitiv darstellen, ähnliche Musik also gruppieren. Der von der Projektgruppe gewählte Ansatz[1] repräsentiert jedes Lied durch einen Merkmalsvektor, bestehend aus Mittelwert, Standardabweichung, Skewness und Kurtosis des RootMeanSquare-Features, sowie Mittelwert und Standardabweichung seiner Ableitung, jeweils über das ganze Lied berechnet. Als weitere Komponente verwenden wir den Winkel im Phasenraum [MM05]. Als Mittel zur Dimensionsreduktion wurde eine Self-Organizing Map (SOM) gewählt[2]. Unsere quadratische SOM besteht aus gerade so vielen Neuronen, dass es mindestens so viele Neuronen wie Lieder gibt. Im ersten Schritt der Berechnung wird die SOM mit den berechneten Merkmalsvektoren trainiert: Jedes Neuron besitzt einen Gewichtsvektor, der die selbe Dimension wie ein Merkmalsvektor hat, der zufällig über dem Raum initialisiert wird, der durch die minimalen und maximalen Einträge der Merkmalsvektoren gebildet wird. Zu jedem Lied wird das Neuron bestimmt, dessen Gewicht den kleinsten euklidischen Abstand zu seinem Merkmalsvektor hat. Dieses Neuron wird best matching unit, kurz BMU genannt. In einem bestimmten Radius um das BMU werden die Gewichte in Richtung des Merkmalsvektors geändert, wobei der Radius nach jeder Iteration verkleinert wird und die Stärke der Veränderung von einer globalen Lernrate und dem Abstand des Neurons zum BMU abhängt.

Nach einigen Iterationen wird das Training abgebrochen und jedem Lied wird ein Neuron zugewiesen. Dazu wird zu jedem Lied (bzw. dessen Merkmalsvektor) das passende BMU bestimmt. Mehrere Lieder können auf ein BMU fallen, deshalb wird zuerst das Lied zugewiesen, dessen Abstand zu seinem BMU maximal ist. So soll der auftretende Fehler beschränkt werden. Das gewählte Neuron steht für die anderen Lieder als BMU nicht mehr zur Verfügung.

In Abbildung 6.5 ist die Visualisierung der beiden Schritte zu sehen. Unabhängig von der SOM-Berechnung werden die Merkmalsvektoren mit einem k-Means-Clustering in fünf Cluster aufgeteilt, um Gebiete auf dem Spielbrett zu erhalten. Diese Cluster sind in der Abbildung durch Farben repräsentiert, schwarze Felder sind nicht von Liedern belegt. Es fällt auf, dass auch in Abbildung 6.5a keine Lieder verschiedener Cluster auf ein Neuron abgebildet werden, da man sonst eine Farbmischung beobachten würde. Die Dimensionsreduktion erhält also Nähe und Distanz zwischen den Repräsentanten der Lieder. Auch in Abbildung 6.5b nach Schritt 2 sieht man, dass Lieder eines Clusters ein Gebiet bilden. Somit ist die Darstellung der Musiksammlung auf dem Spielbrett

[1] unter Anregung durch `http://christianpeccei.com/projects/musicmap/`
[2] Umsetzung basierend auf `http://www.ai-junkie.com/ann/som/som1.html`



(a) Schritt 1: Training der SOM (b) Schritt 2: Verteilung der Lieder

Abbildung 6.5: Visualisierung der Phasen zur Verteilung der Lieder auf dem Spielbrett

tatsächlich nicht zufällig, sondern folgt der Ähnlichkeit der berechneten Merkmalsvektoren. Da Lieder auf dem Spielbrett nicht als Quadrate, sondern als Hexagons angezeigt werden, wird für die Darstellung jede zweite Reihe von Liedern um eine halbe Reihe nach unten geschoben.

KAPITEL 7

Entwicklung des Spiels

Ümit Güler, Matthias Kuchem, Jan Lahni

7.1 Tap by Tap

Tap by Tap ist ein klassisches Rhythmusspiel, welches man auch in abgewandelter Form auf anderen Plattformen wiederfinden kann. In unserer Variante des Spiels erscheinen von der oberen Seite des Bildschirms rote und blaue Kreise. Das Erscheinen der Kreise ist abhängig vom Rhythmusmerkmal des aktuellen Songs, welches im Hintergrund wiedergegeben wird. Die Kreise erscheinen zeitlich versetzt vor dem eigentlichen Zeitpunkt im Song, da die Kreise einen gewissen Weg zurücklegen müssen, bevor sie in der Hitzone sind. Das Rhythmusmerkmal steuert die Kreise so, dass ein Takt genau dann vorliegt wenn sich ein Kreis genau mittig in der Hitzone befindet.

Die Aufgabe des Spielers ist nun, alle erscheinenden Kreise möglichst im richtigen Takt zu tappen. Entsprechend der zwei Farben, welche die Kreise annehmen können, befindet sich jeweils unten links für rote Kreise und unten rechts für blaue Kreise die jeweilige Tapzone. Unter dem Begriff „tappen“ verstehen wir nun das Berühren der entsprechenden Tapzone im möglichst passendem Zeitpunkt. Die Farbe des Kreises gibt vor, welche Tapzone getappt werden muss. Um einen möglichst hohen Spielspassfaktor anbieten zu können, wurden auch Kreise eingeführt, die zusätzlich zu ihrer Farbe auch eine Pfeilrichtung vorgeben. Bei solchen Kreisen muss man statt des Tappens eine Bewegung ausgehend von der richtigen Tapzone in Richtung des Pfeiles auf dem Bildschirm machen.

Die Spielschwierigkeit steigt schrittweise. Der Spieler hat insgesamt drei Leben. Falls ein Kreis nicht rechtzeitig innerhalb der Hitzone getappt wurde, wird dem Spieler ein

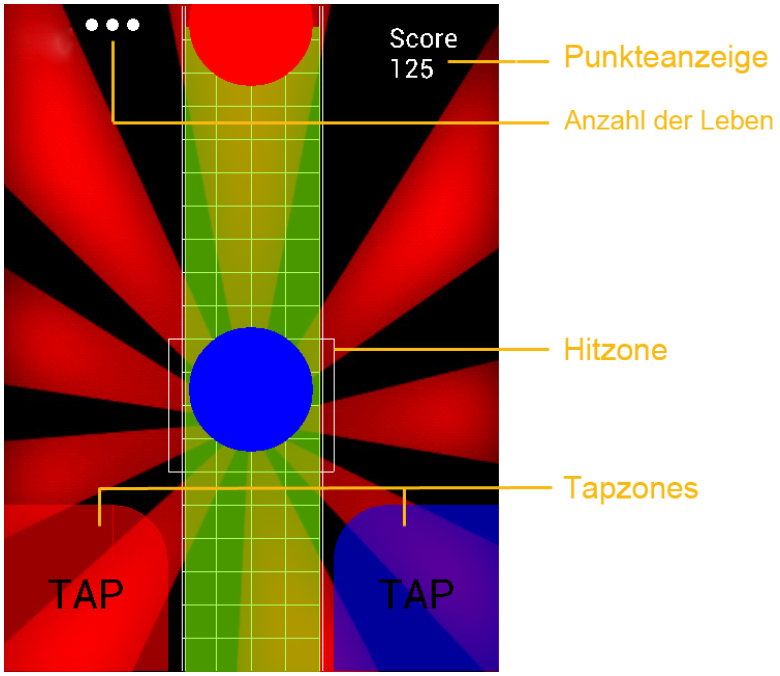


Abbildung 7.1: Tap by Tap - Spielelemente

Leben abgezogen. Ziel ist es, durch eine hohe Trefferquote des Taktes so viele Punkte zu erreichen wie möglich.

Da nicht jeder Spieler gleich gut ist und das Tappen im richtigen Takt manchmal schwierig ist, gibt es auch Punkte, wenn kurz vor oder nach dem Takt getappt wurde. Allerdings muss sich der Kreis dabei immer noch teilweise in der Hitzone befinden. Der Spieler wird beim Erzielen der Punkte per Texteinblendung motiviert. Die Punkteverteilung von sehr ungenau bis zum genauen Tappen geschieht wie folgt:10 Punkte *OH-NO*, 125 Punkte *BETTER*, 250 Punkte *ALMOST* und 500 Punkte *PERFECT*, für einen präzisen Taktschlag. Falls man drei Mal in Reihe 500 Punkte erzielt, erhält man zusätzlich 1000 Punkte als Bonus und ein zusätzliches Leben. So kann man verlorene Leben wieder einholen, um bis zum Spielende durchzuhalten. Falls man jedoch in der falschen Tapzone tappt, werden dem Spieler 10 Punkte abgezogen.

Das Spiel ist zu Ende, wenn der Spieler kein Leben mehr zur Verfügung hat oder wenn der zugehörige Song zu Ende ist.

7.2 TowerDefense

Bei dem Spiel *TowerDefense* (siehe auch Abb. 7.2) erscheinen Gegner am Start eines Pfades und versuchen, zum Ende zu gelangen. Der Spieler muss durch Bauen von Türmen, die auf die Gegner schießen, dies verhindern. Die Türme schießen dabei in Abhängigkeit der laufenden Musik (durch Anpassung von Feuerrate und Schussstärke).

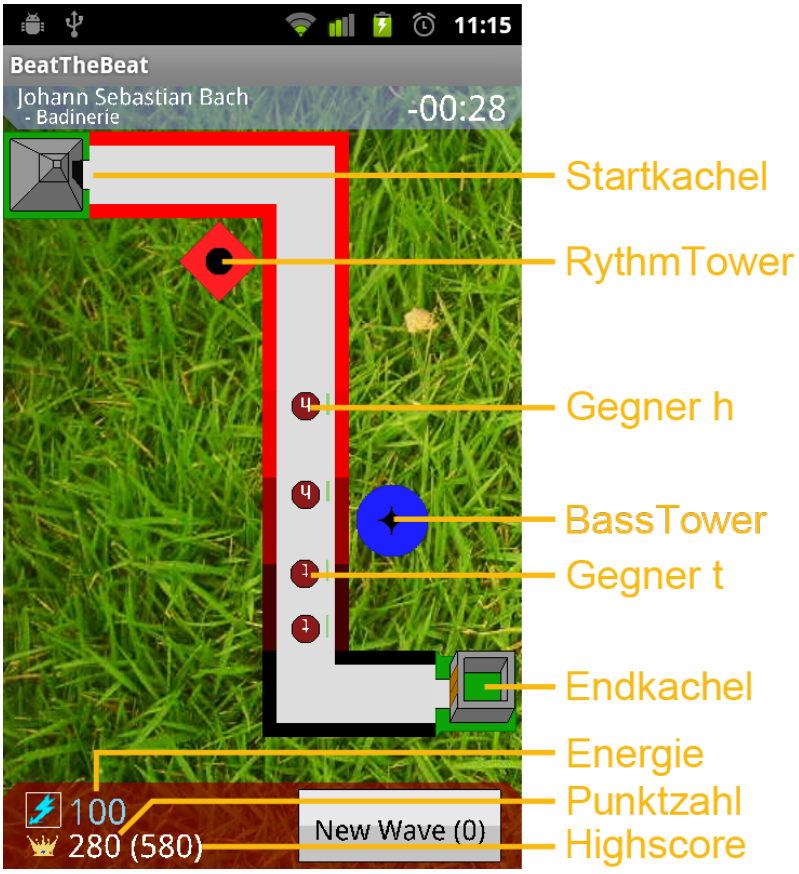


Abbildung 7.2: Towerdefense Spielelemente

7.2.1 Spielfeld

Das Spielfeld ist gerastert in 6×7 Kacheln. Es gibt auf jedem Spielfeld eine Startkachel (die Höhle der Gegner) und Endkachel (die zu verteidigende Burg). Diese sind über Wegkacheln miteinander verbunden und umgeben von Grasfeldern.

7.2.2 Gegner

Es gibt insgesamt 6 Gegnertypen. Dabei erscheint jeder als roter Kreis mit einem der 6 Zeichen aus dem String „Beat the Beat“. Alle Gegnertypen haben eine feste Anzahl Leben, Punkte und Energie. Die Punkte werden dem Spieler gutgeschrieben, wenn er diese Einheit tötet. Wenn der Gegner das Ende des Weges erreicht, wird das 5-fache dieses Werts von der aktuellen Punktzahl abgezogen. Zusätzlich erhält der Spieler Energie mit dem Töten der Einheit, um damit weitere Türme zu bauen oder diese besser auszurüsten. Beim Balancing wurde darauf geachtet, dass Gegnertypen mit den häufiger vorkommenden Buchstaben (z. B.„e“), weniger stark sind, als die weinger Vorkommenden (z. B.„h“).

7.2.3 Gegnerwellen

Die Gegner erscheinen in Wellen von je n Gegnern vom gleichen Typ. Die erste Welle besteht aus Gegnern vom Typ „B“, die zweite vom Typ „e“, die dritte vom Typ „a“, usw. Zusammen bilden die Gegnertypen den String „Beat the Beat the Beat the Beat...“. Zu Beginn des Spiels gilt $n = 2$ (d. h. die erste Welle enthält 2 Gegner). Nach jeweils 9 Wellen (entsprechend dem String „Beat the “) wird n um 1 inkrementiert. Außerdem werden dabei die Lebenspunkte der Gegner vervielfacht (die Gegner werden mit der Zeit schwerer). Wenn sich kein Gegner mehr auf dem Weg befindet wird automatisch eine neue Welle geschickt. Der Spieler kann aber auch manuell neue Wellen anfordern. Er sollte also versuchen möglichst viele Wellen zu starten (um möglichst viele Punkte zu erhalten), ohne dabei auch nur eine Einheit durchzulassen.

7.2.4 Türme

Die Türme werden auf den Grasfeldern gebaut und erreichen mit ihren Schüssen alle direkt angrenzenden Felder (auch diagonal). Sie haben alle die gleichen Basiskosten (Energie) und können ausgebaut werden, um die Stärke und die Reichweite zu erhöhen. Der Spieler startet mit genug Energie für einen Turm. Es gibt vier verschiedene Turmtypen:

RhythmTower schießt im Rhythmus der Musik, mit konstantem Schaden.

SensoryRoughnessTower schießt regelmäßig mit festen Basisschaden und zusätzlichem Schaden, wenn das Lied gerade unruhig ist.

BassTower schießt regelmäßig mit festem Basisschaden und zusätzlichem Schaden, wenn des Lied gerade basslastig ist.

SopranoTower bildet das Gegenstück zum BassTower.

7.3 MusicFighter

In dem Spiel MusicFighter (Abb. 7.3) erscheinen Gegner in Abhängigkeit von dem ausgewählten Musikstück. Der Spieler, repräsentiert durch eine Spielfigur (Abb. 7.4a), muss die Gegner abschießen um Punkte zu erhalten, möglichst ohne selbst getroffen zu werden oder mit diesen zu kollidieren. Das Ziel des Spiels ist es, einen möglichst hohen Punktestand zu erreichen bzw. einen bestehenden Punktestand zu schlagen, bevor das Lied und damit das Spiel zu Ende ist. Die erreichte Gesamtpunktzahl, der momentane Highscore und die Restdauer des Liedes kann der Spieler an Anzeigen im oberen Teil des Bildschirmes ablesen.

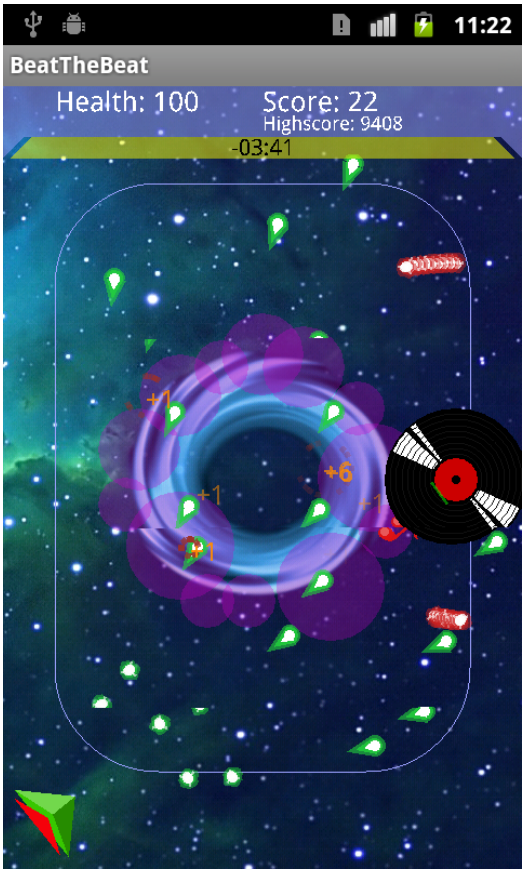


Abbildung 7.3

7.3.1 Steuerung und Spielmechanik

Zur Steuerung der Spielfigur wird das Accelerometer des Geräts abgefragt. Der Spieler hält das Gerät wie ein Tablett und kippt dieses in eine bestimmte Richtung. Die Spielfigur rutscht dann in selbige, vergleichbar mit einer Murmel auf einem kleinen Brett. So kann der Spieler Gegnern und Schüssen ausweichen.

Wird er dennoch getroffen, verliert er Lebenspunkte. Die Farbe des vorderen großen Dreiecks repräsentiert den Lebenspunktestand des Spielers. Es verfärbt sich bei abnehmendem Lebenspunktestand von grün nach rot.

Sinken die Lebenspunkte des Spielers auf Null, kommt es zu einer Explosion, die die Spielfigur über den Bildschirm schleudert. Diese bewegt sich dann von den Rändern abprallend über das Display. Der Spieler muss das Dreieck mit dem Finger durch Berühren einfangen und auf die am Rand des Displays entlanglaufende Linie führen, damit er wieder die Bewegung steuern und schießen kann. Die Explosion zerstört auch alle Gegner und die Gegnererzeugung wird ausgesetzt, bis der Spieler sich wieder zurück ins Spiel gebracht hat. Dies vereinfacht den Wiedereinstieg.

Um zu schießen hat man zwei Möglichkeiten. Entweder man tippt auf den Bildschirm oder man legt einen Finger auf diesen. Geschossen wird dann von der Position der

Spielfigur aus auf die Position der Berührung. Tippen erlaubt ein höheres Schusstempo, während schießen durch Berühren die Übersichtlichkeit für den Spieler erhöht.

7.3.2 Gegner

In der Mitte des Bildschirms befindet sich die Spirale (Abb. 7.4b). Hier erscheinen die Gegner. Die Gegnertypen sind:

Striker (Abb. 7.4c) sind schnell, klein und bewegen sich linear in eine zufällige Richtung. Sie reagieren auf das Bassfeature und ändern in einem, durch die Basslastigkeit des Liedes, bestimmten Intervall synchron alle ihre Richtung, so dass sie auf die aktuelle Position der Spielfigur zuhalten.

Spinner (Abb. 7.4e) sind schnell, klein und verfolgen die Spielfigur

Shooter (Abb. 7.4d) sind größer, langsamer und verfolgen ebenfalls die Spielfigur. Sie schießen einzelne Schüsse in ihre Flugrichtung.

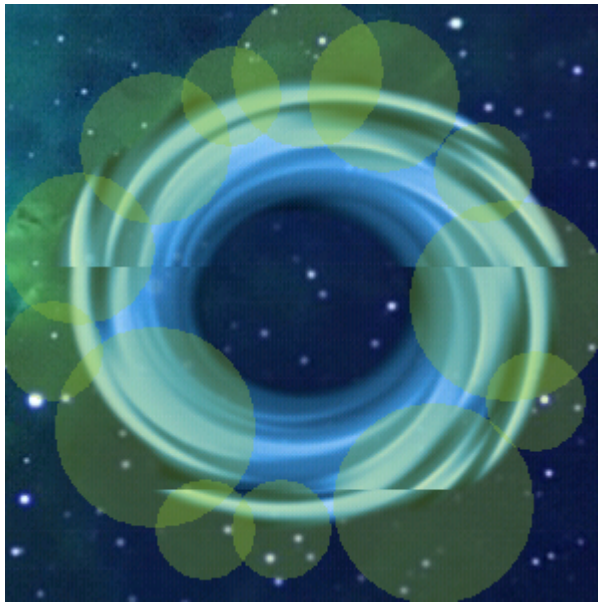
Artillerie (Abb. 7.4f) sind groß, langsam, robust und bewegen sich zufällig über das gesamte Spielfeld. Sie schießen Salven von Schüssen in ihre Flugrichtung.

Mutterschiff (Abb. 7.4g) ist sehr groß und hat sehr viele Lebenspunkte. Der Lebenspunktestand wird durch einen kleinen grünen Balken repräsentiert, damit der Spieler abschätzen kann wie lange er noch für die Zerstörung des Mutterschiff benötigt. Das Mutterschiff bewegt sich langsam über das Spielfeld und dient als zusätzlicher Erzeugungspunkt für Gegner. Ohne diesen war es dem Spieler zuvor möglich alle Gegner sofort beim Erscheinen in der Mitte abzuschießen, ohne in irgendeiner Form ausweichen zu müssen. Das Mutterschiff erzeugt Striker und Spinner oder schießt in eine zufällige Richtung. Hierbei werdend mehrere Schüsse auf einmal abgegeben, die in einer sich ausbreitenden Kette von dem Mutterschiff wegfliegen. Die Schuss- und die Erzeugungsfrequenz hängen vom Bassfeature ab: in basslastigen Passagen schießt es öfter und erzeugt mehr Gegner.

Das Erzeugen der Gegner erfolgt anhand der aus dem Musikstück extrahierten Merkmale. Diese bestimmen Typ und Frequenz der erzeugten Gegner. Der Rhythmus des Liedes bestimmt, wann ein Gegner erscheint. Es wird einer pro „Rhythmusevent“ erzeugt. Die „SensoryRoughness“, welche die Unruhe im Lied repräsentiert, bestimmt hierbei den Typ des erzeugten Gegners. Von hoch nach niedrig werden Spinner, Shooter und schließlich Artillerie erzeugt. Jeder zweite Gegner ist ein Striker. Zu Anfang des Spiels erscheint das Mutterschiff. Wird es vernichtet, wird es nach zwanzig Sekunden anstelle eines normalen Gegners erzeugt. Sollten sich einmal fünfzehn oder mehr Gegner gleichzeitig auf dem Bildschirm befinden, wird die Gegnererzeugung ausgesetzt bis es wieder weniger sind. Dies soll weniger erfahrenen oder weniger geschickten Spielern helfen. Gerade diese Spieler schaffen es unter Umständen nicht die erscheinenden Gegner schnell genug abzuschießen, wodurch sich ohne diese Beschränkung eine große Gegnermasse ansammeln könnte, was wiederum die Schwierigkeit weiter erhöhen würde. Wird ein Gegner vom Spieler abgeschossen, erhält der Spieler Punkte abhängig vom Gegnertypen. Die erhaltenen Punkte werden dem Spieler an der Position des Gegners durch eine kleine, langsam ausblendende Zahl angezeigt. Außerdem entsteht mit einer bestimmten

Chance ein Item, welches sich dann langsam in der ursprünglichen Flugrichtung des Gegners weiterbewegt.

(a) der Spieler

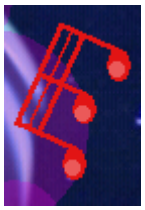


(b) die Spirale

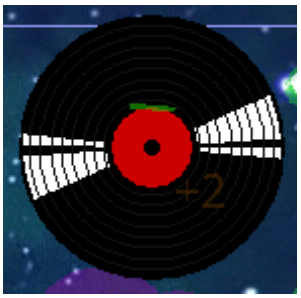
(c) Striker

(d) ein Shooter

(e) ein Spinner



(f) ein Artillerie



(g) das Mutterschiff

Abbildung 7.4: einige Spielelemente

7.3.3 Items

Der Spieler kann Items einsammeln indem er mit dem Finger auf dessen Position auf dem Bildschirm tippt. Für das Einsammeln eines Items erhält er sofort zehn Punkte. Jedes Item hat einen vom Typ abhängigen Effekt. Die meisten geben dem Spieler einen Spezialschuss. Sollte er beim Einsammeln schon einen haben, wird dieser durch den Neuen überschrieben. Hat der Spieler einen Spezialschuss, verbraucht dieser Energie

beim Schießen. Ist sie aufgebraucht, wird statt des Spezialschusses ein deutlich schwächerer Standardschuss abgegeben. Dies soll den Spieler motivieren, überlegt und gezielt zu schießen, um den maximalen Nutzen aus seinem Energievorrat zu ziehen. Die Energie regeneriert sich mit der Zeit langsam von alleine. Dem Spieler wird auch für das Einsammeln eines Items sofort etwas Energie wiederhergestellt. So kann er einen neuen Spezialschuss sofort benutzen. Das Auffüllen der Energie wird dem Spieler an der Position seiner Spielfigur durch eine kleine, langsam ausblendende Zahl angezeigt. Den Gesamtenergiestand kann der Spieler an der Farbe des kleinen hinteren Dreiecks an der Spielfigur erkennen. Dieses verfärbt sich bei abnehmender Energie von blau nach rot. Die verschiedenen Items sind:

Powerschuss gibt dem Spieler einen Schuss der besonders viel Schaden verursacht und Gegner durchschlägt. Diesen Spezialschuss hat der Spieler bei Spielbeginn, dies vereinfacht den Einstieg.

Multischuss erzeugt wenn der Spieler schießt drei statt einem Schuss.

Zielsuchender Schuss sorgt dafür, dass die Schüsse des Spielers sich automatisch ein Ziel suchen.

Rakete lässt den Spieler Raketen verschießen, die sich automatisch ein Ziel suchen, sehr viel Flächenschaden verursachen, aber auch sehr viel Energie verbrauchen.

Reparieren stellt dem Spieler beim Einsammeln die Hälfte seiner Lebenspunkte wieder her.

Energie füllt dem Spieler beim Einsammeln seine Energie komplett wieder auf. Dieses Item wird verhältnismäßig oft von den Gegnern zurückgelassen und soll ungefähr die Hälfte der Energie darstellen, die einem Spieler zur Verfügung steht. Wenn der Spieler seine Energie mit Bedacht ausgibt sollte er in der Lage sein mit ihr so viele Gegner zu töten, dass wieder ein neues Energie Item zurückgelassen wird und er nicht auf die langsame automatische Energieregeneration angewiesen ist.

KAPITEL 8

Tests am lebenden Menschen

Mirko Walter-Huber

Heutzutage ist es in der Computerspiel-Industrie üblich, neue Spiele vor ihrer Veröffentlichung stets von vielen Menschen testspielen zu lassen. Damit ist nicht nur das reine Testen zum Finden von Programmierfehlern gemeint, sondern es soll vielmehr herausgefunden werden, ob ein bestimmtes Spielkonzept auch in der Praxis so funktioniert, wie es geplant war. Das Wichtigste ist dabei: Es soll herausgefunden werden, ob ein Spiel **Spaß** macht.

Die Motivation dahinter ist offensichtlich. Wenn ein Spiel keinen Spaß bereitet, wird es sich nicht gut verkaufen und der Hersteller macht nicht genug Gewinn. Außerdem ist der Begriff Spaß sehr schwammig und jeder Mensch hat eine mehr oder weniger verschiedene Vorstellung davon, was für ihn Spaß bedeutet. Aus diesem Grund sind in der Industrie die Testabteilungen in der Regel relativ groß und man bemüht sich, möglichst viele verschiedene Menschen das eigene Spiel spielen zu lassen. Denn wenn es z. B. sowohl Gelegenheitsspielern, als auch den abgebrühtesten „Hardcore-Gamern“ Spaß macht, wenn es von jung und alt und auch von beiderlei Geschlecht gerne gespielt wird, dann hat es einen großen Markt und wird vermutlich sehr erfolgreich sein.

Spieletests sind also ein wichtiger Bestandteil jeglicher Computerspiel-Produktionen. Daher haben auch wir ein wenig Aufwand in diese Richtung betrieben. Natürlich nicht so ausführlich und nicht mit so vielen Testpersonen wie in der Industrie, aber wenigstens ein bisschen. Von diesen Bemühungen und den zugrunde liegenden Überlegungen soll in diesem Kapitel die Rede sein.

8.1 Feature-Studie

Bevor wir allerdings den Spielspaß testeten, wollten wir durch eine kleine Studie herausfinden, ob unser Konzept, den Spielablauf anhand von Musik-Features bestimmen zu lassen, so funktioniert, wie wir uns das vorstellten. Denn wenn ein Spieler den Zusammenhang zwischen Spielablauf und Dynamik der Musik nicht bemerken würde, dann wären sämtliche Mühen, aus den Liedern die Features zu extrahieren und aus diesen wiederum ein sinnvolles Spielgeschehen abzuleiten, umsonst gewesen.

Daher führten wir eine kleine Studie durch, um diesen Sachverhalt näher zu untersuchen.

8.1.1 Teilnehmer

Für unsere Studie konnten wir 20 Teilnehmer gewinnen. Sie waren gemischten Alters und beiderlei Geschlechts. Die einzigen Informationen, die wir von ihnen erfassten, waren ihre Selbsteinschätzung in Hinblick auf ihre Geübtheit im Umgang mit Computerspielen und ihre Musikalität.

Die Teilnehmer lassen sich also in vier Untergruppen kategorisieren:

- Computerspieler und musikalisch
- Computerspieler aber nicht musikalisch
- kein Computerspieler aber musikalisch
- kein Computerspieler und auch nicht musikalisch

Unsere Vermutung zu dieser Unterteilung war, dass sowohl geübte Computerspieler als auch musikalische Testpersonen den Zusammenhang zwischen Musik und Spielgeschehen besser erkennen würden. Diese Vermutung gründet in der Überlegung, das es musikalischen Menschen vermutlich leichter fallen wird, den Takt eines Liedes herauszuhören. Zusätzlich hat ein Computerspielerfahrener Spieler mehr Übung und Routine im Umgang mit einem Computerspiel und wird sich daher durch die rein motorischen Anforderungen, die unser Spiel stellt, weniger ablenken lassen. Aufgrund der gerade angestellten Überlegungen müsste natürlich die oben zuerst genannte Untergruppe am besten abschneiden, da sie beide der genannten Vorteile vereint.

8.1.2 Testprozedur

Als Testumgebung entwickelten wir eine eigenständige *App* für Android-Handys: Eine modifizierte Version unseres Spiels BeatTheBeat, welches die Testperson wiederholt das Minispiel TapByTap spielen lässt und dann jeweils sein Feedback abfragt.

Wir wollten gerne herausfinden, ob ein Spieler es bemerkt, *wann* wir ihn absichtlich täuschen. Dazu ließen wir ihn das Spiel mit *anderen* Features spielen, die eigentlich nicht zu dem Lied gehören würden, zu welchem er gerade spielt. Unsere Testpersonen durchliefen das Minispiel TapByTap dafür zu ein und demselben Lied jeweils zwei mal. Per Zufall wurde allerdings entweder bei dem ersten, oder bei dem zweiten Spieldurchlauf die echte Feature-Datei durch eine zufällig gewählte (falsche) Feature-Datei ausgetauscht. Trotz

der gleichen Musik reagierte das Spiel also in beiden Spieldurchläufen verschieden. Im Anschluss an einen solchen Doppellauf wurde die Testperson durch die *App* aufgefordert, sich zu entscheiden, ob das Spielgeschehen besser im ersten, oder besser in zweiten Durchgang zu der gespielten Musik passte.

Unsere Testprozedur bestand nun aus drei solchen Doppelläufen, jeweils mit anschließender Frage. Die drei Lieder wurden dabei zufällig aus einem Pool von 82 Liedern gewählt, welcher sich aus vielen verschiedenen Genres zusammensetzt. Jedes Lied wurde dabei außerdem auf seine erste Minute gekürzt, damit der gesamte Ablauf für unsere Testpersonen nicht zu lange dauerte. Auch bekam jeder Testspieler einen Probedurchlauf, der nicht in die Wertung eingegangen ist, um das Spiel erst einmal kennen zu lernen.

8.1.3 Ergebnisse

Teilnehmer			**Features**		
Computer-spieler	**musi-kalisch**	**Anzahl**	**KORREKT erkannt**	**FALSCH erkannt**	**Erken-nungsrate**
ja	ja	7	14	7	0,66
ja	nein	7	14	7	0,66
nein	ja	3	7	2	0,77
nein	nein	3	4	5	0,44
Gesamtanzahl:		20	39	21	0,65

Tabelle 8.1: Ergebnisse unserer Studie zur Wahrnehmbarkeit der Musikabhängigkeit des Spielgeschehens.

8.1.4 Statistische Analyse

Aufgrund der kleinen Teilnehmerzahl machte eine komplette statistische Analyse sämtlicher Untergruppen keinen Sinn. Wie in obiger Tabelle (8.1) zu sehen ist, bestanden zwei der vier Untergruppen nur aus drei Teilnehmern. Auf die Gesamtheit aller Teilnehmer jedoch wandten wir einen Binomialtest an, um zu überprüfen, wie wahrscheinlich es ist, dass unsere Testergebnisse durch puren Zufall entstanden sind. (Die Wahrscheinlichkeit, den Durchlauf mit dem korrekten Feature zu erkennen, oder eben nicht, wäre dann jeweils bei $p = 0{,}5$.)

Der Binomialtest ergab eine Wahrscheinlichkeit von 0,02734 (also unter 3%) dafür, dass unsere Rate von korrekt erkannten Features von 39 aus 60 Durchläufen rein zufällig entstanden ist.

8.1.5 Diskussion

Zuerst einmal muss gesagt werden, dass eine Teilnehmerzahl von 20 Personen eine sehr kleine Stichprobe ist. Um unsere Hypothese repräsentativ zu prüfen, hätten wir eine

deutlich größere Stichprobe benötigt, um auch handfeste Aussagen für sämtliche Untergruppen treffen zu können.

Solange man dies im Blick behält, lassen sich aber sehr wohl deutliche Tendenzen aus unseren Ergebnissen ablesen. Von sämtlichen Testdurchläufen wurde insgesamt bei etwa $^2/_3$ das korrekte, zur Musik passende Spielgeschehen erkannt. Dies könnte bedeuten, dass wir auf dem richtigen Weg sind und unser Spielkonzept eines musikbasierten Spieles so funktioniert, wie wir uns das vorgestellt haben. Allerdings wäre an dieser Stelle eigentlich eine Verifikation unserer Ergebnisse mit (wie oben schon erwähnt) deutlich mehr Teilnehmern nötig, um von gefestigten Aussagen anstelle von Vermutungen reden zu können. Hierzu fehlte uns allerdings die Zeit.

Unsere Erwartungen, dass die Teilnehmer, die sowohl im Umgang mit Computerspielen als auch musikalisch versiert waren, die allerbeste Erkennungsrate liefern würden, konnte in dieser Studie nur eingeschränkt beobachtet werden. So hatte rein statistisch die musikalische Untergruppe *ohne* Computerspielerfahrung die höchste Erkennungsrate von 0,77 ($^7/_9$). Allerdings könnte dies bei nur 3 Personen in dieser Untergruppe einfach nur Zufall gewesen sein. Die Tendenz allerdings, dass sowohl Computerspiel- als auch musikalische Kenntnisse bei unserer Aufgabe hilfreich sind, ist jedoch nicht von der Hand zu weisen. Computerspiel-unerfahrene Testpersonen waren während der Testprozedur beispielsweise so sehr mit der Steuerung beschäftigt, dass die Aufmerksamkeit auf die Musik und das Spielgeschehen nach eigenen Aussagen darunter gelitten hat.

8.2 Game-Balancing

Da für uns nun klar war, dass unser Spiel für den Benutzer erkennbar mit fast jeglicher Musik funktioniert, konnten wir das Ziel in Angriff nehmen, den Spielspaß zu maximieren. Das größte Schlagwort dabei ist das sogenannte *Game-Balancing*.

Unter *Balancing* versteht man die Bemühungen im Spiel ein Gleichgewicht z. B. zwischen der Gegnerstärke und den Spielermöglichkeiten zu erzeugen. Denn diese beiden Größen beeinflussen unter anderem den Schwierigkeitsgrad des Spiels. Ist ein Spiel entweder viel zu leicht, oder viel zu schwer, dann macht es dem Spieler keinen Spaß. Die Spielvariablen nun so zu optimieren, dass die meisten Spieler (mit ihren verschiedenen Fähigkeiten) ein Spiel als angenehm herausfordernd empfinden und Spaß dabei haben, ist das Ziel des *Game-Balancing*.

Was sind aber nun die Größen, die für ein Balancing geändert werden können? Dies hängt natürlich sehr stark von dem jeweiligen Spiel ab, welches man optimieren möchte. Der iterative Balancing-Prozess wird hier im folgenden am Beispiel des Minispiels TowerDefense beschrieben.

Unter anderem beeinflussen die folgenden Spielvariablen die Schwierigkeit dieses Minispiels:

1. Lebenspunkte der Gegner
2. Energie, die der Gegner gibt
3. Punkte, die der Gegner gibt

4. Punkte, die der Spieler abgezogen bekommt, wenn ein Gegner die Endkachel erreicht
5. Faktor, mit dem die Stärke der Wellen erhöht wird
6. Basisschaden, den Türme bei Gegnern verursachen
7. Skalierung des Schadens der Türme mit der Musik
8. Schadenserhöhung durch Upgrades der Türme
9. (Energie-) Kosten für die Upgrades der Türme

Es wurde versucht, diese Werte so zu gestalten, dass auch bei einem langsamen, ruhigen und wenig basslastigen Lied die ersten Gegner mit dem ersten Turm besiegt werden können. Dies konnte natürlich nicht für alle zur Auswahl stehenden Türme ermöglicht werden, aber zumindest sollte es für jedes unserer Testlieder wenigstens einen Turm geben, der die erste Welle an Gegnern komplett erledigen kann. Dies war ein manueller und sehr iterativer Prozess, bei dem die Anzahl der Gegner pro Welle, deren Lebensenergie und der von den Türmen verursachte Schaden angepasst wurden. Der PG ist bewusst, dass diese Werte nur durch eine vergleichsweise kleine Menge von Testliedern erzeugt wurden. Konkret bedeutet das also, dass es gewiss auch reichlich von uns ungetestete Lieder gibt, die unser Design-Ziel, dass das Spiel einen leichten Einstieg gewährt, *nicht* erfüllen.

Allerdings ist das Spiel so aufgebaut, dass es mit allen Liedern möglich ist, Punkte zu erzielen und alle Spieler die gleichen Chancen haben. Mit anderen Worten: Wenn ein Lied sehr ungeeignet für hohe Punktzahlen ist, dann ist es wenigstens für alle Spieler gleich ungeeignet und es kann immer noch ein fairer Wettkampf um die höchste Punktzahl entstehen. (Dies ist ja ein noch übergeordneteres Design-Ziel unseres gesamten Spieles.)

Wenn ein Spieler in TowerDefense nun viele Gegner besiegt, und damit viel Energie gesammelt hat, dann kann er auch viele neue Türme bauen oder alte verbessern. Das Spielgeschehen verlagert sich also von der Start- in die Hauptspielphase. In der Startphase ist es wie oben schon näher erläutert wichtig, die Gegner überhaupt zu besiegen und keinen an das Ende des Pfades gelangen zu lassen. In der Hauptspielphase, die dann beginnt, wenn ein Spieler bereits genug Türme gebaut hat, um alle in dem normalen Tempo kommenden Gegnerwellen vernichten zu können, könnte sich der Spieler ausruhen und das Ende des Liedes abwarten, ohne die Gefahr von Minuspunkten fürchten zu müssen. Allerdings würde er einen besseren Highscore erreichen, wenn er sich über die ‘New Wave’-Schaltfläche vorzeitig neue Gegner holen würde. Seine Türme sind ja offensichtlich stark genug, auch noch mehr zusätzliche Gegner abzuwehren. Erzeugt er sich dabei jedoch zu viele Gegner, erreichen diese das Ende des Pfades und der Spieler erhält dadurch Punktabzüge. Ziel war es also, einen intelligenten Gebrauch der ‘New Wave’-Schaltfläche nötig zu machen. Um dieses Ziel zu erreichen, variierten wir das Mächtigerwerden der Wellen und den Multiplikator für den Punktabzug, der zur Geltung kommt, falls ein Gegner doch einmal die Endkachel erreichen sollte. So ist z. B. der Punktabzug momentan das Fünffache der Punkte, die man sonst für das Besiegen dieses Gegners bekommen hätte. Vorher war er das 10-fache, was sich als eine zu harte Bestrafung herausstellte und hat so die Spieler zu vorsichtig werden lassen.

Um es dem Spieler nicht zu leicht, aber auch nicht zu schwer zu machen, die richtige Balance zwischen Türmen und Gegnern zu finden, wurde zusätzlich die von Gegnern zu bekommende Energie angepasst. Bekommt der Spieler mehr Energie, kann er mehr Türme bauen und dadurch auch mehr Gegner vernichten. Bekommt der Spieler zu viel Energie, ist er in der Lage so viele Gegner zu ordern und auch zu besiegen, dass man den Pfad vor lauter gedrängelten Gegnern kaum noch sieht. Ein differenziertes Spielgeschehen ist hier kaum noch möglich. Aber auch das Gegenteil ist nicht gut. Bei zu wenig Energie kann sich der Spieler schnell nicht mehr gegen die stärker werdenden Gegnerwellen wehren. Das Resultat wäre ein Spiel, das frustrierend wäre und daher keinen Spaß macht. Das Anpassen der Energiewerte war hier wieder ein sehr manueller und iterativer Prozess.

Als abschließendes Beispiel soll das Balancing zwischen dem Bauen von vielen Türmen und dem Ausbauen von bestehenden Türmen angeführt werden. Uns war es wichtig, dass ein Spieler je nach seinen Vorlieben oder der jeweiligen Spielsituation *eine* von diesen im allgemeinen relativ *gleichwertigen Methoden* wählen kann. Mit unseren zuerst gewählten Spielvariablen merkten wir aber sehr bald, dass ein Turm, dessen Reichweite erweitert wurde, deutlich mächtiger ist, als mehrere Türme. Da das Upgraden der Reichweite aber nur unwesentlich teurer war, als das Bauen von neuen Türmen, wurde hier ein Ungleichgewicht erzeugt. Die Preise für ein Upgrade mussten also nach oben korrigiert werden.

Abschließend muss erwähnt werden, dass diese Gameplay-Tests aus Zeitgründen hauptsächlich von den Teilnehmern der PG und deren Angehörigen gemacht wurden. Wünschenswerter wären hier sicherlich deutlich mehr Testpersonen, die zusätzlich das Spiel noch nicht so gut kennen. Auch wäre ein System hilfreich, welches den manuellen Anpassungsprozess der Spielvariablen an die jeweiligen Design-Ziele automatisieren würde, da dies ein sehr zeitintensiver Ablauf war. Im Zeitrahmen der PG wäre das allerdings ein zu ambitionierter Ansatz.

KAPITEL 9

Ausblick

Matthias Kuchem

Im Laufe der PG wurden viele Ideen entwickelt die über das implementierte hinaus gehen. Dieses Kapitel fasst die wichtigsten davon zusammen und gibt einen Einblick in das noch nicht entwickelte Potential von BeatTheBeat.

9.1 Verbesserung der Minispiele

Es ist immer möglich, die bestehenden Minispiele zu verbessern. Dabei muss jedoch darauf geachtet werden, dass die Spiele nicht zu kompliziert werden. Bewusstes Weglassen von Funktionen kann den Spielspaß in BeatTheBeat erhöhen, da der Fokus nicht auf dem einzelnen Minispiel, sondern der Integration der Musik liegt.

9.2 Neue Minispiele

Auch bei neuen Minispielen sollten Spiele bevorzugt werden, bei denen der Zusammenhang zur Musik erkennbar ist. Es ist bei fast jedem Spiel möglich die Gegner durch Musik steuern zu lassen, indem man die Musikfeatures auf die Menge der Steuermöglichkeiten anwendet. Allerdings erkennt der Spieler diesen Zusammenhang oft nicht direkt.

Weiterhin schränkt das Spielkonzept das Minispiel im wesentlichen auf die Benutzung genau eines Liedes ein. Damit entfallen die meisten Ideen der PG aus der ursprünglichen Ideensammlung.

9.3 Vermarktung

Das Spielkonzept wirft große Probleme bei der Vermarktung des Spiels auf. So ist es in der aktuellen Fassung nötig, dass alle Spieler alle Lied- und Feature-Daten aller auf dem Spielbrett befindlichen Lieder haben. Allerdings können diese aufgrund von Urheberrechten üblicherweise nicht verteilt werden. Die folgenden Ideen sind im Rahmen der PG entstanden:

9.3.1 Benutze nur „freie“ Musik

„Freie Musik“ kann von Verbrauchern ohne Entgelt bezogen und üblicherweise auch verteilt werden. Dabei ist natürlich im einzelnen die konkrete Lizenz zu beachten. Das Spiel kann ohne Lieder verkauft werden und verwendet kostenlos erhältliche Musikpakete. Die Musikfeatures können bei der Bereitstellung der Musikdateien generiert werden.

9.3.2 Musik-Bezahl-Modell

Der Spieler kann Spielbretter betreten, auch wenn er nicht alle erforderlichen Lieder hat. Diese Felder können dann nicht betreten werden und erfordern das Kaufen der Musik in einem eigens angebundenen Shop. Damit die Kosten für den Spieler nicht zu hoch werden, können die Musikfeatures von vorhandenen Liedern kostenfrei bezogen werden.

9.3.3 Spieler zahlt für Features

Der Spieler kann wie beim vorigen Vorschlag alle Spielbretter betreten. Wenn er ein Lied nicht hat, muss er die Lieddatei selbst besorgen (Links zu eingängigen Online-Musikshops mit Provision werden bereit gestellt). Die Featuredaten erhält er dann gegen einen Unkostenbeitrag von dem Spielebetreiber. Es müssen also bei der Erstellung eines Spielbretts alle Featuredaten vorliegen.

Literaturverzeichnis

[BBvB+] Beck, Kent, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland und Dave Thomas: *Manifest für Agile Softwareentwicklung*. http://agilemanifesto.org/iso/de/.

[BP10] Becker, Arno und Marcus Pant: *Android 2: Grundlagen und Programmierung*. Dpunkt Verlag, 2010.

[DmGH+11] Delbrügger, Tim, Ümit Güler, Jannic Hartwecker, Annika Jordan, Matthias Kuchem, Jan Lahni, Dimitri Scheftelowitsch, Nils Vortmeier und Mirko Walter-Huber: *Zwischenbericht PG554 - Mobile Musikwelten*. Technischer Bericht, Lehrstuhl Algorithm Engineering, Fakultät für Informatik, Technische Universität Dortmund, 2011.

[Glo11] Gloger, Boris: *Scrum: Produkte zuverlässig und schnell entwickeln*. Hanser, München, 3. Auflage, 2011.

[Ker10] Kerger, Felix: *Ogre 3D 1.7 Beginner's Guide*. 2010.

[MG96] Moore, Brian C. J. und Brian R. Glasberg: *A Revision of Zwicker's Loudness Model*. Acustica – acta acustica, 82:335–345, 1996.

[MM05] Mierswa, Ingo und Katharina Morik: *Automatic feature extraction for classifying audio data*. Machine Learning Journal, 58:127–149, 2005.

[PV11] Preuss, Mike und Igor Vatolkin: *Mobile Musikwelten: Spielerische Exploration von digitalen Musiksammlungen*. In: Windmüller, Stephan (Herausgeber): *INFO zu den Projektgruppen der Fakultät für Informatik mit Beginn im SoSe 2011*, Seiten 13–16. 2011.

[Sch04] Schwaber, Ken: *Agile Project Management with Scrum.* Microsoft Press, Redmond, WA, 2004.

[Set98] Sethares, William A.: *Tuning, Timbre, Spectrum, Scale.* Springer, 1998.

[TVE08] Theimer, Wolfgang, Igor Vatolkin und Antti Eronen: *Definitions of Audio Features for Music Content Description.* Technischer Bericht TR08-2-001, Lehrstuhl Algorithm Engineering, Fakultät für Informatik, Technische Universität Dortmund, 2008.

[VTB10] Vatolkin, Igor, Wolfgang Theimer und Martin Botteck: *AMUSE (Advanced MUSic Explorer) - A Multitool Framework for Music Data Analysis.* In: *Proceedings of the 11th International Society for Music Information Retrieval Conference (ISMIR)*, Seiten 33–38, 2010.