HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Game-Programmierung auf der Android Plattform

Bachelorarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Frühjahrssemester 2012

Autoren:	Marco Pfiffner, Mathias Fasser
Betreuer:	Prof. Hans Rudin
Experte:	Daniel Hiltebrand
Gegenleser:	Dr. Daniel Keller

HSR Hochschule für Technik Rapperswil
Oberseestrasse 10
Postfach 1475
CH-8640 Rapperswil
www.hsr.ch

Erklärung der Eigenständigkeit

Wir erklären hiermit,

- dass wir die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt haben, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt ist oder mit dem Betreuer schriftlich vereinbart wurde,
- dass wir sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben haben.
- dass ich keine durch Copyright geschützten Materialien (z.B. Bilder) in dieser Arbeit in unerlaubter Weise genutzt habe.

Ort, Datum: Rapperswil, 15.06.2011

Name, Unterschrift:

Marco Pfiffner

Mathias Fasser

Danksagung

Wir danken folgenden Personen für Ihre Unterstützung während der Bachelorarbeit:

- **Prof. Hans Rudin** für die Betreuung unserer Bachelorarbeit.
- **Sandi Elezovic** für das Erstellen der Grafiken.
- **Romina Brandstetter** für das Korrekturlesen der Arbeit.

Inhaltsverzeichnis

1 Abstract

Das Ziel dieser Bachelorarbeit[1] ist es, sich mit der Game-Programmierung und speziell mit der Programmierung für Games auf der Android Plattform auseinander zusetzen. Nachdem in einer Studie Informationen über die digitale Spielentwicklung gesammelt wurden, hat sich das Projektteam einen Überblick über die verfügbaren Game-Engines auf Android verschafft. Durch die Definition eines Game-Konzepts und den Informationen aus der Studie konnte eine passende Game-Engine gefunden werden.
Mittels der AndEngine wurde anschliessend ein Game - Wilson's Adventures - entwickelt. Da das Game auf physikalischen Grundprinzipien basiert, wurde zusätzlich zur AndEngine die Physik-Engine Box2D verwendet.

Es wurde bewusst ein Game-Konzept mit passender Storyline gewählt, welches auf Erweiterbarkeit und Vielseitigkeit aufbaut. Die Erweiterbarkeit ermöglichte dem Projektteam das schrittweise Auf- und Ausbauen des ursprünglichen Game-Konzepts. Die Vielseitigkeit sorgte dafür, dass viele Bereiche der Game-Programmierung behandelt werden konnten.

Das Ergebnis dieser Arbeit ist ein Game, welches auf Android Version 2.2 basiert. Konfigurationen und Level-Definitionen wurden in XML-Files ausgelagert und werden während dem Spiel eingelesen. Durch ein Kollisionsmodell, das beliebig komplexe Körper darstellen kann, und der Box2D Collision Detection wurde die Game-Logik implementiert. Zusätzlich wurde mittels SQLite ein Scoring-Modell realisiert.

Diese und weitere Bereiche des entwickelten Games sorgen dafür, dass viele Komponenten wiederverwendet werden können und somit eine gute Grundlage bieten, um neue Ideen auf dem Android Markt zu verwirklichen.

[1] Die genaue Aufgabenstellung befindet sich im Anhang A.

2 Management Summary

2.1 Ausgangslage

Der Einzug des Smartphones als treuer Begleiter vieler Menschen eröffnete eine neue Plattform für die Softwareentwicklung. Der hart umkämpfte Markt und die rasant steigende Anzahl an Apps war für das Projektteam Anlass genug, um sich bereits in der Studienarbeit genauer in die Android-Technologie einzuarbeiten. Dabei ging es um eine Business-Application für die Firma Centrado.

Die Tatsache, dass heutzutage jeder Smartphone Besitzer eine Spielkonsole mit sich trägt, eröffnete auch für die digitale Spielindustrie neue Wege. Vor allem für kleinere Teams von Game-Entwicklern bietet sich heute eine erstklassige Möglichkeit um auf dem Markt mitzuwirken. Die Faszination gegenüber der Spielentwicklung sowie die positiven Eindrücke aus der Semesterarbeit bewegten das Projektteam dazu, ein Game auf der Android Plattform zu entwickeln.

2.2 Vorgehen, Technologie

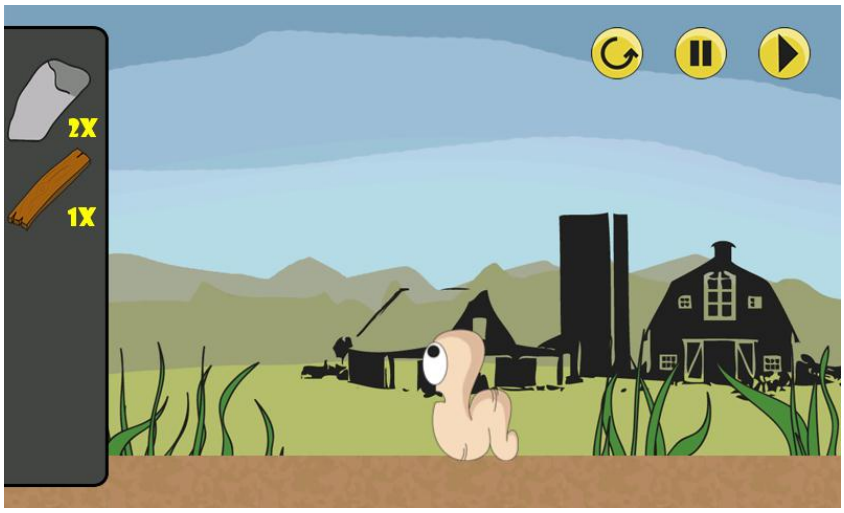
Zu Beginn der Arbeit wurde ein Game-Konzept entwickelt, welches die Rahmenbedingungen des Games abstecken sollte. Dabei wurde der grobe Ablauf des Games mit den Interaktionen des Spielers und die Storyline definiert.

In einer Studie erwarb sich das Projektteam anschliessend Wissen über die Spiele-Entwicklung. Dabei wurde die historische Entwicklung von digitalen Spielen betrachtet. Die Veränderung der Spielprinzipien über die Jahre hinweg war dabei ein wichtiger Punkt. Vor allem neue Spielprinzipien, welche stark von den Smartphones geprägt oder sogar erst dadurch entstanden sind. Das in dieser Arbeit entwickelte Spiel beruht auf den Prinzipien der Casual Games. Die gesammelten Informationen über verfügbare Android-Game-Engines wurden dokumentiert und ausgewertet. Mittels des Game-Konzepts konnten die Anforderungen an die Game-Engine priorisiert und ausgewertet werden. Die Wahl fiel auf die neue, jedoch bereits vielfach benutzte AndEngine.

Nachdem sich das Projektteam in die AndEngine eingearbeitet hatte, wurden parallel zwei Prototypen entwickelt, welche die grössten technischen Risiken beseitigen sollten. Die Aufteilung der Prototypen in Dynamic-Loading und Game-Logik setzte den Grundstein für eine saubere Trennung dieser Bereiche.
Das Dynamic-Loading stellt das Grundgerüst des Games dar, das auch gut wiederverwendet werden kann. Das Dynamic-Loading umfasst das Laden von Levels und Level-Kategorien aus XML-Files.
Die Game-Logik ist nur teilweise wiederverwendbar, da sie die konkreten Anforderungen aus dem Game-Konzept umsetzt und schlussendlich Wilson's Adventures ausmacht. Die grossen Herausforderungen bezüglich der Game-Logik waren die Collision Detection sowie das Kollisionsmodell der einzelnen Gegenstände. Nebst der Einbindung von Grafiken und Audio wurde ein Scoring-Modell implementiert, welches die Level-Freischalte-Logik kapselt und Highscores in einer SQLite Datenbank speichert.

2.3 Ergebnisse

Das Ergebnis dieser Bachelorarbeit ist ein Android-Game, welches zur Kategorie der Casual Games gehört. Der Spieler muss dabei in übersichtlichen Levels die Hauptfigur des Games, Wilson, beschützen. Per Drag'n'Drop können Gegenstände auf dem Level platziert werden, damit eine Angriffswelle von oben den Wurm nicht verletzt. Sofern Wilson von den Angriffselementen nicht getroffen wurde, gilt das Level als bestanden.



Da das Ziel in dieser Bachelorarbeit nicht auf die Entwicklung eines marktreifen Game gelegt wurde, ist auch nur eine geringe Anzahl Levels vorhanden. Ebenfalls müssten besser passende Sounds erstellt werden, um das Game der breiten Masse zu präsentieren.
Dem Projektteam war es wichtig, die Spielentwicklung kennen zu lernen und sich dabei in möglichst viele herausfordernde Gebiete einzuarbeiten. Ebenfalls priorisierte das Projektteam eine Software-Architektur, die in weiteren Games hauptsächlich nur um die Game-Logik ergänzt werden muss.

2.4 Ausblick

In dieser, für das Projektteam, ersten Begegnung mit der Game-Programmierung wurden viele interessante Aspekte durchleuchtet und gelernt. Durch die Möglichkeit, dass auch kleine Teams auf dem Smartphone-Markt als Gameentwickler agieren können, bleibt dieses Thema auch in Zukunft interessant.
Mit den gewonnenen Erfahrungen und den wiederverwendbaren Modulen kann in Zukunft eine Spielidee effizient umgesetzt werden.

Ob das Projektteam mit Wilson's Adventures erste Erfahrungen auf dem Android-Markt sammelt, ist noch nicht sicher. Da es aber nur ein kleiner Schritt in Richtung der Marktreife ist, kann es durchaus sein, dass das Spiel in absehbarer Zeit auf dem Android-Markt zum Download bereit steht.

3 Game-Studie

3.1 Einführung

Durch die Game-Studie soll sich das Projektteam das benötigte Wissen aus dem Bereich Game-Programmierung und speziell der Game-Programmierung auf Android aneignen. Es geht nicht um konkrete Problemstellungen, sondern vielmehr darum, die Game-Programmierung grundsätzlich kennen zu lernen. Dabei wird die Geschichte der Game-Entwicklung mit dem Schwerpunkt auf mobile Games behandelt. Ebenfalls gibt es eine Einführung in die verschiedenen Game-Genres. Anschliessend soll durch diese Studie das Angebot an Game-Engines für Android analysiert werden.
Durch die Studie soll es dem Projektteam anschliessend möglich sein, das Projekt besser einschätzen zu können und eine passende Game-Engine zu finden.

3.2 Digitales Zeitalter

In der heutigen Zeit ist die digitale Datenverarbeitung nicht mehr wegzudenken. Wir begegnen der Computer-Technik überall. Sei es direkt bei der Arbeit mit dem Computer, beim Geld abheben, beim Surfen im Internet oder indirekt beim Einkaufen an der Kasse, bei der Bedienung eines Lifts oder bei der Fahrt mit modernen Autos. Was zuerst für die digitale Informationsverarbeitung gedacht war, hat sich so rasant weiter entwickelt, dass daraus viele unterschiedliche Einsatzgebiete und Funktionen entstanden sind.

3.3 Computer Games[2]

Ein aus der digitalen Informationsverarbeitung entstandener Zweig war schon sehr früh die Spielentwicklung. Bereits 1958 wurde vom Amerikaner William Higinhotham „Tennis for Two“ entwickelt, welches als eines der ersten Computerspiele gilt. Die Entwicklung von „Tennis for Two“ fand damals auf einem Grossrechner an einer Universität statt.

In den 1970er Jahren war es bereits möglich, gewisse Spiele an Spielautomaten zu spielen. Eines der bekanntesten Spiele davon war „Pong“, welches als Nachfolger von „Tennis for Two“ gilt. Durch diesen Schritt sowie durch die Entwicklung von Spielkonsolen für Heimanwender, hatten viele Menschen plötzlich die Möglichkeit, digitale Spiele zu spielen.

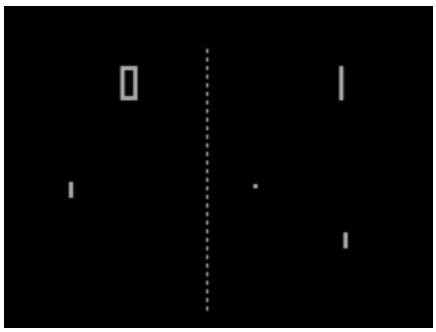


Abbildung 1: Pong (Quelle: de.wikipedia.org)

In den 1980er Jahren wurden zwei neue Technologien zur Plattform für digitale Spiele. Zum einen kamen Videogames auf, welche sich auf speziellen Spielkonsolen spielen liessen. Andererseits brachte die Einführung des PCs eine neue Plattform, welche für die Spielindustrie sehr interessant war.

[2] [WikiDEComputerSpiel]

3.4 Mobile Games

Die Revolution der Smartphones brachte viele Veränderungen mit sich. Man ist jederzeit online und somit erreichbar, der Zugriff aufs Internet ist praktisch von überall aus möglich, Terminkalender, Mailverkehr sowie Videos und Bilder hat man immer dabei.

Nebst den Business-, Finanz-, Reise- und Shopping-Apps gibt es viele weitere App-Kategorien, welche den Alltag erleichtern.

Die grosse Beliebtheit der Smartphones liess jedoch auch eine neue Plattform für Spiele entstehen, welche heute im grossen Stil benutzt wird. Mit Spielen auf dem Smartphone eröffneten sich für die Spiele-Industrie ganz neue Möglichkeiten.

3.4.1 Spieler

Was früher eine spezielle Konsole oder einen besonders leistungsstarken Computer benötigte und dadurch mit hohen Kosten verbunden war, tragen heute viele Personen selbstverständlich mit sich herum. Man braucht somit keine zusätzliche Hardware, was früher jeweils die erste Hürde darstellte, wenn man ein Computerspiel spielen wollte.
Dies bietet einen Vorteil für spielbegeisterte Personen, jedoch auch für die Spiele-Entwickler. Der Vorteil für die Entwickler ist offensichtlich: Auch Smartphone Benutzer, welche sich nie spezielle Spiel-Hardware angeschafft hätten, sind heute potenzielle Kunden für Mobile-Spiele.

Herkömmliche Spiele wurden entweder an Konsolen oder auf Computern gespielt. Der Spieler musste also zu Hause am jeweiligen Gerät sein, damit er spielen konnte. Mit dem Smartphone hat man hingegen die Möglichkeit, ein Spiel immer und überall zu spielen. Sei es im öffentlichen Verkehr, beim Anstehen in einer Menschenschlange oder zu später Stunde am Küchentisch.

Die Tatsache, dass ein Spiel auf dem Smartphone jederzeit gespielt werden kann, hatte auch einen gewissen Einfluss auf die bisherigen Spiel-Genres, welche sich auch auf den neuen Smartphone Markt angepasst haben.
Die Spiele-Entwickler konnten nicht mehr davon ausgehen, dass sich der Spieler eine Stunde für sein Hobby einplant. Vielmehr ist es heute normal, kurz in ein Spielerlebnis einzutauchen, um es jedoch genauso schnell wieder zu beenden, da man sich beispielsweise endlich an die erste Position an der Kasse des Baumarkts vorgekämpft hat.

Es gibt heute viele Spiele, welche genau für diese Anwender konzipiert sind. Dies setzt voraus, dass ein Spiel in wenigen Minuten gespielt werden kann, der Spielspass jedoch nicht verloren geht.

3.4.2 Entwickler[3]

Die Tatsache, dass die Spieler ein kurzes, jedoch fesselndes Spielerlebnis haben wollen, bringt einige Veränderungen für den Entwickler mit sich.

Wenn ein Computer- oder Konsolenspiel entwickelt wird, kann dies schnell bis zu 10 Mio. Dollar kosten. Bei sehr komplexen Spielen, welche zusätzlich für mehrere Plattformen entwickelt werden, kann die Entwicklung durchaus bis zu 100 Mio. Dollar kosten. Bei diesen Spielen kostet allein das Software Development Kit (SDK) tausende Dollars. Wenn das Spiel auf aufwendigen 3D-Animationen basiert, kommen zusätzlich noch CAD und Visualisierungs-Programme zum Einsatz, die ebenfalls sehr teuer sein können.

Bei der Entwicklung eines Games für den Smartphone-Markt ist es nicht ungewöhnlich, wenn ein Spiel nur von einem Entwickler oder in kleinen Teams entwickelt wird. Da das SDK gratis ist und eine kostenlose Engine verwendet werden kann, fallen neben den Registrierungsgebühren für den Android Market keine Kosten an.
Natürlich können, je nach Aufbau des Spiels, höhere Kosten entstehen. Jedoch ist es auch mit beschränkten Mitteln durchaus möglich ein Spiel zu entwickeln, das auf dem Markt jedem Smartphone Benutzer angeboten werden kann.

3.4.3 Genres[4]

Es gibt keine Standards zum Kategorisieren unterschiedlicher Games, dennoch lassen sich verschiedene Genres mit speziellen Eigenschaften definieren. Die folgende Liste ist keineswegs abschliessend, sie soll jedoch veranschaulichen, welche Genres in Bezug auf die mobile Game-Entwicklung relevant sein könnten.

3.4.3.1 Action

Das Action Genre umfasst eine breite Palette an unterschiedlichen Games, da sich dieses Genre wiederum in viele Subgenres unterteilen lässt. So gehören zum Beispiel Kampf-, Labyrinth- und Shooter-Games zu den Action Games. Vielfach handelt es sich bei Action Games um grafisch sehr anspruchsvolle Games. Da solche Games selten von Einzelpersonen entwickelt werden und bislang noch nicht viele grosse Gamehersteller auf dem Mobile-Markt tätig sind, ist die Auswahl an solchen Games ziemlich beschränkt.
Es hat sich auch gezeigt, dass die Leute noch nicht bereit sind, die höheren Entwicklungs-Kosten für ein solches Game zu bezahlen. Es wird sich zeigen, inwiefern sich der Android-Markt in diese Richtung entwickeln wird.

[3] [LeAnGaPro]

[4] [BeAnGa]

In Abbildung 2: Grand Theft Auto 3 (Quelle: play.google.com) ist ein sogenannter Third-Person Shooter zu sehen. Dabei bezieht sich Third-Person auf die Sicht des Spielers. In einem Ego-Shooter bzw. First-Person Shooter spielt man in der Sicht eines Charakters, hingegen in einem Third-Person Shooter schaut man einem Charakter „über die Schulter".

Abbildung 2: Grand Theft Auto 3 (Quelle: play.google.com)

3.4.3.2 Strategie

Das Strategie Genre besteht oftmals aus zeitintensiven Spielen. Es geht darum, Hindernisse und Gegner mit einer ausgeklügelten Strategie zu überwinden oder zu besiegen. Sei es in einem traditionellen Brettspiel oder auf einem virtuellen Schauplatz des 2. Weltkriegs. In Abbildung 3: Die Siedler HD (Quelle: www.gameloft.de) ist ein typisches Strategie-Spiel zu sehen.

Abbildung 3: Die Siedler HD (Quelle: www.gameloft.de)

Da diese Games meistens sehr zeitaufwendig sind und der durchschnittliche Android-Gamer kurze Game-Sessions bevorzugt, hält sich der Erfolg solcher Games in Grenzen.

3.4.3.3 Abenteuer

Abenteuer-Spiele bestehen aus einer interessanten und detaillierten Storyline. Oftmals wird man in die Rolle eines Detektivs versetzt und versucht verlorene Gegenstände zu finden oder unterschiedlichste Verbrechen aufzuklären. Vielfach herrscht eine etwas gruselige Atmosphäre. Abbildung 4: The Secret of Grisly Manor (Quelle: market.android.com) zeigt ein Spiel, indem man seinen verschwundenen Grossvater finden muss.

Abbildung 4: The Secret of Grisly Manor (Quelle: market.android.com)

3.4.3.4 Simulation

In einem Simulations-Game können reale Situationen nachgespielt werden. So gehören beispielsweise Sportspiele, Flug- und Rennsimulatoren sowie Lebenssimulationen in dieses Genre.
„Need for Speed“ ist eines der renommiertesten Spiele aus dem Simulations-Genre (Abbildung 5: Need for Speed Shift (Quelle: play.google.com)).

Abbildung 5: Need for Speed Shift (Quelle: play.google.com)

3.4.3.5 Puzzle / Rätsel

Eines der populärsten Genres in der mobilen Game Welt stellen die Puzzle- und Rätselspiele dar. Viele Kassenschlager des Android Markets stammen aus dieser Gruppe („Cut the Rope“, „Where’s my Water?“, „Greedy Spiders“ usw.). In der Abbildung 6: Cut the Rope (Quelle: play.google.com) ist ein Screenshot von „Cut the Rope“ zu sehen, eines der erfolgreichsten Android-Games überhaupt.

Abbildung 6: Cut the Rope (Quelle: play.google.com)

3.4.3.6 Casual Games

Die genaue Definition eines Casual Games ist schwierig. Dennoch stellt dieses Genre die wohl grösste Gruppe von Games auf dem Android Market dar. Casual Games überschneiden sich mit diversen anderen Genres. Durch ihre leichte Zugänglichkeit, die intuitive Bedienung und schnelle Erfolgserlebnisse sind sie optimal auf die Gelegenheitsspieler am Smartphone zugeschnitten. Durch ihre Einfachheit adressieren sie ein viel grösseres Zielpublikum als andere Games.
Eine Game-Session dauert nur ein paar Minuten. Es soll ein optimaler Zeitvertreib für Zwischendurch sein, sei es an der Bushaltestelle, auf der Toilette oder im Warteraum des Zahnarztes.
Prominente Casual Games sind „Doodle Jump“, „Cut the Rope“ oder auch „Angry Birds“.

Abbildung 7: Angry Birds (Quelle: play.google.com)

3.4.3.7 Folgerung für diese Arbeit

Die momentan prominentesten Titel der Mobile-Games stammen grösstenteils aus dem Bereich der Casual Games. Wenn man die Grundprinzipien der Casual-Games beachtet und in das Game-Konzept einfliessen lässt, kann ein viel breiteres Zielpublikum angesprochen werden.

Beim Erarbeiten des Game-Konzepts gilt es diese Grundprinzipien zu beachten und mit den vorhandenen Ideen zu vereinen. Die intuitive Benutzung sowie die kurze Rundendauer sind als nichtfunktionale Anforderungen in die Anforderungsanalyse aufgenommen worden.

3.5 Game-Engines

Was unterscheidet den Zweig der Spielentwicklung von der Entwicklung anderer Software wie zum Beispiel einer Business Applikation oder einem Browser?

Ein Spiel basiert auf kreativen Ideen, die das Spielkonzept sowie die Storyline und die Charaktere definieren. Zusätzlich greift ein Spiel meistens intensiv auf aufwändige 2D oder 3D Animationen, unterschiedliche Sound-Effekte und spezielle Input-Varianten zu. Ebenfalls befindet sich das ganze Spiel in einem Loop, welcher das Spiel erst spielbar macht. Dabei werden die eben genannten, spielspezifischen Abläufe vereint und für den Spieler zum Erlebnis.

All diese, mit einem Spiel verbundenen, Aufgaben werden in einer Game-Engine abstrahiert und somit den Spiele-Entwicklern zur Verfügung gestellt.

3.5.1 Geschichte[5]

Für lange Zeit entwickelten alle Spiele-Entwickler ihre eigenen Game-Engines, um diese dann ausschliesslich für ihre Spiele verwenden zu können. Heutzutage lohnt sich der Aufwand meistens nicht, eine eigene Game-Engine zu entwickeln. Firmen, welche ihr Kerngeschäft auf die Entwicklung der Engines verlegt haben, bieten heute bezahlbare Game-Engines an. Somit wird es für die Spiel-Entwickler möglich, sich hauptsächlich auf die Entwicklung des Spiels zu konzentrieren und dadurch schneller neue Spiele präsentieren zu können.

Folgende weitere Vorteile bietet die Verwendung einer bereits existierenden Game-Engine:

- Eine bestehende Game-Engine wurde bereits von mehreren Personen und Firmen verwendet. Dies lässt darauf schliessen, dass bereits Optimierungen gemacht wurden, von welchen man als Benutzer profitieren kann.
- Bei Problemen mit der Verwendung der Engine kann vielfach auf eine Community zurückgegriffen werden, welche bereits Foren-Einträge oder Wiki Hilfestellungen darüber gesammelt hat und zur Verfügung stellt.
- Bei der Verwendung von Open-Source Engines hat man die Möglichkeit, eigene Anpassungen oder Erweiterungen vorzunehmen, welche dann wiederum von anderen Entwicklern verwendet werden können.

[5] [GameCareerGuide], [LeAnGaPro]

3.6 Android Game-Engines

3.6.1 Einführung

Das Angebot von Game-Engines, welche die Game-Entwicklung auf Android Smartphones ermöglichen, ist bereits gross.

Die Game-Engines für mobile Geräte lassen sich, wie auch herkömmliche Engines, in mehrere Sparten unterteilen. Eine wichtige Unterscheidung ist die Aufteilung in 2D- und 3D-Engines. Zusätzlich ist die Verwendung webbasierter Engines für die Game-Programmierung möglich.

Folgende Engines für die Spiel-Programmierung auf Android wurden während dieser Studie berücksichtigt. Die Listen sind nicht abschliessend.

3.6.2 2D-Engines

- AndEngine
- Corona Game Edition
- Cuttlefish Engine
- Google App Inventor
- Libgdx Framework
- Mages Engine
- Rokon 2D Game Engine

3.6.3 3D-Engines

- Airplay SDK 4.2
- alien3d
- catcake
- jMonkeyEngine
- jPCT-AE
- loon-simple
- Unity3D 3.0
- Unreal Engine

3.6.4 Webbasierte-Engines

- Aves Engine
- Flash

3.7 Auswahl der passenden Engine

3.7.1 Auswahlkriterien

Um eine passende Engine zu finden, müssen zuerst einige Fragen bezüglich des Spiels geklärt werden:

- Wird das Spiel in 2D oder 3D dargestellt?
- Benötigt man eine Physik-Engine?
- Welche Technologien kommen in Frage?

Jedoch auch andere Kriterien sind wichtig für die Auswahl der passenden Engine:

- Wie bekannt ist die Engine
  - Referenzprojekte (bekannte Spiele)?
  - Besteht eine Community?
  - Wie sehen die Zukunftspläne der Engine aus?
- Wie gross ist das Budget, welches für den Kauf einer Engine zur Verfügung steht?
- Sollte das Spiel auf mehreren mobilen Plattformen ausgeführt werden können?

3.7.2 Gewichtung der Kriterien

Nach Fertigstellung der ersten Skizzen sowie der Definition des Gameplays und der Storyline konnten folgende Entscheide gefällt werden:

- Das Spiel soll in einer 2D Umgebung dargestellt werden.
- Ebenfalls ist eine Physik-Engine nötig, da man mit realen Gegenständen etwas konstruieren muss.
- Das Projektteam interessiert sich für die OpenGL Technologie und möchte, wenn möglich, eine Engine verwenden, welche auf OpenGL basiert.
- Der Bekanntheitsgrad und die Grösse der Community rund um die Engine sind für den Entscheid sehr wichtig, da dies das erste Projekt in diesem Umfeld für das Projektteam ist.
- Wenn möglich soll eine kostenlose Engine verwendet werden.
- Die Portierbarkeit auf andere Mobile-Plattformen hat in dieser Arbeit keine Priorität.

3.7.3 Auswertung

Aufgrund der Wahl einer 2D-Engine wurden einige Engines genauer betrachtet. Diese sind im folgenden Kapitel aufgelistet und beschrieben. Da das Projektteam die OpenGL basierten Engines bevorzugt, wurde keine der beiden webbasierten Engines näher betrachtet. Die folgenden Unterkapitel zu den einzelnen 2D-Engines wurden aus Informationen der Hersteller und von Meinungen der Benutzer der Engines erstellt (siehe Quellenangaben).
Das Projektteam hat sich aufgrund dieser Informationen auf eine, für die Bachelorarbeit passende, Engine entschieden. Unter dem Titel Ausschlaggebende Fakten sind jeweils die für das Projekt relevanten Punkte aufgelistet.

3.7.3.1 AndEngine

Beschreibung:
AndEngine ist eine schnelle und vielfach benutzte 2D Engine, welche für Android optimiert ist. Die Engine unterstützt Split Screen, Network Multiplayer sowie Multi Touch und eine Physik-Engine.

Freeware:
Ja

Ausschlaggebende Fakten:
Die Engine ist kostenlos und wurde bereits oft für bekannte Spiele verwendet. Teilweise mit über 5 Mio. Downloads. Im Internet stösst man nur auf gute Erfahrungsberichte und eine grosse Community. Ebenfalls kann die Physik-Engine Box2D verwendet werden.

Links:
http://www.andengine.org

3.7.3.2 Corona Game Edition

Beschreibung:
Die Corona Engine benutzt eine eigene Sprache, welche mit ActionScript vergleichbar ist. Die Engine ist sehr schnell und bietet eine eingebundene Physik-Engine.

Der grosse Vorteil von Corona ist die Portierbarkeit auf mehrere Plattformen (Android, iOS, Kindle Fire, Nook Color) und die Unterstützung der unterschiedlichen Screen Grössen und Versionen. Ebenfalls bietet Corona eine automatische Portierung von Smartphone Apps auf Tablets.

Freeware:
Nein

Ausschlaggebende Fakten:
Die Engine ist bereits sehr etabliert und bietet auch eine grosse Community. Jedoch ist die Engine kostenpflichtig und konzentriert sich nicht speziell auf die Entwicklung von Spielen für die Android Plattform.

Links:
http://anscamobile.com/corona/games/index.html

3.7.3.3 Cuttlefish Engine

Beschreibung:
Cuttlefish ist eine neue Lösung um ein Spiel für mehrere Plattformen zu erstellen. Dabei muss nur die Game-Logik implementiert werden, alles andere wird von der Engine übernommen. Anschliessend kann das Spiel auf dem Android Market oder auf dem App-Store angeboten werden, da es in einer plattformunabhängigen Umgebung erstellt wurde.

Freeware:
Die Verwendung der Engine ist kostenlos. Die Applikation (Designer), welche für die Erstellung des Spiels benötigt wird, ist jedoch kostenpflichtig. Ebenfalls muss ein Abo gelöst werden, wenn die App gebildet werden soll, damit sie auf den Markt gestellt werden kann.

Ausschlaggebende Fakten:
Die Vorteile dieser Engine liegen darin, dass ein Game möglichst einfach erstellt werden kann und anschliessend auf unterschiedliche Plattformen deployed werden kann. Dies ist zwar sehr praktisch, kommt jedoch für diese Bachelorarbeit nicht in Frage, da das Projektteam die Game-Programmierung auf Android kennen lernen möchte. Ebenfalls ist die Engine nicht kostenlos.

Links:
http://www.cuttlefishengine.com/

3.7.3.4 Google App Inventor

Beschreibung:
App Inventor ist eine Anwendung von Google, die es erlaubt, Software-Anwendungen für das Betriebssystem Android zu programmieren.

Der App Inventor verwendet eine grafische Schnittstelle, die den Benutzeroberflächen Scratch und StarLogo TNG sehr ähnlich ist und es Anwendern per Drag and Drop ermöglicht, grafische Blöcke (grafische Programmiersprache) zu einer Anwendung für Mobiltelefone mit dem Android-System zu erstellen.

Seit dem 31. Dezember 2011 wurde der Support für App Inventor im Rahmen der Auflösung von Google Labs beendet. Das Projekt soll jedoch vom MIT weiter betrieben werden.

Freeware:
Ja

Ausschlaggebende Fakten:
Da der Support für diese Engine eingestellt wurde und die Weiterführung des Projekts noch nicht sicher ist, kommt diese Engine nicht in Frage.

Links:
http://appinventor.googlelabs.com/about/

3.7.3.5 Libgdx

Beschreibung:
Libgdx ist eine Spielentwicklungsframework für 2D und 3D Spiele. Das Framework ist sehr mächtig. Erstellte Spiele können nicht nur auf andere Mobile-Plattformen portiert werden, sondern auch als Desktopanwendung auf Computer und umgekehrt. Ein Spiel, welches als Desktopanwendung entwickelt wurde, kann mittels 6 Zeilen Code auf mobile Plattformen portiert werden.

Freeware:
Ja

Ausschlaggebende Fakten:
Die Vorteile dieser Engine liegen darin, dass ein Game möglichst einfach erstellt werden kann und anschliessend auf alle Plattformen (Mobile und Desktop) deployed werden kann. Dies ist zwar sehr praktisch, birgt jedoch einen Nachteil, da das Projektteam die Game-Programmierung auf Android kennen lernen möchten.

Links:
http://code.google.com/p/libgdx/
http://libgdx.badlogicgames.com/

3.7.3.6 Mages Engine

Beschreibung:
Mages ist eine Client / Server Game Engine für Android. Ebenfalls ist eine Schnittstelle für Windows Mobile in Bearbeitung. Entwickler können sich hauptsächlich auf die Kern-Logik des Spiels und die GUI-Entwicklung konzentrieren. Der Netzwerk-Multiplayer Aspekt wird stark von der Engine unterstützt.

Freeware:
Ja

Ausschlaggebende Fakten:
Da die Engine auf Multiplayer Games ausgerichtet ist und das weitere Bestehen der Engine nicht sicher ist, wird diese Engine in dieser Bachelorarbeit nicht verwendet.

Links:
http://code.google.com/p/mages/

3.7.3.7 Rokon

Beschreibung:
Eine erste Version von Rokon wurde bereits vor längerer Zeit entwickelt und ist deshalb nicht mehr aktuell. Nachdem der Entwickler längere Zeit an der libgdx Engine mitgearbeitet hat, wird er nun Rokon neu konzipieren und dabei auf Teilen von libgdx aufbauen. Die neue Version soll sehr leistungsstark und flexibel werden. Ebenfalls soll mit Rokon auf die Physik-Engine Box2D zugegriffen werden können.

Freeware:
Ja

Ausschlaggebende Fakten:
Zurzeit ist nicht sicher, ab wann die Engine vollumfänglich zur Verfügung steht. Ebenfalls wird die Community zu Beginn eher klein sein. Dieser Punkt ist dem Projektteam jedoch sehr wichtig.

Links:
http://rokonandroid.com/
http://code.google.com/p/rokon/

3.7.4 Entscheid Game-Engine

Das Projektteam hat sich anhand der gesammelten Informationen für die AndEngine entschieden. Diese bietet, nach Meinung des Projektteams, die beste Abdeckung der gewichteten Kriterien für diese Arbeit.

Folgende Features bringt die AndEngine mit sich:

- 2D Engine
- Anbindung an Physik Engine (Box2D)
- OpenGL
- Grosse Community
- Kostenlos

Die AndEngine bietet im Android Market eine kostenlose App an, die durch mehrere unterschiedliche Beispiele aufzeigt welche Möglichkeiten die Engine bietet. Das Projektteam konnte sich mit Hilfe dieser App in kurzer Zeit über folgende, für das Projekt notwendige, Teilbereiche der Engine ein Bild machen.

- Animationen
- Touch Input
- Physik
- Grafiken
- Audio

Die App ist unter dem Namen „AndEngine – Beispiele“ im Android Market zu finden.

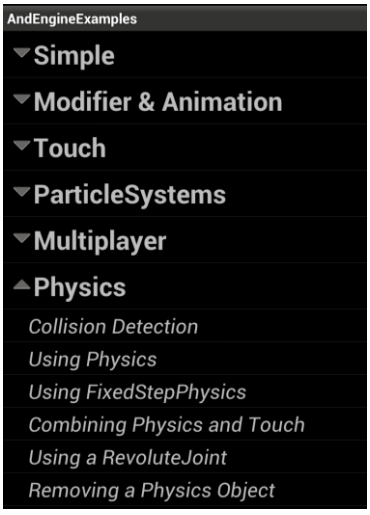


Abbildung 8: AndEngine – Beispiele

Auf diese Beispiels-App wird innerhalb der Dokumentation mehrere Male verwiesen, da die Beispiele gut strukturiert und einfach dargestellt sind.

Zusätzlich wurden bereits Lehrbücher zur Android Game-Programmierung geschrieben (Learning Android Game Programming siehe Literaturverzeichnis), die ebenfalls die AndEngine verwenden. Dies war ein weiteres Argument, welches zeigt, dass die Engine etabliert ist und auch gute Chancen hat in Zukunft weiterentwickelt zu werden.

3.8 Übersicht AndEngine

3.8.1 Entwickler

Die AndEngine wurde von Nicolas Gramlich entwickelt. Anschliessend hat er die Engine als Open-Source Projekt für jeden zugänglich gemacht.

Sein Ziel war es, eine kostenlose und einfache Engine zu entwickeln, welche auch für Neueinsteiger die Möglichkeit bietet sich in diesem rasant wachsenden Markt behaupten zu können. Dabei soll sie jedoch die Kreativität von erfahrenen Entwicklern nicht einschränken.

Abbildung 9: AndEngine Logo (Quelle: andengine.org)

3.8.2 Statistik[6]

Im Dezember 2011 waren bereits über 200 Spiele auf dem Android Market verfügbar, welche auf der AndEngine basieren.

Eine Liste der Spiele ist hier ersichtlich:
http://wiki.andengine.org/List_of_Apps_and_Games

Es gibt mehrere Spiele mit bereits über 1 Mio. Downloads und einige mit über 5 Mio. Downloads.

Unter folgendem Link sind viele Videos bekannter AndEngine-Spiele zu finden. Dabei wird meistens die Anzahl Downloads sowie das User-Rating angegeben:
http://www.andengine.org/blog/

3.8.3 Physik-Engine

AndEngine unterstützt grundlegende physikalische Möglichkeiten, welche verwendet werden können. Der Physik Bereich stellt bei AndEngine jedoch nicht das Hauptaugenmerk dar. Vielmehr besitzt AndEngine eine bequeme Anbindung an eine bereits vielfach verwendete Physik-Engine namens Box2D. Weitere Informationen über Box2D sind im nächsten Kapitel zu finden.

3.8.4 Links

Website: http://www.andengine.org
Forum: http://www.andengine.org/forums/
Blog: http://www.andengine.org/blog
Download Source Code: http://code.google.com/p/andengine/
Tipps und Tutorials: http://theorynine.com/labs/andengine-tips-and-tutorials/

[6] [LeAnGaPro]

3.9 Übersicht Physik-Engine Box2D[7]

3.9.1 Entwickler

Box2D ist eine kostenlose Open Source Physik-Engine, welche von Erin Catto entwickelt wurde. In 2006 wurde die Engine erstmals vorgestellt. Nachdem Box2D zuerst bei Sourceforge angeboten wurde, wird die aktuelle Box2D Version zurzeit bei Google Code gehostet.

Abbildung 10: Box2D Logo (Quelle: http://box2d.org/)

3.9.2 Eigenschaften

Da Box2D in C++ geschrieben ist, kann die Physik-Engine überall verwendet werden, wo ein C++ Compiler verfügbar ist. Dies führte dazu, dass Box2D bisher bei Spielen auf folgenden Plattformen eingesetzt wurde:

- Nintendo DS
- Wii
- Android
- iPhone

Mit Box2D als Physik-Engine sind viele Spiele entstanden, die grossen Erfolg haben. Die wohl bekanntesten Beispiele sind:

- Angry Birds
- Crayon Physics Deluxe
- Tiny Wings

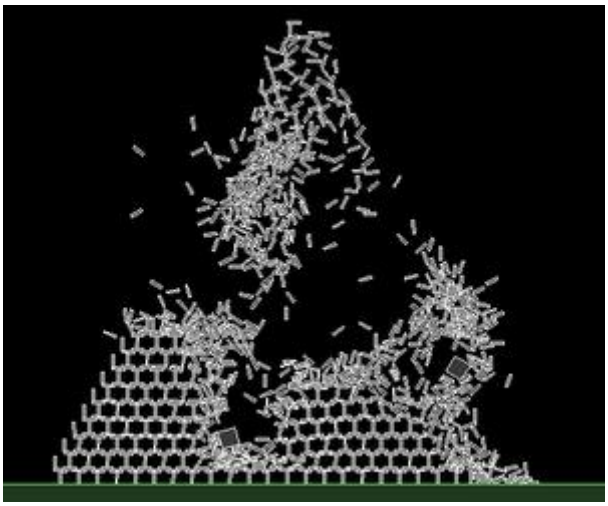


Abbildung 11: Box2D (Quelle [WikiEnBox2D])

Ebenfalls wurde Box2D auch bei vielen Flash Games verwendet, die im Internet angeboten werden.

Neben der AndEngine gibt es weitere Game-Engines, die Box2D verwenden. So zum Beispiel die zuvor erwähnten Engines Corona und Crocos2D.

3.9.3 Features

Eine Liste mit allen Features von Box2D ist unter http://box2d.org/about/ zu finden.

3.9.4 Links

Webseite: http://www.box2d.org/
Google Code: http://code.google.com/p/box2d/
AndEngine Box2D Extension:
https://github.com/nicolasgramlich/AndEnginePhysicsBox2DExtension

[7] [WikiENBox2D]

4 Anforderungsanalyse

In diesem Kapitel werden die funktionalen und nichtfunktionalen Anforderungen des Spiels diskutiert. Dabei wird nicht auf das Game-Konzept im Speziellen eingegangen. Es umfasst Use Cases und Features, welche nicht zwingend implementiert werden.

4.1 Personas

Die folgenden Personas stehen für drei Benutzergruppen, die einen Grossteil der heutigen Smartphone-Benutzer ausmachen. Einige Erkenntnisse aus der Game-Studie sind in diese Personas eingeflossen und haben ebenfalls dazu beigetragen, die Anforderungen an das Game zu definieren.

4.1.1 Persona 1 (Gustav Gelegenheitsspieler)

- Gustav ist 25 Jahre alt arbeitet auf einer Bank.
- Er ist Gelegenheitsspieler und fährt täglich ca. 1h mit Bus/Bahn.
- Das Android-Phone ist sein treuer Begleiter.
- Er ist gerne bereit für ein gutes Spiel/ App ein „paar Franken“ zu bezahlen.
- Während der Bahnfahrt arbeitet er normalerweise, aber das Smartphone ist ein guter Zeitvertreib für zwischendurch, beispielsweise bei Wartepausen an Bahnhöfen und Bushaltestellen.
- Er mag es nicht, wenn das Spiel zu viele Videos und Dialoge zur Storyline zeigt. Er will die Zeit nutzen, um zu spielen.

4.1.2 Persona 2 (Sabrina Spielsüchtig)

- Sabrina ist 18 Jahre alt und macht zurzeit eine Lehre zur Kauffrau.
- Sabrina spielt seit ihrer Kindheit Computerspiele. Da sie, wie bereits erwähnt, eine Lehre absolviert, hat sie kein Geld, um ihren mittlerweile veralteten Gamer-PC zu ersetzen.
- Sie hat sich jedoch ein Android Smartphone gekauft, um nicht auf ihre tägliche Game-Session verzichten zu müssen.
- Neben ihrer halbstündigen Busfahrt zur Arbeit, spielt sie auch gerne über die Mittagspause.
- Sabrina kostet jedes Spiel bis aufs Letzte aus. Sie freut sich jeweils über Bonuslevels und Belohnungen für sehr gute Spielergebnisse.
- Da ihre Kreditkarte seit dem Smartphone Kauf ohnehin überlastet ist, lädt sie meistens kostenlose Spiele herunter. Ebenfalls spielt sie, aufgrund ihres eingeschränkten Abos, meistens offline um das Datenvolumen gering zu halten.

4.1.3 Persona 3 (Karl Knobel)

- Karl ist 52 Jahre alt und leitet eine eigene Firma, die im Design von Logos und Werbematerial tätig ist.
- Karl hält nichts von Computerspielen und hatte deshalb auch noch nie einen Gamer-PC oder eine Spiel-Konsole.
- Karl löst sehr gerne Sudoku und andere Rätsel.
- Da Karl selbstständig ist, hat er sich vor einem Jahr ein Smartphone gekauft, um vom Zug aus immer auf seine Termine und Kontakte zuzugreifen.

- Beim letzten monatlichen Besuch im Rätselclub hat ihm ein langjähriger Freund sein neues Smartphone gezeigt. auf welchem er bereits einige Knobelspiele installiert hat. Seither sucht sich Karl ganz gezielt Android-Spiele, bei denen er sein Talent als Rätsellöser ausleben kann.
- Karl liebt die Herausforderung und tauscht sich nun monatlich, mit seinem Freund aus dem Rätselclub, über aktuelle Highscores seiner wenigen Smartphone-Games aus.
- Durch seinen Beruf als Grafiker achtet Karl sehr auf die Gestaltung der Games. Schlecht designte Smartphone-Spiele kommen für ihn nicht in Frage.

4.2 Use Cases

4.2.1 Use Case Diagramm

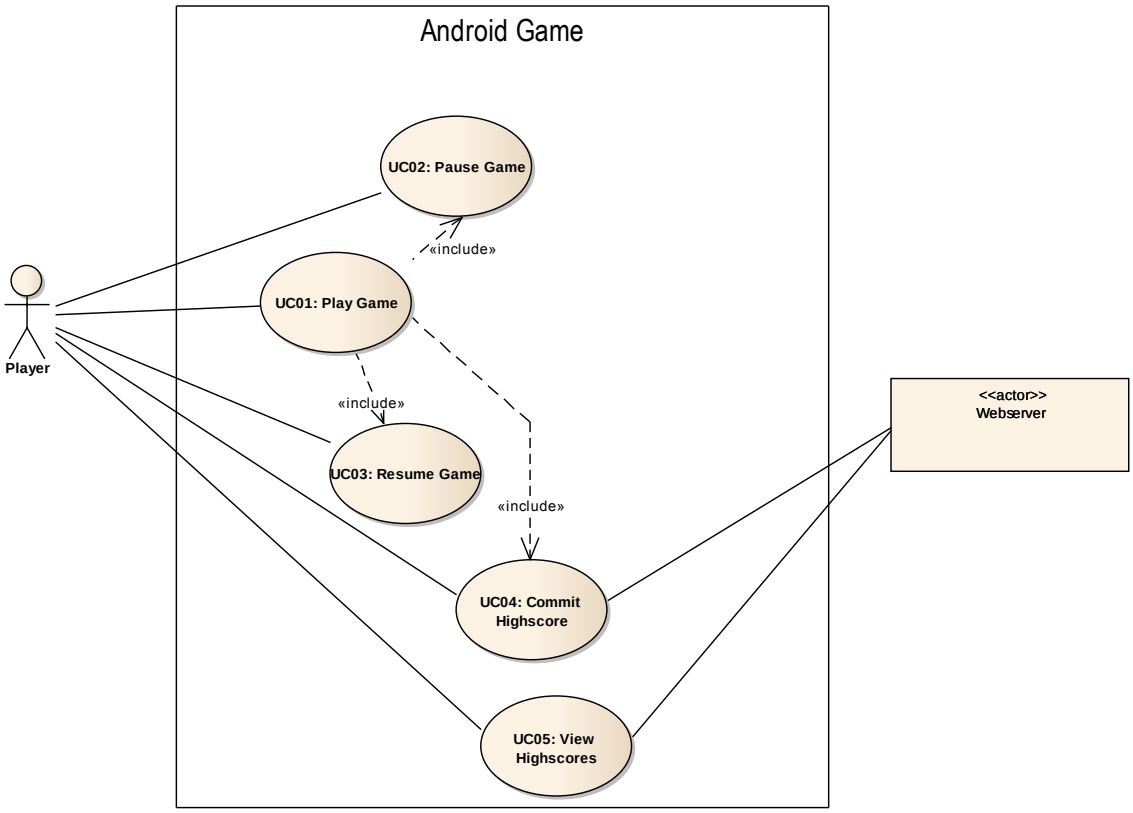


Abbildung 12: Use Case Diagramm

4.2.2 Aktoren

Aktor	Beschreibung
Player	Spieler des Android Games
Webserver	Webserver mit Service zur Verwaltung von Highscores

Tabelle 1: Aktoren

4.2.3 Use Case Kurzbeschreibungen

Use Case	Beschreibung
UC01: Play Game	Spieler startet und spielt das Game.
UC02: Pause Game	Spieler / System pausiert das Game
UC03: Resume Game	Spieler nimmt das Game nach einer Pause (UC02) im gespeicherten Zustand wieder auf.
UC04: Commit Highscore	Highscore wird an Webserver geschickt.
UC05: View Highscores	Bestenliste wird vom Webserver geladen.

Tabelle 2: Use Case Beschreibungen

Der genaue Ablauf des UC01: Play Game wird durch das Game-Konzept sowie die Storyline definiert. Diese werden im Kapitel 5 Game-Design ausführlich beschrieben.

4.3 Optionale Anforderungen

Die Use Cases UC04 und UC05 sind optionale Anforderungen. Eine Bestenliste mit Highscores wäre für eine Publikation auf dem Market ein schönes Feature, jedoch wird der Ausarbeitung und Implementation des Games mehr Priorität zugeschrieben.

4.4 Nichtfunktionale Anforderungen

Da das Game dem Genre Casual Games angehören sollte, lassen sich speziell die nichtfunktionalen Anforderungen Benutzbarkeit und Attraktivität wie folgt definieren.

4.4.1 Benutzbarkeit

- NF01: Das Game soll in der Bedienung intuitiv und einfach sein, wo nötig werden Hinweise und Hilfen in das Game eingebaut.
- NF02: Das Game wird an bestehende und bewährte Designs angelehnt, Stichwort Best Practices.
- NF03: Das Game soll offline spielbar sein.
- NF04: Das Game soll grundsätzlich auf jedem Android-Phone gespielt werden können.

4.4.2 Attraktivität

- NF05: Das Game soll grafisch ansprechend sein.
- NF06: Das Game soll ein breites Publikum adressieren.

4.4.3 Sicherheit

- NF07: Das Game speichert keine sensiblen Daten.
- NF08: Das Game verlangt keine unnötigen Berechtigungen vom System.

4.4.4 Reife

- NF09: Das Game muss nicht zwangsläufig bis zur Marktreife entwickelt werden.

4.4.5 Wartbarkeit

- NF10: Der gesamte Programm-Code sowie die Dokumentation im Code sollen in Englisch verfasst sein.
- NF11: Wo es sinnvoll (Aufwand und Nutzen) erscheint, sollen Unit-Tests geschrieben werden.
- NF12: Konfigurationen sollen, wenn möglich, in externe Dateien ausgelagert werden.

4.4.6 Erweiterbarkeit

- NF13: Levels und Kategorien sollen möglichst ohne Programmieraufwand geändert und hinzugefügt werden können.

4.4.7 Umgebung (Vorgaben)

- NF14: Die App soll auf Android Phones mit Version 2.2 und höher laufen.

5 Game-Design

In diesem Kapitel wird die anfänglich nur vage definierte Spielidee ausgearbeitet. Die Storyline definiert dabei den Ablauf und das höhere Ziel des Games. Im Game-Konzept werden wichtige Punkte der Storyline konkretisiert. Dieses Dokument ist während der ganzen Arbeit gewachsen, da immer neue Ideen rund um das Spiel aufkamen. Dabei beinhaltet das Kapitel auch viele Ideen, welche während der Arbeit noch nicht umgesetzt wurden, jedoch in Bezug auf die Weiterentwicklung des Games festgehalten wurden. Dies ist damit zu erklären, dass das hauptsächliche Ziel dieser Arbeit darin bestand, möglichst viele Aspekte der Game-Programmierung kennen zu lernen.

5.1 Spielidee

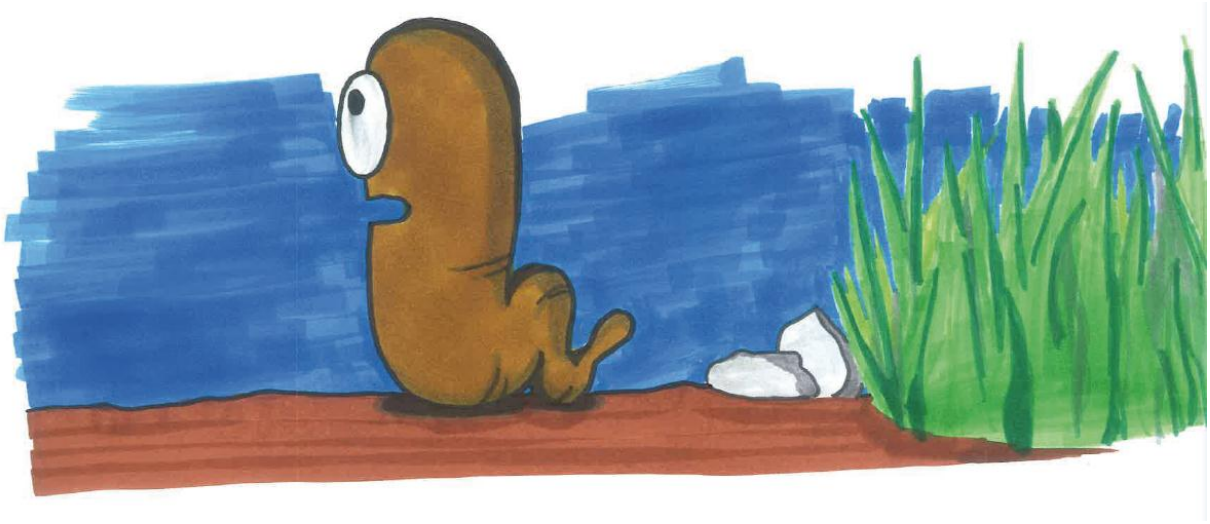


Abbildung 13: World Sketch [SanEl]

Die Spielidee besteht darin, dass der Spieler versuchen muss, den Wurm mit diversen Hilfsmitteln vor unterschiedlichen Bedrohungen bzw. Angriffen zu schützen. Damit ein Level erfolgreich beendet werden kann, darf der Wurm während der gesamten Dauer des Angriffs nicht mit den angreifenden Elementen (Gift, Wasser, Hagel usw.) in Berührung kommen. Wird der Wurm getroffen, kann der Spieler das Level nochmals versuchen.

Grundsätzlich kann ein Level in wenigen Minuten zu Ende gespielt werden. Die Bedienung ist intuitiv und wird, falls nötig, mit eingeblendeten Hilfen erläutert. Das Game wird sich somit im Genre der Casual Games ansiedeln.

5.2 Storyline

Die Unterkapitel Ausgangslage, Suche und Ende zeigen den chronologischen Verlauf der Story des Games auf. Am Schluss jedes Kapitels ist jeweils in kursiver Schrift beschrieben, wie dem Spieler die Storyline während des Games erzählt werden kann.

5.2.1 Teil 1: Ausgangslage

Wilson, der Wurm, lebt mit seiner Freundin Winnie auf dem Areal eines Bauernhofs unter der Erde. Die beiden Würmer waren bisher nur in der Erde und kennen sich somit an der Oberfläche überhaupt nicht aus.
Wilson und Winnie befinden sich bei einem gemütlichen Abendessen. Dabei muss Wilson hilflos mitansehen, wie Winnie entführt wird. Der Bauer gräbt mit seinem Spaten den Boden um und reisst Winnie dummerweise aus der Erdhöhle. Wilson ist schockiert und starr vor Schreck. Aus Angst, selber von einer Bedrohung getroffen zu werden, sind seine Augen von diesem Augenblick an stets nach oben gerichtet.

Nachdem er den ersten Schock überstanden hat, wird ihm klar, was er zu tun hat. Seine Mission, auch wenn es seine letzte sein sollte, besteht darin, seine Winnie zu finden und wieder zurück zu holen.

Diese Informationen werden dem Spieler zu Beginn des Games mit Hilfe von Bildfolgen oder Videos vermittelt.

5.2.2 Teil 2: Suche

Die Reise an der Oberfläche bereitet ihm viele Probleme und bringt ihn in gefährliche Situationen, welche er ohne die Hilfe des Spielers nicht überleben würde.

Die Suche nach Winnie führt Wilson durch verschiedene Schauplätze. So startet er seine Suche in der näheren Umgebung des Bauernhofs. Anschliessend können beliebige weitere Schauplätze in die Storyline eingebunden werden.

Während dem Game können vereinzelt Bilder oder Bildfolgen eingeblendet werden, um die Story weiter zu erzählen. Beispielsweise beim Freischalten eines neuen Schauplatzes.

5.2.3 Teil 3: Ende

Nach einer langen und gefährlichen Reise durch unterschiedliche Schauplätze findet Wilson seine Winnie unversehrt wieder. Und wenn sie nicht gestorben sind, dann leben sie noch heute.

Auch das Ende wird mithilfe von Bildfolgen erzählt.

5.2.4 Charaktere

In den folgenden zwei Unterkapiteln werden die beiden Hauptcharaktere des Games vorgestellt.

5.2.4.1 Wilson

Die Hauptfigur des Games ist Wilson. Er muss hilflos mitansehen, wie seine Freundin Winnie entführt wird. Das Game begleitet ihn auf seiner Rettungsmission. Während des ganzen Games schaut er ängstlich nach oben, aus Angst selber Opfer eines Angriffs zu werden.

Abbildung 14: Skizze Wilson [SanEl]

5.2.4.2 Winnie

Winnie wird unbeabsichtigt von einem Bauer aus der Erde gerissen und von ihrem Geliebten getrennt.
Sie hat keine Ahnung, wohin sie gebracht wird. Sie ist ihrem Schicksal hilflos ausgeliefert.

Abbildung 15: Skizze Winnie [SanEl]

5.3 Game-Konzept

5.3.1 Schauplätze

Da nicht definiert ist wie die Suche genau verläuft, kann das Game jederzeit mit neuen Schauplätzen ergänzt werden.

Die unterschiedlichen Schauplätze erlauben es auch verschiedene Bedrohungen und Gegenstände in das Game einzubauen, um somit das Game unterhaltsam und abwechslungsreich zu halten. Die Schauplätze werden im weiteren Verlauf als Level-Kategorien bezeichnet. Grund dafür ist der Aufbau des Spiels, wobei eine Level-Kategorie immer aus mehreren ähnlichen Levels besteht.

Folgende Schauplätze wären denkbar:

Level-Kategorie	Schauplatz	Bedrohung	Gegenstände / Hilfsmittel
1	Bauernhof	Bauer (Schleuderstreuer)	Holz, Steine, Käfer, …
2	Stadt	Wetter, Abfall	Becher, Zigaretten, Krümel, …
3	Wilder Westen	Pfeile, Steinschlag	Steine, Stroh, Holz, Kakteen, …
4	Weltall	Meteoriten, Aliens	Steine, Aliens, Metall, …
5	Unterwasser	Netz, Fische	Korallen, Pflanzen, Steine, …

Tabelle 3: Schauplätze

Wie bereits erwähnt besteht jeder Schauplatz aus Gegenständen, welche zum Schutz von Wilson verwendet werden können, aus einer Bedrohung und einem Hintergrundbild.

5.3.2 Eingabemethoden

Wilson kann nicht gesteuert werden. Die Hilfsmittel können mittels Drag & Drop auf die Spielfläche gezogen werden. Hat der Spieler einen falschen Lösungsweg gewählt und merkt dies, kann er das Level über den Restart-Button in den Ausgangszustand zurück versetzen.

5.3.3 Lösungsvarianten

Damit ein Level als gelöst gilt, muss Wilson die Angriffswelle ohne Schaden überstehen. Wilson muss, je nach Level, unterschiedlich in Sicherheit gebracht werden. Folgende Auflistung zeigt mögliche Varianten von Levels auf:

- Wilson wird mithilfe von Gegenständen geschützt.
- Wilson wird durch Gegenstände an einen sicheren Ort gelockt. Auf dem Weg zum sicheren Ort gibt es Schikanen, welche durch Gegenstände passierbar gemacht werden müssen.
- Wilson begibt sich in sicheres Gebiet, welches nicht angegriffen wird (Wilson braucht dabei keinen offensichtlichen Schutz).
- Wilson kann durch spezielle Elemente zum Hüpfen gebracht werden und somit seinen Standort ändern.
- Wilson kann durch Elemente an einen anderen Ort transportiert werden.

5.3.4 Angriffswelle

Ziel des Games ist es, Wilson vor den unterschiedlichen Angriffswellen, welche je nach Schauplatz unterschiedlich sein können, zu schützen. Die Angriffswelle wird erst gestartet, wenn der Spieler den Start-Button anklickt.

Für zukünftige Levels könnten nebst dem Einsatz von Gegenständen auch die anderen Lösungsvarianten in die Levels eingebaut werden. Wenn Wilson zum Beispiel an einen sicheren Ort gelockt werden kann, könnte das Level auch Elemente enthalten, die Wilson vor dem Start der Angriffswelle ausschalten. Dies würde den Schwierigkeitsgrad des Games erhöhen, da bereits vor der Angriffswelle auf Gefahren geachtet werden muss. Denkbar wären Spreng- oder Giftstoffe sowie Klippen, in welche Wilson stürzen könnte.

5.3.5 Physik

Damit aus den Hilfsmitteln ein Schutz aufgebaut werden kann, wird eine Physik-Engine benötigt. Die Physik-Engine sorgt für die Gravitation und ermöglicht es dem Spieler Hilfsmittel zu stapeln und sich stützende Konstruktionen aufzubauen.

Ausserdem kann Wilson beispielsweise durch eine Wippe oder einen Flaschenzug transportiert werden.

5.3.6 Schwierigkeit

Pro Level gibt es nur einen Schwierigkeitsgrad. Jedoch kann die Schwierigkeit der verschiedenen Levels variieren.

5.3.7 Belohnung

Damit der Spieler für seine Leistungen belohnt werden kann, stehen folgende Varianten zur Verfügung:

- Sammeln bestimmter Gegenstände
- Zeitbonus
- Möglichst wenig der zur Verfügung gestellten Gegenstände benutzen.

Denkbar wäre eine Bestenliste, welche die Resultate der Spieler online sammelt sowie lokale Belohnungen wie Accessoires für den Wurm oder Bonuslevels.

5.3.8 Grafik

Die Grafiken und Animationen des Games werden im „Comic-Style“ gezeichnet. Es wird ein ähnlicher Style angestrebt, wie in den bekannten Spielen Angry Birds, Cut the Rope und Doodle Jump. Das Design soll farbig und fröhlich wirken (Abbildung 16: Angry Birds Design (Quelle: play.google.com)).

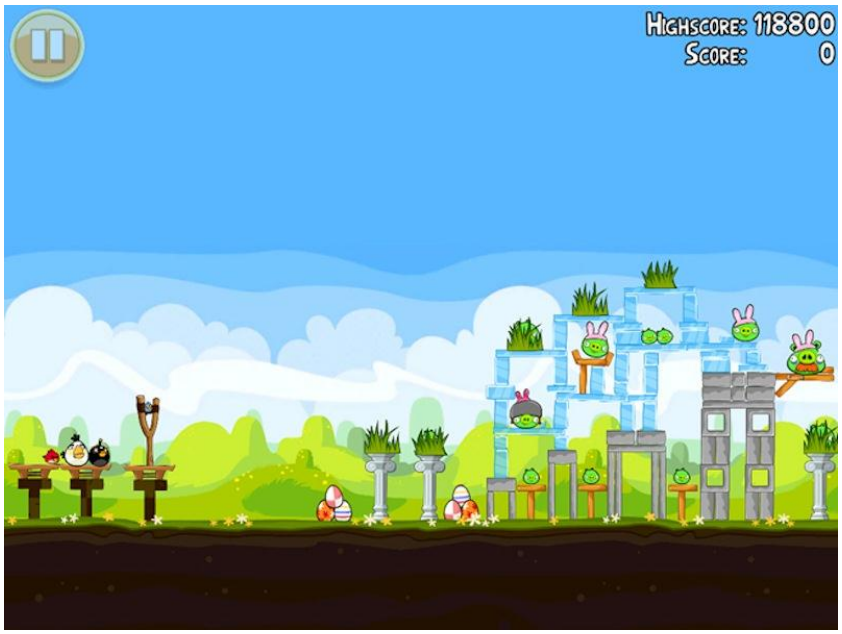


Abbildung 16: Angry Birds Design (Quelle: play.google.com)

5.3.9 Sound

Die Hintergrund-Musik soll ebenfalls fröhlich klingen und den Comic-Style untermalen. Bei einer Angriffswelle kann die Musik etwas bedrohlicher werden.

Aktionen des Spielers und andere Ereignisse werden mit Soundeffekten ausgeschmückt. Beispielsweise beim Platzieren von Gegenständen, wenn sich Wilson bewegt oder wenn er von einem Angriff getroffen wird.

5.3.10 Skizze Game-Konzept

Folgende Skizze veranschaulicht das vorgestellte Game-Konzept.

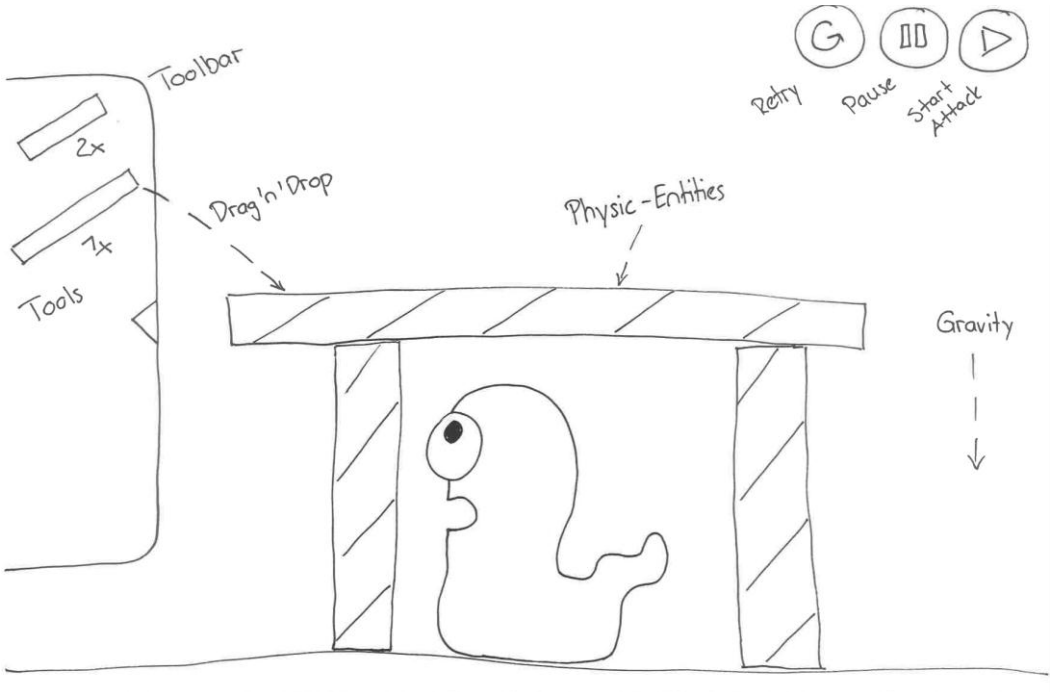


Abbildung 17: Skizze Game-Konzept

5.4 Gameplay

5.4.1 Storyboard

Das folgende Storyboard zeigt die Navigationspfade von Wilson's Adventures vom Start des Spiels, über die Menünavigation bis in die verfügbaren Menüs innerhalb der Levels. Dabei wird das Game durch das Anklicken des App-Icons im Android–Launcher gestartet.

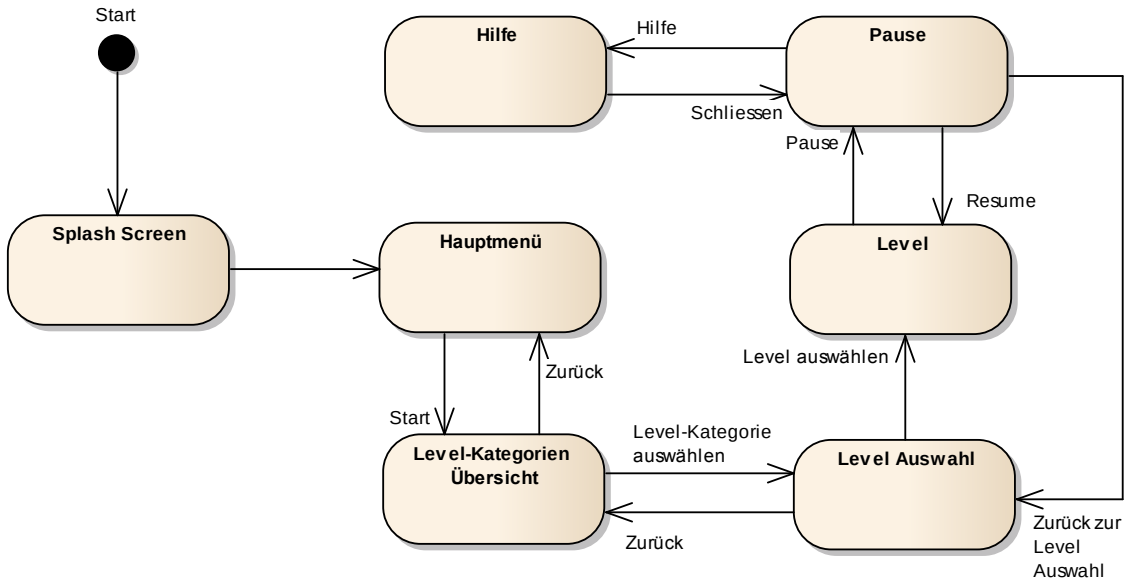


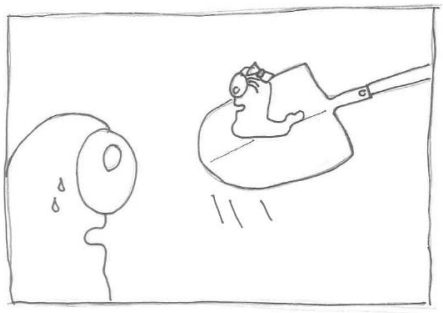
Abbildung 18: Storyboard

1. Das Game startet mit einem Splash Screen. Im Hintergrund werden die benötigten Ressourcen geladen.
2. Sobald alle benötigten Ressourcen geladen sind, wird das Hauptmenü angezeigt. Im Hauptmenü können die Soundeffekte ein- und ausgeschaltet, Herstellerinformationen gelesen sowie das Spiel gestartet werden.
3. Sobald im Hauptmenü das Spiel gestartet wird, erscheint eine Story gebundene Übersicht der verschiedenen Schauplätze.
4. Nachdem ein entsprechender Schauplatz ausgewählt wurde, kann über die Level-Auswahl ein konkretes Level ausgewählt und gespielt werden.
5. Das Game kann über den Pause- oder Back-Button pausiert werden. Wenn sich das Game im Pausen-Modus befindet, wird dem Spieler ein kleines Menü angezeigt, wo er wiederum die Soundeffekte ein- bzw. ausschalten, eine Hilfestellung einblenden oder in die Level-Auswahl zurück navigieren kann.

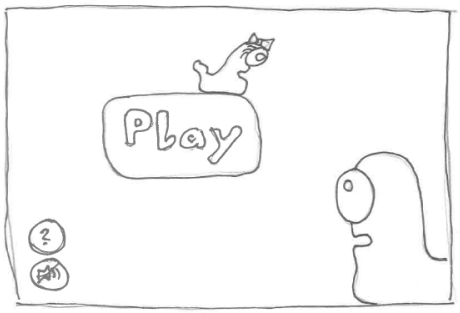
5.4.2 Skizzen der Screens

Die einzelnen Screens, welche im Storyboard verknüpft als Navigationspfade durch das Game erklärt wurden, sind unterhalb mittels Skizzen dargestellt. Dies sollte dem Projektteam helfen, bereits zu diesem Zeitpunkt den Umfang der Menünavigation und der benötigten Grafiken festzulegen.

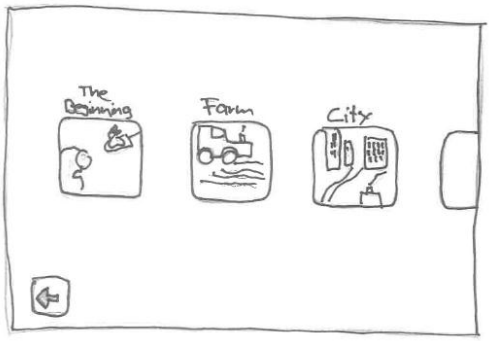
Splash Screen



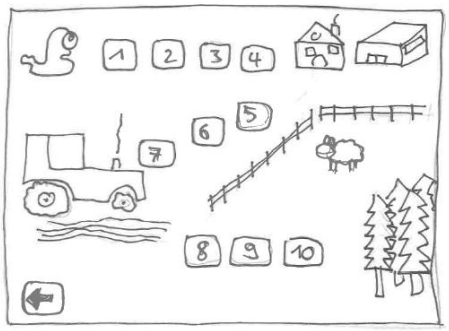
Hauptmenü



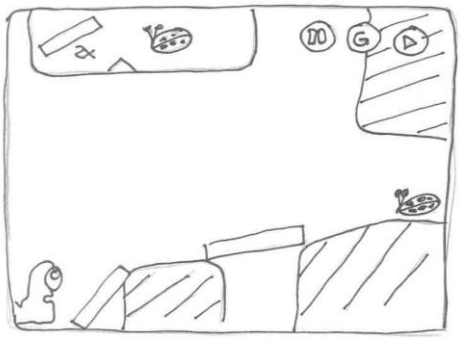
Level-Kategorien Übersicht



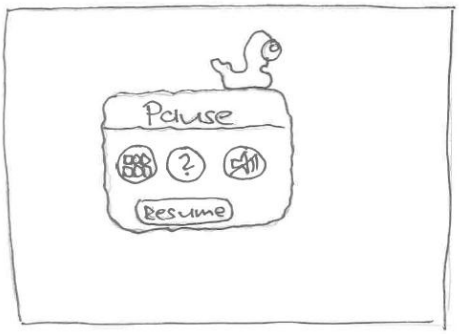
Level Auswahl



Level



Pause-Menü



Hilfe

Tabelle 4: Screen Skizzen

6 Domainanalyse

Die Domainanalyse zeigt den Einfluss des Game-Konzepts auf die Domain-Klassen.

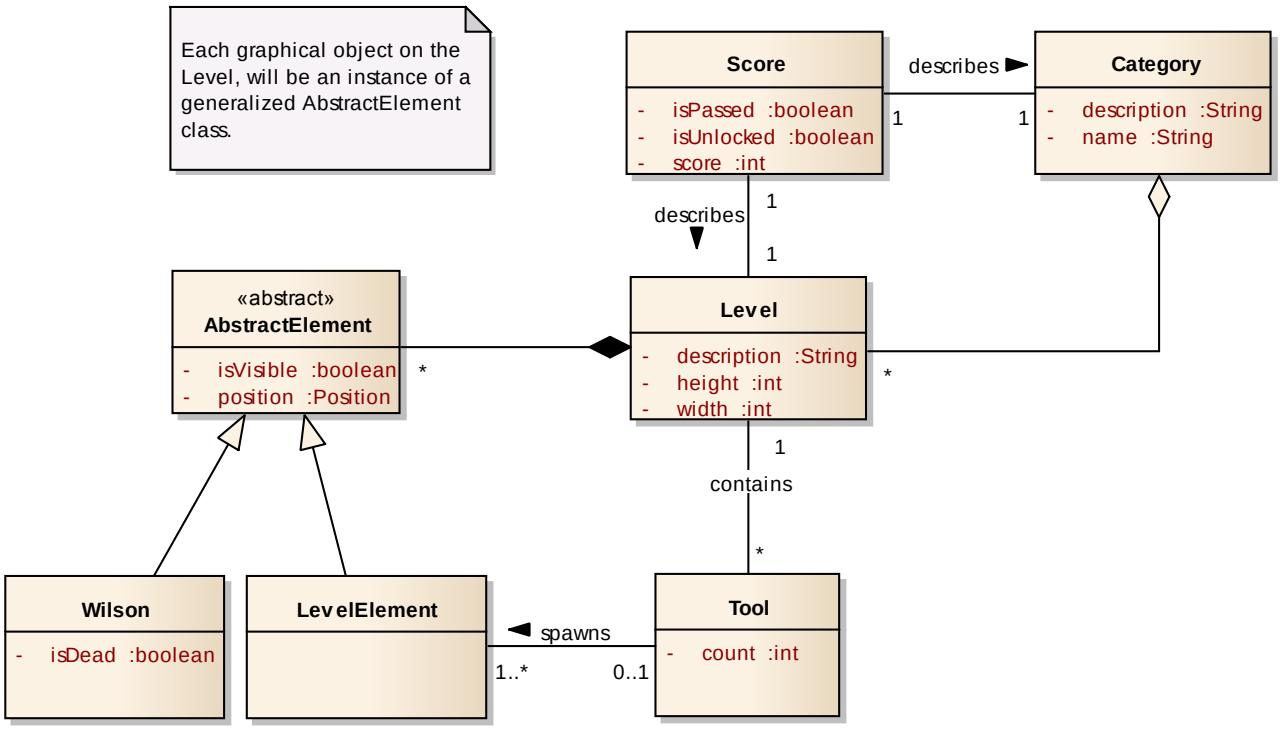


Abbildung 19: Domainmodell

6.1 Component-Entity Modell

Da in der Game-Programmierung mit einem normalen objektorientierten Ansatz bei der Erstellung von Charakteren und Level-Elementen durch die Vielfalt verschiedener Eigenschaften grosse Objekthierarchien entstehen können, wird an dieser Stelle häufig das Component-Entity Pattern[8] angewendet, so auch in der AndEngine.

Alle Elemente, die auf die Spieloberfläche gezeichnet werden sind Entities. Diese können unterschiedliche Komponenten enthalten, welche beispielsweise das Aussehen (Textur), das physikalische Verhalten oder ein spezielles Input-Verhalten (Touch-Listener) für das entsprechende Entity realisieren. Während beim ursprünglichen Ansatz diese Komponenten als Teil der Klasse (z.B. Wilson) implementiert worden wären, sind diese nun in wiederverwendbare Komponenten aufgeteilt.

Das Component-Entity Modell verhindert so tiefe Vererbungshierarchien und es muss nicht zwangsläufig für jedes „spezielle“ Element des Spiels eine eigene Klasse erstellt werden. Die unterschiedlichen Elemente sind Entities mit spezialisierten Komponenten und nicht wie im Domain Modell Unterklassen von einem AbstractElement.

[8] http://gameprogrammingpatterns.com/component.html

7 Grundlagen Game-Programmierung

Dieses Kapitel gibt einen kurzen Einblick in das Game Development sowie in die unterschiedlichen Bereiche der AndEngine und der Box2D Physic-Engine.

7.1 Game-Development

Im Prinzip besteht jedes Spiel aus denselben Bausteinen. Die entsprechenden Module abstrahieren die Kommunikation mit dem zu Grunde liegenden Betriebssystem und umfassen meistens folgende Funktionen:

- **Window Management**
  Dieses Modul ist für die Erstellung und Verwaltung der verschiedenen Screens verantwortlich. Es definiert welcher Screen zu welchem Zeitpunkt sichtbar ist. Dieses Modul ist stark an das System gekoppelt, so wird beispielsweise bei einem Anruf das Game minimiert und die Anruf-Anwendung gezeigt.
- **Input**
  Was wäre ein Spiel ohne Benutzereingaben? Dieses Modul abstrahiert die Spielsteuerung und ermöglicht dem Spieler mit der virtuellen Welt zu interagieren.
- **File I/O**
  Natürlich muss ein Spiel auch Zugriff auf Konfigurations- und Multimedia-Daten haben.
- **Audio**
  Das Audio-Modul ist verantwortlich für das Laden und Wiedergeben von Soundeffekten.
- **Graphics**
  Dieses Modul bestimmt, was auf den unterschiedlichen Screens zu sehen ist. Ausserdem ist es für das Laden der unterschiedlichen Grafiken (Texturen) zuständig.
- **Game Framework**
  Das Game Framework verbindet alle zuvor genannten Module und bietet eine Schnittstelle für die Implementation unterschiedlicher Games.

7.1.1 Game Loop[9]

Das Herz aller Computerspiele stellt der Game Loop mit seinen unterschiedlichen Aufgaben dar. Etwas vereinfacht lässt sich der Game Loop in folgendem Zyklus darstellen (Abbildung 20: Game Loop).

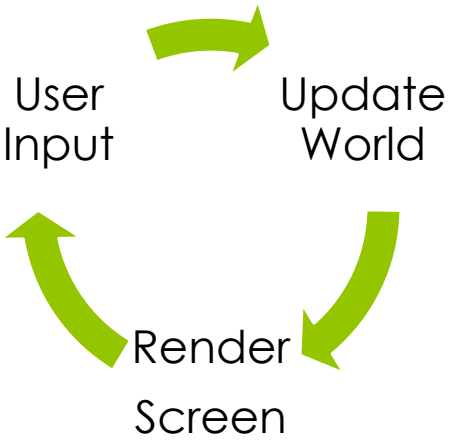


Abbildung 20: Game Loop

Die genauen Schritte des Game Loops können sich je nach Art des Spiels ein wenig unterscheiden. Folgende Schritte sind Teil des Game Loops in „Wilson's Adventures":

- User Input entgegen nehmen
- Game Status gemäss Input aktualisieren
- Game Status gemäss Zeitverlauf aktualisieren
- Kollisionserkennung zwischen Objekten
- Objekte in der Physikwelt aktualisieren
- Anzeige gemäss Game Status aktualisieren
- Anzeige rendern
- Soundeffekte wiedergeben

[9] [BeAnGa]

7.2 AndEngine[10]

Um die AndEngine und ihre Komponenten einfacher verstehen zu können, ist es hilfreich, diese mit einem Filmset zu vergleichen. Diese Analogie ist sehr hilfreich und wird auch im Buch [LeAnGaPro] verwendet um die Funktionen und Aufgaben der unterschiedlichen Komponenten näher zu erläutern.

7.2.1 Camera

Die Kamera definiert, was der Spieler zu einem bestimmten Zeitpunkt zu sehen bekommt. Die Kamera kann, wie im Film, über die aktuelle Szene schwenken oder zoomen. Je nach Situation und Art des Spiels kann der Spieler oder das Spiel selbst die Kontrolle über die Kamera haben.

7.2.2 Scene

Ein Spiel basiert ebenfalls auf verschiedenen Szenen, welche unterschiedliche Schauplätze darstellen. In einem Film gibt es jedoch einen fixen Ablauf, ein Spiel hingegen verläuft abhängig von den Eingaben des Spielers.

7.2.3 Layer

Eine Szene besteht aus unterschiedlichen Schichten, welche von hinten nach vorne (unterste Schicht zuerst) gezeichnet werden. Dadurch kann ein sogenannter 2 ½ D Effekt erzeugt werden.

7.2.4 Sprite

Sprites sind die visuellen Repräsentationen der Schauspieler und Objekte in der Film-Analogie. Sie bewegen sich über die Szenen und stellen oft Charaktere oder interaktive Objekte dar.

7.2.5 Entity

Entity ist die Superklasse aller grafischen Komponenten (Sprites, Scenes, Shapes, Lines usw.) der AndEngine. Alle Entities haben gewisse Eigenschaften wie Farbe, Rotation und Position.

7.2.6 Modifier

Die Eigenschaften von Entities können mit Hilfe von Modifiers verändert werden. Diese Änderungen können sofort oder schrittweise ausgeführt werden.

7.2.7 Texture

Eine Texture entspricht einer Bilddatei. Sie kann aus Performance-Gründen aus mehreren TextureRegions bestehen. So können alle für eine Animation benötigten TextureRegions in eine Texture gepackt und unter einmal vom System geladen werden. In der folgenden Abbildung sind zwei separate TextureRegions in einer Datei zusammengesetzt.

Abbildung 21: Texture

[10] [LeAnGaPro]

7.2.8 Texture Region

Eine TextureRegion definiert, wie ein Sprite zu einem gewissen Zeitpunkt aussieht.

Abbildung 22: TextureRegion

7.2.9 Engine

Die Engine verwaltet die jeweils aktuelle Szene und bestimmt wann und wie die Entities der Szene aktualisiert werden. Sie koordiniert das Handling der Benutzereingaben sowie das Rendering der aktualisierten Grafiken und ist verantwortlich für den zeitlichen Ablauf des Spiels. In der Film-Analogie ist die Engine daher vergleichbar mit einem Regisseur.

7.2.10 BaseGameActivity

Die BaseGameActivity ist das Bindeglied zwischen der AndEngine und dem Android-System. Sie vereint die Engine und den Lifecycle der App. Die BaseGameActivity ist somit der Grundstein aller AndEngine-Games.

7.2.11 Particle System

Mithilfe von Partikel Systemen können aufwändige Effekte dargestellt werden, bei welchen viele kleine Partikel zum Einsatz kommen. Einsatzgebiete für Partikel Systeme sind zum Beispiel die Darstellung von Flüssigkeiten, Feuer oder Rauch.

7.2.12 PhysicsConnector

Um AndEngine Objekte in die mitgelieferte Box2D Physik-Engine aufzunehmen, müssen entsprechende PhysicsConnectors verwendet werden. Diese bilden die rein grafischen Objekte auf physikalische Körper ab, welche dann von der Physik-Engine verwaltet werden können.

7.3 Box2D Physic-Engine

Wie in der Studie bereits kurz erläutert wurde, gibt es eine Erweiterung für die AndEngine, welche die Box2D Physik Engine integriert. Die wichtigsten Klassen der Box2D Engine werden an dieser Stelle aufgelistet und kurz beschrieben.

7.3.1 PhysicsWorld

Die Physikwelt definiert die Rahmenbedingungen für die ihr angehörenden physikalischen Objekte (Bodys) und simuliert das physikalische Verhalten anhand dieser.

7.3.2 Body

Ein Body ist die physikalische Repräsentation eines Sprites aus der AndEngine. Box2D unterstützt primitive physikalische Körper wie Kreise, Rechtecke, Dreiecke, Linien sowie Polygone mit bis zu 8 Eckpunkten. Bodys besitzen diverse physikalische Eigenschaften wie Masse, Reibungskoeffizient, Dichte und Elastizität.

7.3.3 Shape

Box2D stellt die Verwendung von zwei Shapes zur Verfügung. Dazu zählen einerseits der Kreis sowie das Polygon. Wenn ein Body erstellt wird, kann er mit einem Shape über das Fixture verbunden werden. Die Collision Detection eines Box2D Bodys wird anschliessend anhand des Shapes erkannt. Damit auch komplexere Körper in der Physik-Welt abgebildet werden konnten, wurde eine Erweiterung für die Engine geschrieben, welche im Implementations-Kapitel (Seite 61) näher diskutiert wird.

7.3.4 BodyType

Jeder Body besitzt einen entsprechenden BodyType, dieser definiert, wie sich der Body von der Physikwelt beeinflussen lässt. Es gibt drei verschiedene Bodytypen. Ein StaticBody wird fix positioniert (z.B. Abgrenzungen). Ein KinematicBody besitzt ein vordefiniertes Bewegungsmuster. Ein DynamicBody wird hingegen vollständig über die physikalischen Gesetze der Physikwelt bewegt.

7.3.5 Fixture

Die physikalischen Eigenschaften Dichte, Elastizität und Reibungskoeffizient der Bodys werden diesen über sogenannte Fixtures hinzugefügt.

7.3.6 PhysicsFactory

Die PhysicsFactory bietet Methoden um FixtureDefs sowie Bodys für die jeweiligen AndEngine Entities zu erstellen.

7.3.7 PIXEL_TO_METER_RATIO_DEFAULT

Box2D verwendet das Metrische System (Meter, Kilogramm, Sekunden) für die physikalischen Berechnungen. Die Konstante PIXEL_TO_METER_RATIO_DEFAULT konvertiert Grössenangaben von Pixel in Meter.

7.4 Klassenstruktur AndEngine / Box2D

Folgendes Diagramm (Abbildung 23: Klassenstruktur AndEngine / Box2D) zeigt das Zusammenspiel der vorgestellten Klassen exemplarisch. Aus Gründen der Übersicht wurden die meisten Attribute und Methoden weggelassen.

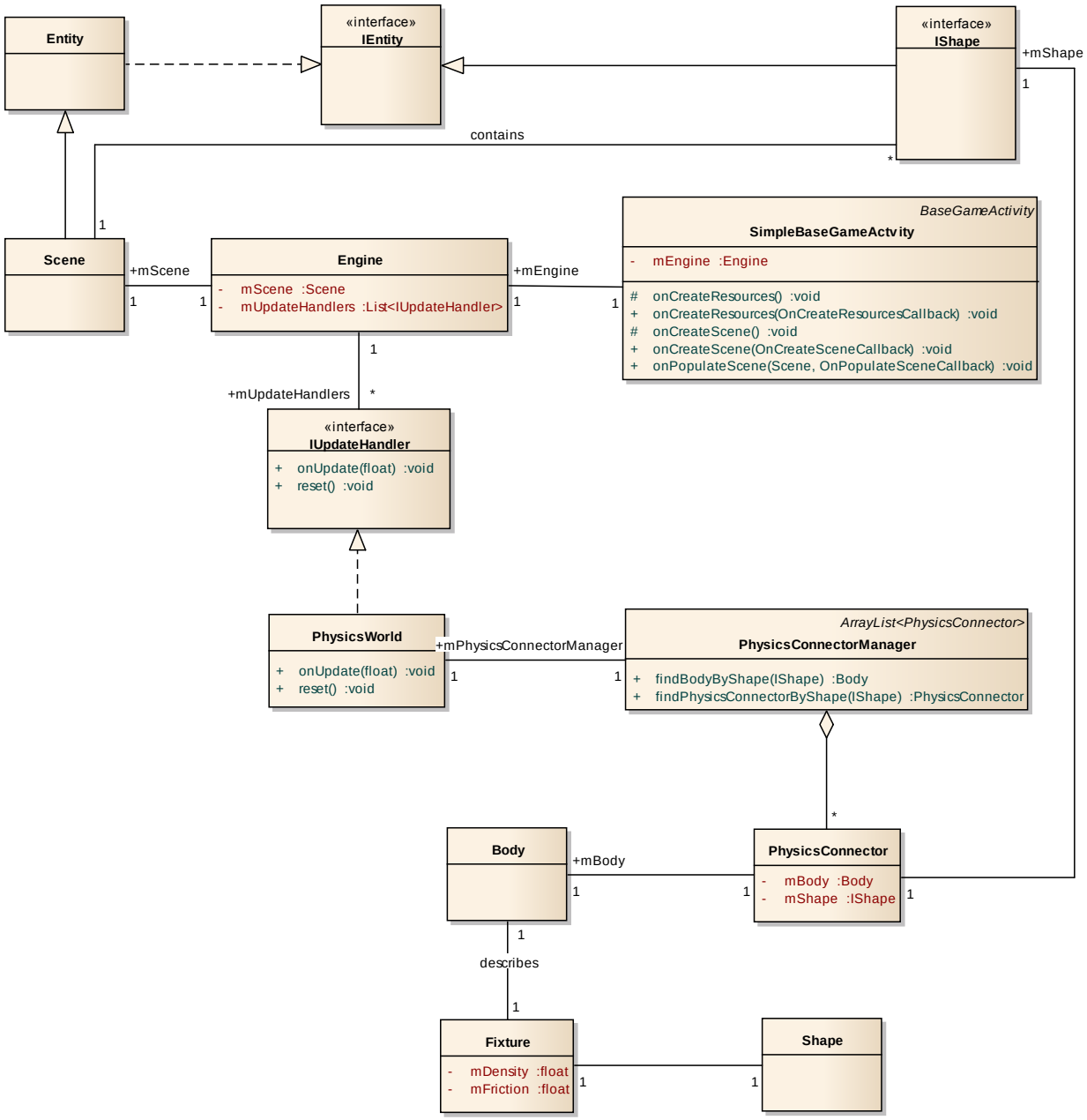


Abbildung 23: Klassenstruktur AndEngine / Box2D

7.5 Beispiel

Anhand eines kleinen Beispiels sollen die einzelnen Komponenten etwas verständlicher gemacht werden.

Im Beispielprojekt wird ein Sprite mit zugehörigem Body erstellt. Dieser wird anschliessend auf die Scene gezeichnet und von der Physikwelt verwaltet. Der Körper fällt aufgrund der Gravitation aus dem sichtbaren Bereich.

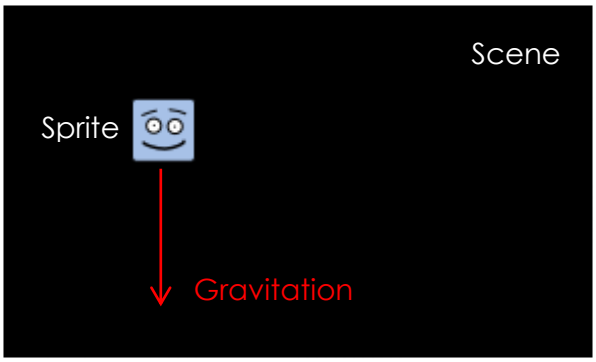


Abbildung 24: PhysicsExample

- **SimpleBaseGameActivity**
  Die PhysicsExampleActivity erbt von der SimpleBaseGameActivity und überschreibt die Implementation von onCreateEngineOptions(), um eine Engine mit den gewünschten Optionen zu instanziieren. Ebenfalls wird eine Instanz einer Camera an die Engine übergeben. Die restlichen Instanz-Variablen der Activity werden zu einem späteren Zeitpunkt erläutert.

```
public class PhysicsExampleActivity extends SimpleBaseGameActivity {
	private static final float CAMERA_WIDTH = 800;
	private static final float CAMERA_HEIGHT = 480;
	private Camera mCamera;
	private BitmapTextureAtlas mTexture;
	private ITextureRegion mTextureRegion;
	private PhysicsWorld mPhysicsWorld;
	private Scene mScene;

	@Override
	public EngineOptions onCreateEngineOptions() {
		mCamera = new Camera(0, 0, CAMERA_WIDTH, CAMERA_HEIGHT);
		return new EngineOptions(true, ScreenOrientation.LANDSCAPE_FIXED, new
				FillResolutionPolicy(), mCamera);
	}
```

Code-Snippet 1: PhysicsExampleActivity

- **Texture und TextureRegion**
  Damit auf die benötigte Textur zugegriffen werden kann, muss diese in der onCreateResources() Methode geladen werden. Dazu muss eine Textur-Instanz über die Klasse BitmapTextureAtlas erzeugt werden. Diese benötigt als Parameter einerseits den TextureManager, welcher Teil der Engine ist, andererseits die Breite und Länge der zu ladenden Textur. Diese Grössen müssen in zweier Potenzen (32, 64, 128, …) angegeben werden. Anschliessend kann die entsprechende Bitmap-Datei (face_box.png), mit Hilfe der Factory, in die TextureRegion geladen werden.

```
@Override
protected void onCreateResources() {
	mTexture = new BitmapTextureAtlas(getTextureManager(), 32, 32);
	mTextureRegion = BitmapTextureAtlasTextureRegionFactory
			.createFromAsset(mTexture, this, "gfx/face_box.png", 0, 0);
	mTexture.load();

}
```

Code-Snippet 2: Texture / TextureRegion

- **Scene und Sprite**
  In der onCreateScene() Methode wird die initiale Scene für die Engine erstellt und dieser übergeben. Dazu wird zuerst eine Scene erstellt, anschliessend wird dieser ein Sprite angefügt. Die ersten zwei Parameter des Sprite Konstruktors sind die X- und Y-Koordinaten, welche die Position des Sprites auf der Scene definieren. Ebenfalls wird an dieser Stelle die zuvor geladene TextureRegion übergeben und so definiert, wie das Objekt aussieht.

```
@Override
protected Scene onCreateScene() {
	mScene = new Scene();
	Sprite sprite = new Sprite(200, 200, mTextureRegion,
				getVertexBufferObjectManager());
	mScene.attachChild(sprite);
	return mScene;
}
```

Code-Snippet 3: Scene und Sprite

- **PhysicsWorld**
  Bislang wurde ein Sprite erzeugt und auf die Scene gezeichnet. Damit sich dieses Sprite physikalisch korrekt verhält, muss es noch in die Physikwelt aufgenommen werden.
  Dazu wird zuerst eine Physikwelt wie folgt erstellt. Der erste Parameter definiert die in der virtuellen Welt herrschende Gravitation und der zweite Parameter dient zur Performance-Optimierung und deaktiviert die physikalischen Berechnungen eines ruhenden Bodys. Anschliessend wird die Physikwelt als Updatehandler auf der Scene registriert. So wird sichergestellt, dass die Auswirkungen der Physik auf der Scene aktualisiert werden.

```
Vector2 gravity = new Vector2(0, SensorManager.GRAVITY_EARTH);
boolean allowSleep = true;
mPhysicsWorld = new PhysicsWorld(gravity, allowSleep);
mScene.registerUpdateHandler(mPhysicsWorld);
```

Code-Snippet 4: PhysicsWorld

- **Body und PhysicsConnector**

  Anschliessend muss ein physikalischer Körper (Body) für das Sprite erstellt werden. Üblicherweise sind die Bodys vom Typ DynamicBody, da diese allen Einflüssen der Physikwelt ausgesetzt sind. Um den Body definitiv in die Physikwelt aufzunehmen wird ein PhysicsConnector erstellt, welcher Body und Sprite verbindet, und an die Physikwelt übergeben.

```
FixtureDef fixtureDef = PhysicsFactory.createFixtureDef(1.0f, 1.0f, 0.5f);
Body body = PhysicsFactory.createBoxBody(mPhysicsWorld, sprite,
                    BodyType.DynamicBody, fixtureDef);
PhysicsConnector physicsConnector = new PhysicsConnector(sprite, body);
mPhysicsWorld.registerPhysicsConnector(physicsConnector);
```

Code-Snippet 5: Body / PhysicsConnector

7.6 Collision Detection

Da die Collision Detection einen wichtigen Punkt in der Game-Logik von Wilson's Adventures darstellt, wird bereits im Grundlagenkapitel darauf eingegangen.

AndEngine bietet eine eigene Methode an, um Kollisionen zwischen Shapes zu erkennen. Sprites verhalten sich dabei gleich wie Shapes. Diese Möglichkeit der Collision Detection bezeichnet man als Shape–Collision.
Wird zusätzlich zur AndEngine die Physik-Engine Box2D verwendet, kann die darin integrierte Collision Detection verwendet werden, welche als Box2D-Collision Detection bezeichnet wird.

7.6.1 Shape-Collision

Wie im Kapitel 7.2 erläutert, stellen Sprites innerhalb der AndEngine die visuelle Repräsentation von einzelnen Objekten dar. Als Beispiel für die Shape-Collision wird ein Projekt der AndEngine-Beispiel-App verwendet (Abbildung 25: Shape-Collision). Die beiden Shapes, das Quadrat und die Linie, drehen sich um die eigene Achse. Mit den beiden Joysticks kann das Sprite mit der Gesicht-Textur bewegt und rotiert werden. Sobald sich das Sprite mit der Gesicht-Textur und eines der beiden anderen Elemente berühren – es findet eine Kollision statt - färbt sich das rotierende Element rot.

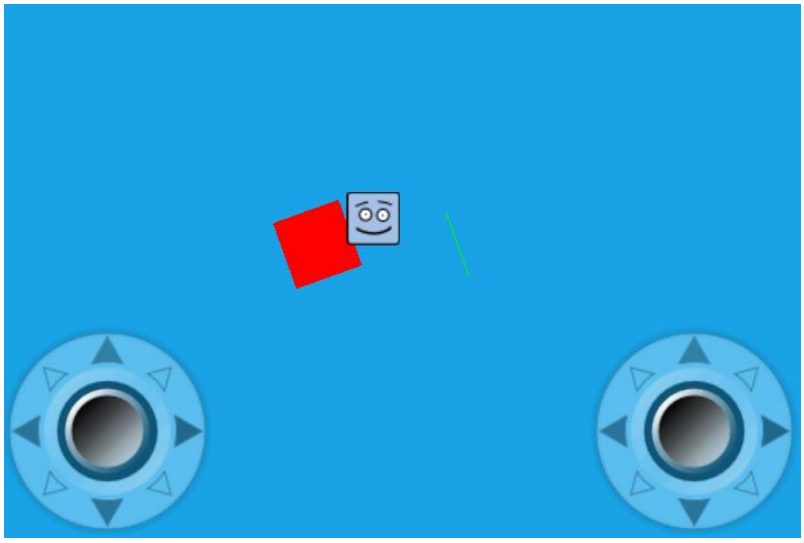


Abbildung 25: Shape-Collision

Um auf die Kollision zu reagieren muss ein UpdateHandler auf der aktuellen Szene registriert werden. Der UpdateHandler definiert für welche Shapes und Sprites überprüft werden soll, ob eine Kollision stattgefunden hat. Diese Überprüfung findet nun bei jedem UpdateEvent auf der aktuellen Szene statt.

Das Code-Snippet 6: Shape Collision zeigt den Einsatz des UpdateHandlers. Das Rechteck und die Linie wurden zuvor als Shapes definiert (centerRectangle und line). Das Sprite mit der Gesicht-Textur hat den Namen face. Sobald nun eine Kollision detektiert wurde, wird über die setColor-Methode die Farbe der Shapes verändert.

```
scene.registerUpdateHandler(new IUpdateHandler() {
	@Override
	public void reset() { }

	@Override
	public void onUpdate(final float pSecondsElapsed) {
		if(centerRectangle.collidesWith(face)) {
			centerRectangle.setColor(1, 0, 0);
		} else {
			centerRectangle.setColor(0, 1, 0);
		}

		if(line.collidesWith(face)){
			line.setColor(1, 0, 0);
		} else {
			line.setColor(0, 1, 0);
		}
	}
});
```

Code-Snippet 6: Shape Collision

7.6.2 Box2D-Collision

Beim Einsatz der Box2D-Engine kann auf die Physik Collision Detection zugegriffen werden. Die Variante mit der in Box2D die Kollisionen detektiert werden, unterscheidet sich grundsätzlich von der Shape-Collision. Hier wird bei der Physikwelt ein ContactListener registriert. Dieser ContactListener wird bei jeder Kollision zwischen physikalischen Objekten (Bodys) aufgerufen. Um einen Overhead für nicht relevante Kollisionen zu verhindern, kann ein Collision Filter eingesetzt werden. Dabei werden die nicht relevanten Kollisionen ignoriert.

Sobald eine Kollision stattgefunden hat, wird die beginContact() Methode mit dem Parameter Contact aufgerufen. Nebst beginContact() steht auch eine Methode für die Nachbearbeitung zur Verfügung. Über das erhaltene Contact Object kann auf die zwei kollidierten physikalischen Körper (Bodys) zugegriffen werden.

Im Code-Snippet 7: Box2D-Collision ist die Zuweisung eines ContactListeners an die Physikwelt ersichtlich. Über das Contact Objekt werden die beiden involvierten Bodys ausgelesen. Anschliessend kann, entsprechend der Game-Logik, die Auswirkung der Kollision bearbeitet werden.

```
this.mPhysicsWorld.setContactListener( new ContactListener(){

	@Override
	public void beginContact(Contact pContact) {
		Body bodyA = pContact.getFixtureA().getBody();
		Body bodyB = pContact.getFixtureB().getBody();
	}
});
```

Code-Snippet 7: Box2D-Collision

8 Architektur und Design

8.1 Architektonische Ziele und Rahmenbedingungen

Das architektonische Hauptziel ist es, möglichst viele Komponenten des Games für zukünftige Projekte wiederverwenden zu können. Da jedoch weite Teile der Logik gamespezifisch sind, betrifft dies vor allem die Komponenten zur Ressourcenverwaltung, Scoring-Mechanismen sowie Parser für Konfigurationsdateien.

8.1.1 Konfigurationsdateien

Damit die Software erweiterbar und einfach konfigurierbar bleibt, werden möglichst viele Informationen in XML-Dateien ausgelagert. Dies umfasst im Speziellen die Definition von Level-Kategorien und Levels. So kann das bestehende Game ohne Programmieraufwand mit zusätzlichen spielbaren Inhalten ergänzt und für den Spieler interessant gehalten werden.

Die unterschiedlichen XML-Dateien werden im Kapitel XML Dateien auf Seite 67 genauer erläutert.

8.2 Logische-Sicht

8.2.1 Schichtung

Die folgende Abbildung zeigt die Schichtenarchitektur von Wilson's Adventures.

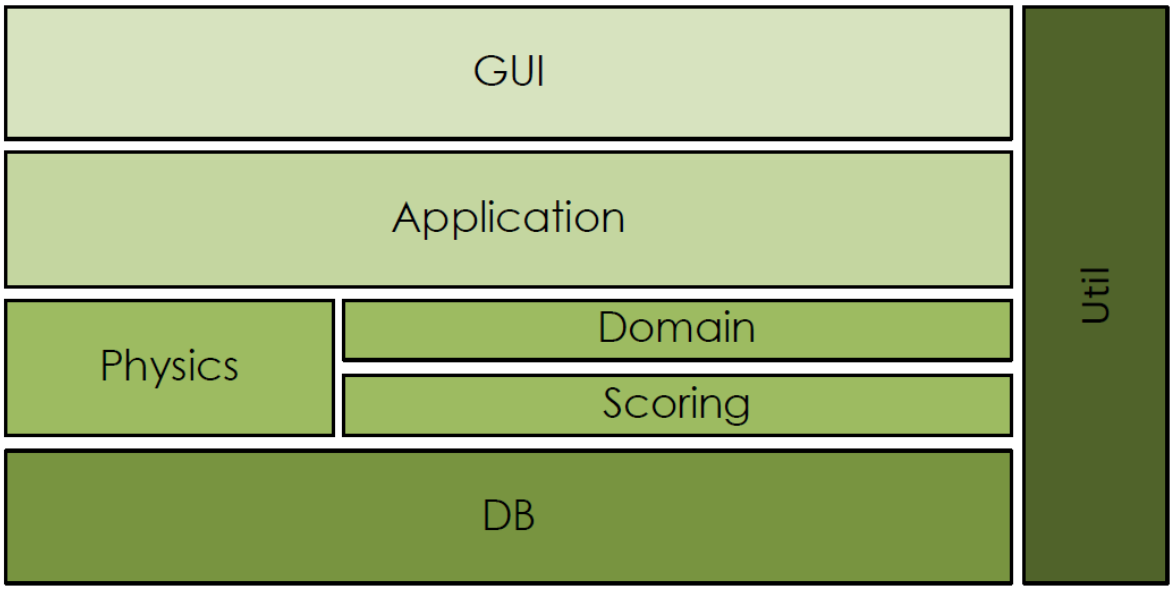


Abbildung 26: Layers

- **GUI**
  Im GUI-Layer befinden sich alle Klassen, welche für den Spieler als GUI-Objekt sichtbar werden. Im Speziellen sind dies Szenen und Sprites.
- **Application**
  Der Application-Layer beinhaltet die Klassen zur Programmsteuerung. Dies sind unter anderem Parser und Klassen zur Ressourcenverwaltung.
- **Physics**
  Im Physics-Layer finden sich alle Klassen wieder, welche für physikalische Repräsentationen oder Berechnungen verwendet werden.
- **Domain**
  Der Domain-Layer beinhaltet die wenigen Domain-Klassen. Dies sind die Klassen Level, Category und Tool.
- **Scoring**
  Der Scoring-Layer umfasst die Logik zur Freischaltung von spielbaren Inhalten sowie die Berechnung von Highscores.
- **DB**
  Im DB-Layer wird der Zugriff auf die SQLite Datenbank gekapselt, in welcher die Highscores sowie die Freischaltung der Levels und Kategorien gespeichert werden.
- **Util**
  Im Util-Layer befinden sich die Game-Options. Diese enthalten Definitionen (Konstanten), welche für das gesamte Game gültig sind (z.B. Kamera-Optionen).

8.2.2 Package Diagramm

Das Package-Diagramm wiederspiegelt die Schichtenarchitektur.

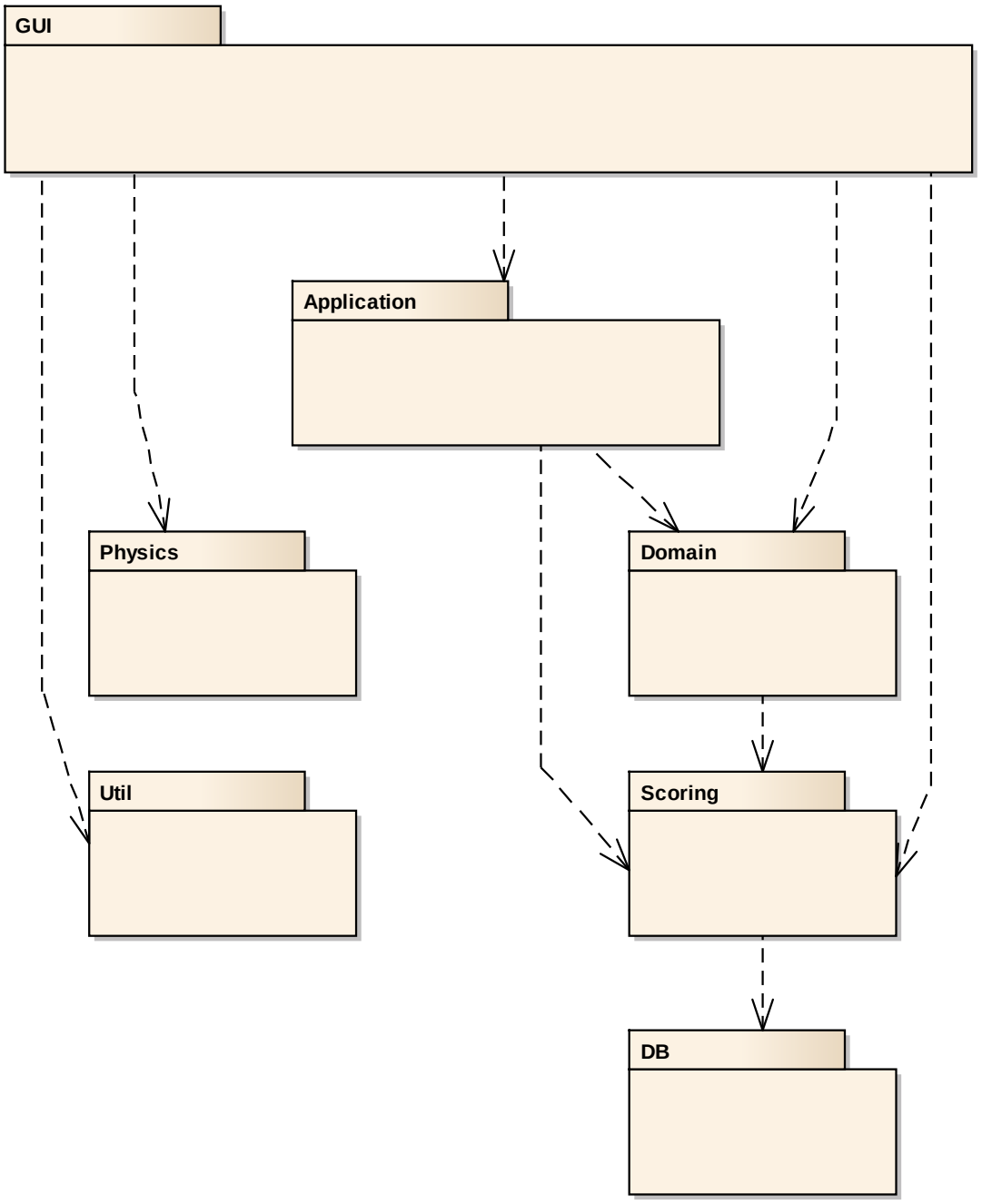


Abbildung 27: Package Diagramm

8.3 Prozess-Sicht[11]

Android instanziiert für jede Applikation einen eigenen Prozess. Dieser beinhaltet standardmässig einen GUI-Thread. Damit bei zeitintensiven Aufgaben nicht die App einfriert, arbeitet Wilson's Adventures mit zusätzlichen Threads. Die Android Library umfasst ein Konzept, welches es erlaubt, elegant eine Multithread-Applikation zu realisieren.

8.3.1 AsyncTasks

Die abstrakte Klasse AsyncTask<Params, Progress, Result> bietet die Möglichkeit eine Aufgabe im Hintergrund auszuführen und sobald diese fertig ist, die Resultate auf dem GUI darzustellen. Das wäre sonst problematisch, weil nur der GUI-Thread Änderungen auf der Benutzeroberfläche vornehmen darf.

Die AsyncTask<Params, Progress, Result> Klasse umfasst folgende Methoden:

Methode	Beschreibung	Ausführender Thread
onPreExecute()	Diese Methode wird direkt nach dem Start des Tasks ausgeführt. In der Regel wird hier der Task vorbereitet und zum Beispiel ein Ladebalken auf der Benutzeroberfläche angezeigt.	GUI-Thread
doInBackground(Params…)	Sobald die Methode onPreExecute() durchgelaufen ist, startet der Hintergrund-Thread. Diese Methode beinhaltet die zeitintensiven Schritte.	Hintergrund-Thread
publishProgress(Progress…)	Diese Methode wird dazu verwendet, um Fortschritte oder Log-Daten aus der doInBackground(Progress…) Methode an die GUI zu melden.	Hintergrund-Thread
onProgressUpdate(Progress…)	Nachdem die publishProgress(Progress…) Methode aufgerufen wurde, wird die onProgressUpdate(Progress…) vom GUI-Thread ausgeführt.	GUI-Thread
onPostExecute(Result)	Diese Methode wird nach Beendigung des Tasks ausgeführt und erlaubt es die Resultate auf der Benutzeroberfläche anzuzeigen.	GUI-Thread

Tabelle 5: AsyncTask-Methoden

In Wilson's Adventures werden zwei AsyncTasks verwendet. Einerseits der RessourceLoaderTask zum Laden der benötigten Ressourcen auf dem Splash-Screen, andererseits der SceneLoaderTask zum asynchronen Laden einer neuen Scene. Diese beiden Tasks werden in der Implementations-Sicht genauer beschrieben.

[11] [SA]

8.3.2 AsyncTaskResult

Die Klasse AsyncTaskResult<T> erlaubt es die Resultate sowie Exceptions (bzw. Fehlermeldungen) aus der doInBackground(Params…) Methode auf der Benutzeroberfläche anzuzeigen. Es ist empfehlenswert als Rückgabe-Typ eines AsnycTasks die AsyncTaskResult Klasse zu verwenden.
Über die Methode exceptionOccured() kann abgefragt werden, ob im Hintergrund-Thread eine Exception aufgetreten ist und allenfalls eine Fehlermeldung auf der Benutzeroberfläche angezeigt werden. Ist keine Exception aufgetreten, kann das Resultat normal verarbeitet werden.
Ohne diese Hilfsklasse wäre es schwierig, passende Fehlermeldungen zu den Exceptions, die in einem Hintergrund-Task auftreten, auf der Benutzeroberfläche anzuzeigen.

Die Klasse AsyncTaskResult<T> beinhaltet folgende Methoden und Konstruktoren:

Methode	Beschreibung
AsyncTaskResult(T result)	Konstruktor zum Erstellen eines „Result-Objects“
AsyncTaskResult(Exception e)	Konstrukor zum Erstellen eines „Exception-Result-Object“
boolean exceptionOccured()	Diese Methode liefert true falls es sich um ein „Exception-Result-Object“ handelt.
T getResult()	Liefert das Resultat, falls vorhanden
Exception getException()	Liefert die Exception, falls vorhanden

Tabelle 6: AsyncTaskResult-Methoden

8.4 Deployment-Sicht

Das Game liegt im Standard Android Application Package (.apk) Format vor und kann somit grundsätzlich auf alle Android-Geräte installiert werden. Aus Kompatibilitätsgründen wird als minimale Version Android 2.2 benötigt.

8.5 Daten-Sicht

Damit der Fortschritt des Spielers persistent gespeichert werden kann, wird eine SQLite Datenbank verwendet. SQLite wird nativ von Android unterstützt. Folgende, in Abbildung 28: ScoreTable ersichtliche, Tabelle wird zur Persistierung der Highscores von Levels und Kategorien verwendet.

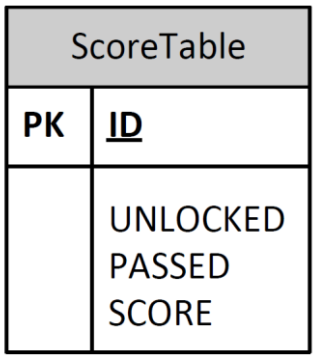


Abbildung 28: ScoreTable

Die beiden Klassen Level und Category leiten von der abstrakten Klasse AbstractScore ab und überschreiben die für die Persistierung benötigten Methoden.

Die genaue Funktionsweise des Scoring-Modells wird anschliessend in der Implementations-Sicht genauer erläutert.

8.6 Implementations-Sicht

8.6.1 Android Manifest

Jede Android Applikation besitzt eine AndroidManifest.xml Konfigurationsdatei. Diese Datei enthält essentielle Informationen über die Applikation, welche vom Android System benötigt werden, um diese zu starten und wunschgemäss zu verwalten. Unter anderem sind folgende Informationen im Manifest enthalten:

- Java Package der Applikation (dient im System als ID für die Applikation)
- Deklaration der Applikations-Komponenten (Activities, Services usw.)
- Berechtigungen, die von der Applikation benötigt werden (z.B. Internetzugang)
- Deklaration von benötigten Hard- und Software Features (Multitouch, Kamera usw.)
- Benötigtes API Level (min. SDK Version)
- Third-Party Libraries

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="ch.hsr.wa"
    android:versionCode="1"
    android:versionName="1.0" >
    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="8" />
    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:name=".GameActivity"
            android:configChanges="orientation"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Code-Snippet 8: AndroidManifest.xml

Wie im nächsten Kapitel beschrieben, besteht das Game nur aus einer Activity. Da auch ansonsten keine speziellen Konfigurationen nötig sind, ist das Manifest kurz und übersichtlich. Die einzige Ausnahme stellt das Attribut android:configChanges="orientation" in der Deklaration der GameActivity dar. Das Attribut bewirkt, dass eine Neuausrichtung des Geräts von der Applikation selber und nicht vom System verwaltet wird. Das System würde den Activity Lifecycle neu durchlaufen. Folglich würde eine neue Instanz der Engine erstellt. Mit dieser Konfiguration wird dies verhindert.

8.6.2 GUI

Im Unterschied zu einer Android-Business-Applikation besteht „Wilson's Adventures" nur aus einer Activity. Der Grund dafür ist, wie bereits erwähnt, dass pro Activity eine Engine erstellt wird und diese an den Lifecycle der Activity gebunden ist. Logischerweise wird aber nur eine Instanz der Engine benötigt und somit auch nur eine Activity.

Für die Gestaltung verschiedener Benutzeroberflächen können in der AndEngine unterschiedliche Scenes verwendet werden. Folglich wird für jede „Ansicht" eine eigene Scene erstellt. Die Engine besitzt zu jedem Zeitpunkt eine Scene, welche die aktuelle Benutzeroberfläche definiert. Damit die Navigation unter den verschiedenen Scenes einwandfrei funktioniert wurde die SceneManager-Klasse eingeführt. Diese benachrichtigt über ein Callback-Interface die Activity bzw. Engine, sobald diese eine neue Scene anzeigen soll.

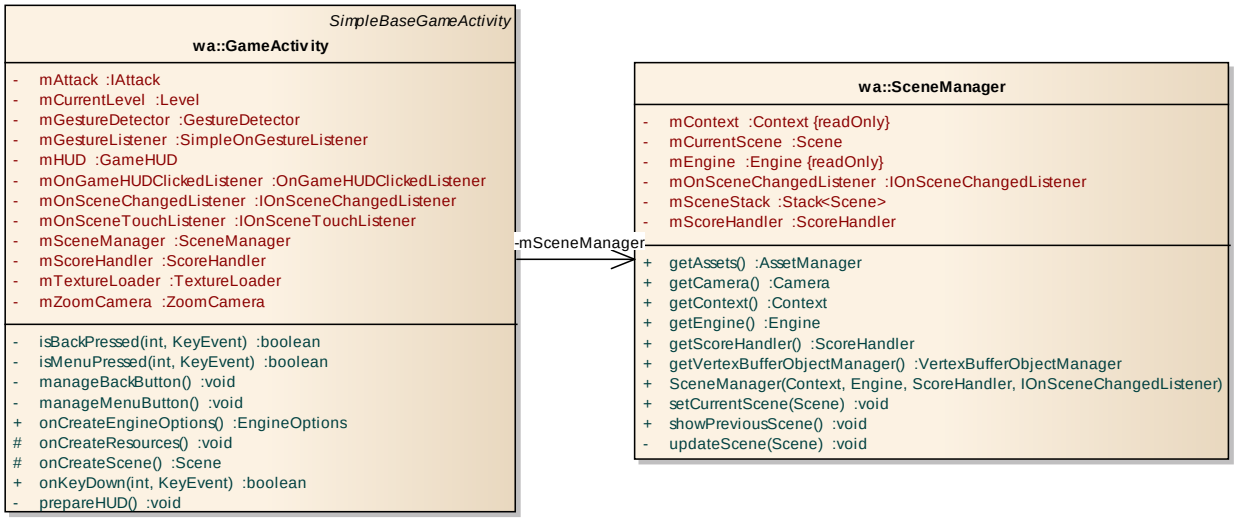


Abbildung 29: GameActivity / SceneManager

Alle Scenes können über den SceneManager die nächste Scene anzeigen lassen. Die erste Scene ist die SplashScene (siehe Code-Snippet 9: onCreateScene Methode).

```
@Override
protected Scene onCreateScene() {
	return new SplashScene(mSceneManager, mTextureLoader,
			getVertexBufferObjectManager());
}
```

Code-Snippet 9: onCreateScene Methode

Die SplashScene lädt alle benötigten Ressourcen und zeigt, sobald diese geladen wurden, das Hauptmenü (MainMenuScene) an.

8.6.3 SceneLoaderTask

Da beim Laden einer neuen Scene zum Teil zeitintensivere Aufgaben, wie zum Beispiel Datenbank-Zugriffe, ausgeführt werden müssen, können die Scenes mithilfe des SceneLoaderTasks asynchron geladen werden. Hierfür wird das Konzept der AsyncTasks verwendet. AsyncTasks ist ein Konzept der Android Library, welches im Kapitel AsyncTasks auf Seite 58 näher erläutert wird.

Damit der SceneLoaderTask verwendet werden kann, muss die zu ladende Scene von der LoadableScene erben und die load-Methode entsprechend überschreiben.

```
public class LoadableScene extends Scene {
	private boolean mLoaded = false;

	public void load() {
		mLoaded = true;
	}

	public boolean isLoaded() {
		return mLoaded;
	}
}
```

Code-Snippet 10: LoadableScene

Der SceneLoaderTask führt dann im Hintergrund-Thread die load-Methode aus und zeigt anschliessend die neue Scene an.

```
public class SceneLoaderTask extends AsyncTask<String, String,
	AsyncTaskResult<Scene>> {

	private final GameScene mGameScene;
	private final SceneManager mSceneManager;

	public SceneLoaderTask(GameScene pGameScene, SceneManager pSceneManager) {
		this.mGameScene = pGameScene;
		this.mSceneManager = pSceneManager;
	}
	@Override
	protected AsyncTaskResult<Scene> doInBackground(String... params) {
		mGameScene.load();
		return new AsyncTaskResult<Scene>();
	}
	@Override
	protected void onPostExecute(AsyncTaskResult<Scene> result) {
		mSceneManager.setCurrentScene(mGameScene);
	}
}
```

Code-Snippet 11: SceneLoaderTask

Eine solche LoadableScene wird folglich nicht mehr direkt über den SceneManager gesetzt, sondern durch das Ausführen eines SceneLoaderTasks. Folgendes Code-Snippet zeigt exemplarisch, wie eine Scene asynchron geladen werden kann.

```
new SceneLoaderTask(mNextScene, mSceneManager).execute();
```

Code-Snippet 12: SceneLoaderTask ausführen

Da die Applikation nur aus einer Activity besteht, würde ein Klick auf den Back-Button zur Folge haben, dass die Applikation vom System geschlossen wird. Damit der Spieler den Back-Button wie gewohnt verwenden kann, muss sich der SceneManager merken, welche Scenes nacheinander angezeigt wurden. Zusätzlich muss die Default-Implementation des Back-Buttons auf der Activity überschrieben werden.

Neben der Default-Implementation des Back-Buttons wird auch die Implementation des Menü-Buttons überschrieben. Der Menü-Button wird im Game verwendet, um die Toolbar ein- und auszublenden.

```
@Override
public boolean onKeyDown(int pKeyCode, KeyEvent pEvent) {
	if (isBackPressed(pKeyCode, pEvent)) {
		manageBackButton();
		return true;
	} else if (isMenuPressed(pKeyCode, pEvent)) {
		manageMenuButton();
		return true;
	} else {
		return super.onKeyDown(pKeyCode, pEvent);
	}
}
```

Code-Snippet 13: onKeyDown Methode

Die Chronik über die angezeigten Scenes wird auf dem SceneManager mit einem Stack implementiert.

Wird über die setCurrentScene-Methode eine neue Scene gesetzt, wird die alte auf den Stack gelegt. Ruft man anschliessend die showPreviousScene-Methode auf, wird die zuletzt angezeigte Scene vom Stack geholt und angezeigt.

```
public void setCurrentScene(Scene pScene) {
	if (pScene != mCurrentScene) {
		mSceneStack.push(mCurrentScene);
		updateScene(pScene);
	}
}

private void updateScene(Scene pScene) {
	mCurrentScene = pScene;
	mOnSceneChangedListener.onSceneChanged(mCurrentScene);
}

public void showPreviousScene() {
	if (mSceneStack.peek() != null)
		updateScene(mSceneStack.pop());
}
```

Code-Snippet 14: SceneManager Stack

Der zuvor beschriebene Ablauf ist in Abbildung 30: Sequenzdiagramm SceneLoaderTask grafisch illustriert.

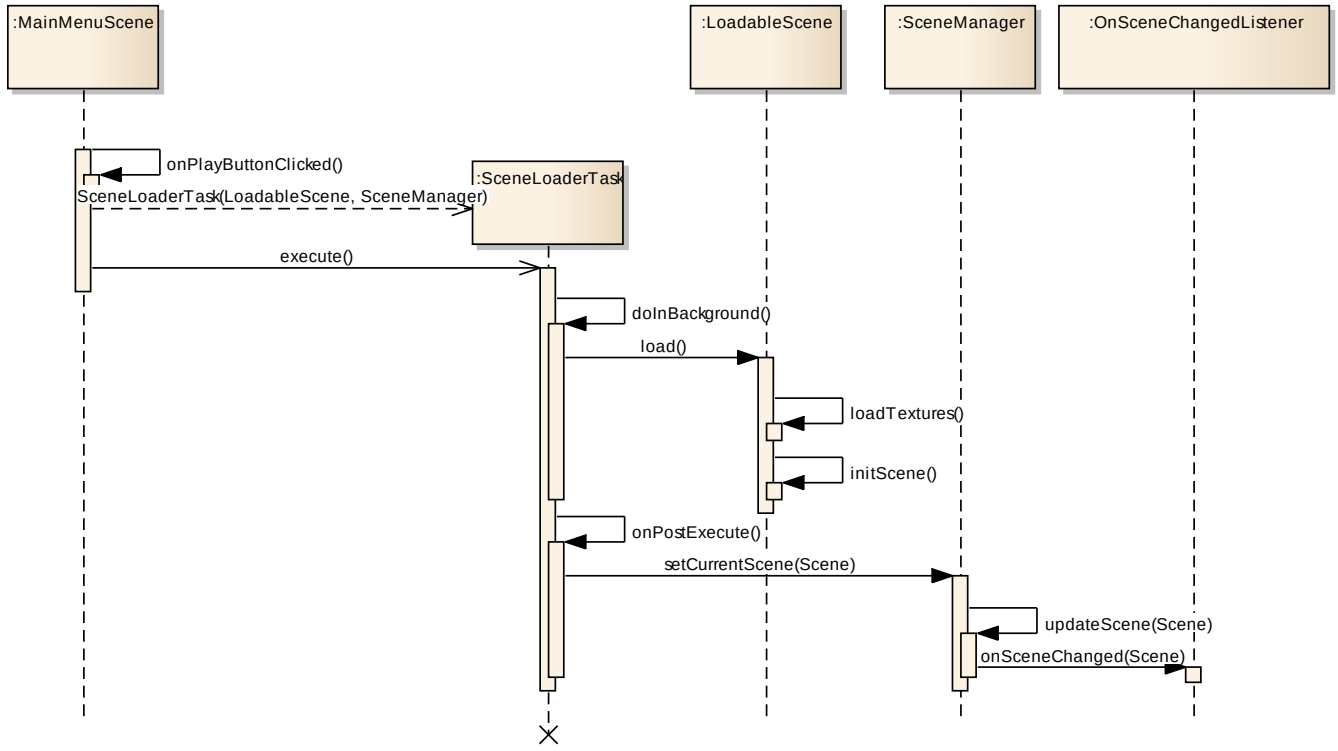


Abbildung 30: Sequenzdiagramm SceneLoaderTask

8.6.4 TextureLoader

Im Game werden unterschiedlichste Texturen von verschiedenen Szenen und anderen GUI-Elementen, zum Teil mehrmals, benötigt. Damit eine Textur aus Versehen nicht mehrfach geladen wird und der ganze Prozess ohnehin etwas umständlich ist, wurde der TextureLoader eingeführt. Der TextureLoader übernimmt das Laden der Textur. Es muss lediglich der Pfad zur gewünschten Textur angegeben werden. Der TexturLoader speichert alle geladenen Texturen in einer Map und sorgt dafür, dass diese auch bei mehrfacher Verwendung nur einmal geladen werden.

Der TextureLoader stellt zum Laden von Texturen unterschiedliche Methoden zur Verfügung. Im häufigsten Fall wird eine Textur benötigt, welche aus nur einer TextureRegion besteht. Diese kann über die loadTexture()-Methode geladen werden.

```
public ITextureRegion loadTexture(String pPath) {
	ITextureRegion texture;
	if (mLoadedTextures.containsKey(pPath)) {
		texture = mLoadedTextures.get(pPath);
	} else {
		texture = createTexture(pPath);
		mLoadedTextures.put(pPath, texture);
	}
	return texture;
}
```

Code-Snippet 15: TextureLoader

In gewissen Fällen müssen mehrere TextureRegions mithilfe von nur einer Textur geladen werden. Dies ist zum Beispiel der Fall, wenn ein Sprite seine Textur ändern kann (Sound-Button: on/off). Für diesen Zweck enthält der TextureLoader die loadMultipleTextures()-Methode, welche ein Array von Textur-Pfaden entgegen nimmt und die unterschiedlichen Bitmap-Dateien in eine Textur lädt.

Damit Sprites animiert dargestellt werden können, werden sogenannte TiledTextureRegions benötigt (siehe Grundlagen). Der TextureLoader kann diese TiledTextureRegions über die loadTiledTexture()-Methode laden.

Da eine Schriftart (Font) ebenfalls in Form einer Textur geladen wird, wird diese ebenfalls vom TextureLoader geladen und verwaltet. Damit alle Texte einheitlich erscheinen, wird jedoch eine vordefinierte Schriftart geladen, welche überall verwendet werden kann.

8.6.5 XML Dateien

An Verschiedensten Stellen wurden XML-Dateien um Konfigurationen und andere Informationen auszulagern. Im folgenden Kapitel werden die unterschiedlichen XML-Dateien vorgestellt und deren Aufbau kurz erläutert.

8.6.5.1 Kategorien

Die unterschiedlichen Level-Kategorien können in der Datei categories.xml definiert werden. Diese befindet sich unter folgendem Pfad /assets/levels/. Die Kategorien sind dem Spielverlauf entsprechend geordnet. Zuerst wird die Farm Kategorie freigeschaltet, darauf folgen City und zuletzt Space.

Das categories.xml enthält folgende Tags:

- **Categories**
  Enthält alle Kategorien in chronologischer Folge.

- **Category**
  Liefert die Informationen für eine Level-Kategorie. Das Vorschaubild der Kategorie (im Menü) wird per Namenskonvention aus dem Ressourcen-Ordner geladen.

Attribute	Werte / Beschreibung
name	Kategoriename
levels	Pfad zu Levels der entsprechenden Kategorie
resources	Pfad zu Kategorie spezifischen Grafiken

Tabelle 7: Category Tag

```
<?xml version="1.0" encoding="utf-8"?>
<categories>

    <category
        name="Farm"
        levels="levels/farm/"
        resources="gfx/farm/" />

    <category
        name="City"
        levels="levels/city/"
        resources="gfx/city/" />

    <category
        name="Space"
        levels="levels/space/"
        resources="gfx/space/" />

</categories>
```

Code-Snippet 16: categories.xml

8.6.5.2 Levels

Pro Level wird ebenfalls ein XML definiert, welches Informationen über die Levelgrösse, den Hintergrund, statische Level-Elemente sowie über die Position des Wurms und die verschiedenen Hilfsmittel, die dem Spieler zur Verfügung stehen, enthält.
Eine Level-Definition enthält folgende Tags:

- **Level**
  Beschreibt den Level-Aufbau und die enthaltenen physikalischen Gegenstände. Zudem enthält es eine Toolbar-Definition.

Attribute	Werte / Beschreibung
background	Pfad zum Hintergrundbild
resources	Pfad zu Level spezifischen Grafiken
height	Höhe in Pixel
width	Länge in Pixel

Tabelle 8: Level-Tag

- **PhysicEntity**
  Beschreibt einen Physikalischen Gegenstand auf dem Level.

Attribute	Werte / Beschreibung
bodyShape	circle, box oder complex
bodyDescription	Pfad zu Body-Definition (XML), benötigt falls bodyShape complex
bodyType	Definiert physikalisches Verhalten des Bodys (StaticBody, KinematicBody, DynamicBody).
texture	Pfad relativ zu Kategorie-Resourcen (resources)
userData	Optional; Wird verwendet um bei der Kollisionserkennung die unterschiedlichen Körper zu identifizieren. (z.B. world oder worm)
x	Ursprungsposition (X-Koordinate)
y	Ursprungsposition (Y-Koordinate)

Tabelle 9: PhysicEntity-Tag

- **Toolbar**
  Enthält Tool-Definitionen (<tool>).
- **Tool**
  Beschreibt ein Hilfsmittel, das dem Spieler zur Verfügung steht.

Attribute	Werte / Beschreibung
bodyShape	circle, box oder complex
bodyDescription	Pfad zu Body-Definition (XML), benötigt falls bodyShape complex
count	Anzahl dieses Hilfsmittel, die dem Spieler zur Verfügung steht. Dieses Attribut ist optional und wird standardmässig auf 1 gesetzt.
preview	Vorschaubild in der Toolbar
texture	Textur des Hilfmittels

Tabelle 10: Tool-Tag

Anschliessend ist eine XML-Definition eines einfachen Levels zu finden. Es enthält zwei verschiedene physicEntities. Einerseits den Boden und andererseits den Wurm. Ausserdem stehen dem Spieler drei Hilfsmittel (Tools), in Form von zwei Eimern und einer Holzlatte, zur Verfügung.

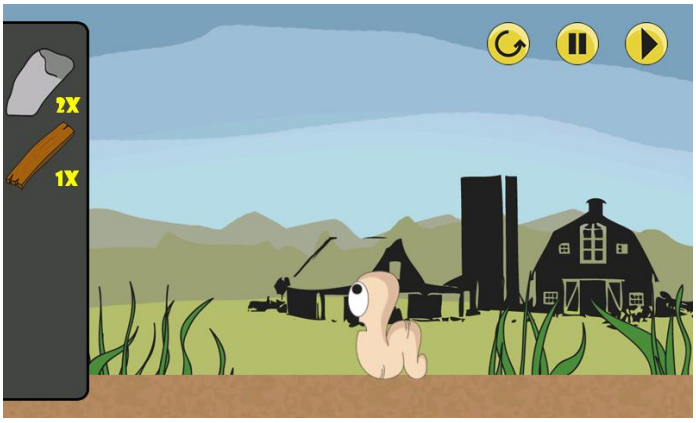


Abbildung 31: Level Beispiel

```xml
<?xml version="1.0" encoding="utf-8"?>
<level
    background="bg.png"
    height="480"
    resources="gfx/farm/"
    width="800" >
    <physicEntity
        bodyShape="box"
        bodyType="StaticBody"
        texture="box_800x50.png"
        userData="world"
        x="0"
        y="430" />
    <physicEntity
        bodyDescription="wilson.xml"
        bodyShape="complex"
        bodyType="DynamicBody"
        texture="worm.png"
        userData="worm"
        x="400"
        y="200" />
    <toolbar>
        <tool
            bodyShape="box"
            count="1"
            preview="box_preview.png"
            texture="box_long.png" />
        <tool
            bodyDescription="bucket.xml"
            bodyShape="complex"
            count="2"
            preview="bucket_preview.png"
            texture="bucket.png" />
    </toolbar>
</level>
```

Code-Snippet 17: Level XML

8.6.5.3 Body Definitionen

Falls das bodyShape-Attribut einer Physic-Entity oder eines Tools den Wert „complex“ besitzt, muss eine entsprechende Body-Definition im Ordner /assets/bodies/ vorhanden sein. Wie diese XML-Files erstellt werden können, wird im Kapitel Kollisionsmodell auf Seite 74 diskutiert. Ein Beispiel ist im Code-Snippet 18: Body-Definition ersichtlich.

Eine Body-Definition enthält folgende Tags:

- **ComplexBody**
  Der Complex-Body Tag eröffnet und terminiert die Body-Definition. Er umfasst alle im Body enthaltenen Polygone (<polygon>).

- **Polygon**
  Ein Polygon beinhaltet seine Vektordaten (<vertex>), welche seine Eckpunkte beschreiben.

- **Vertex**
  Ein Vertex-Tag definiert einen Eckpunkt innerhalb eines Polygons. Ein Polygon kann aus maximal 8 Eckpunkten bestehen.

Attribute	Werte / Beschreibung
x	X-Koordinate des Eckpunktes
y	Y-Koordinate des Eckpunktes

Tabelle 11: Vertex-Tag

Ein Beispiel ist im Code-Snippet 18: Body-Definition ersichtlich.

```
<?xml version="1.0" encoding="UTF-8"?>
<complexbody>
    <polygon>
        <vertex
            x="0.27322298"
            y="0.40649787" />
        <vertex
            x="99.726776"
            y="0.54200023" />
        <vertex
            x="87.431694"
            y="99.729" />
        <vertex
            x="11.338799"
            y="99.45799" />
    </polygon>
</complexbody>
```

Code-Snippet 18: Body-Definition

8.6.6 RessourceLoaderTask

Beim Starten des Games bietet es sich an, die später benötigten Ressourcen frühzeitig zu laden. Dadurch kann der Spieler ohne Wartezeiten durch das Menü navigieren, da die benötigten Ressourcen bereits geladen wurden.

Der RessourceLoaderTask wird zu diesem Zweck auf dem Splash-Screen gestartet. Sobald alle benötigten Ressourcen geladen wurden, wird automatisch ins Hauptmenü navigiert. Der RessourceLoaderTask verwendet dazu einen AsyncTask[12].

Die Methode doInBackground lädt die Level-Kategorien und die dazugehörigen Levels gemäss den Informationen in der Konfigurationsdatei categories.xml. Sofern das Game zum ersten Mal gestartet wird, wird ebenfalls die Datenbank mit den Scoring-Informationen initialsiert.

```
@Override
protected AsyncTaskResult<String> doInBackground(String... params) {
	try {
		List<Category> categoryList = loadCategories();
		loadLevels(categoryList);
		initScoring(categoryList);
		mSceneManager.setCategoryList(categoryList);
	} catch (IOException e) {
		return new AsyncTaskResult<String>(e);
	}
	return new AsyncTaskResult<String>();
}
private List<Category> loadCategories() throws IOException {
	CategoryLoader cl = new CategoryLoader(mTextureLoader, mSceneManager);
	return cl.loadCategories();
}
```

Code-Snippet 19: RessourceLoaderTask: doInBackground

Falls beim Laden der Ressourcen eine Exception auftritt, kann diese über das AsyncTaskResult Objekt an die onPostExecute-Methode übergeben werden. Dort kann der Exception entsprechend eine Fehlermeldung angezeigt werden. Falls keine Exception auftritt, wird die MainMenuScene angezeigt und so in das Hauptmenü navigiert.

```
@Override
protected void onPostExecute(AsyncTaskResult<String> result) {
	if (!result.exceptionOccured()) {
		new SceneLoaderTask(mNextScene, mSceneManager).execute();
	} else {
		showError(result.getException());
	}
}
```

Code-Snippet 20: RessourceLoaderTask onPostExecute

[12] Kapitel 8.3.1 AsyncTasks

8.6.7 Collision Detection

Wie im Designentscheid Angriffswelle (Collision Detection) festgelegt, wurde die Box2D Collision Detection verwendet. Bei der Box2D Collision Detection reagiert der gesetzte ContactListener bei jeder Kollision. Ihm wird dabei ein Contact Objekt übergeben, über welches man die beiden involvierten Bodys erhält.

Für die Collision Detection in Wilson's Adventures wurden drei Arten von Bodys definiert, die in einem Level vorkommen können und somit voneinander unterschieden werden müssen. Dabei handelt es sich um folgende drei Arten:

- Wilson
- Level (Boden, Wände, Tools, ..)
- Angriffselemente

Erst wenn bei einer Kollision zwischen diesen drei Objekten unterschieden werden kann, ist folgender, für den Spielverlauf notwendiger, Entscheid möglich.

- Wenn ein Angriffselement mit Objekten des Levels kollidiert, kann das Angriffselement durch einen passenden Effekt (z.B. Explosion) verschwinden. Dies bedeutet für den Spielverlauf, dass immer noch die Möglichkeit besteht, das Level zu bestehen.
- Sobald jedoch ein Angriffselement mit Wilson kollidiert, hat dies eine Auswirkung auf den weiteren Spielverlauf. Das Level gilt somit als nicht bestanden.

Eine bequeme Möglichkeit um die ähnlichen, jedoch von der Game-Logik her unterschiedlichen Elemente zu unterscheiden, stellen die UserData dar. Diese können jeder Entity in Form eines beliebigen Objects mitgegeben werden. Für die Collision Detection wurden Strings verwendet, um die Bodys unterscheidbar machen. Code-Snippet 21: UserData zeigt die Zuweisung der UserData bei bei der Erstellung eines Angriffselements.

```
public static final String USER_DATA_ATTACK = "attack";
public static final FixtureDef FIXTURE_DEF = PhysicsFactory.createFixtureDef(1,
      0.1f, 0.5f);

Body body = PhysicsFactory.createCircleBody(mPhysicsWorld, sprite,
      BodyType.DynamicBody, FIXTURE_DEF);
body.setUserData(GameOptions.USER_DATA_ATTACK);
```

Code-Snippet 21: UserData

Nachdem der Body erstellt wurde, werden ihm die UserData gesetzt. Ab jetzt kann dieser Body bei einer Kollision eindeutig der Kategorie Angriffselemente zugewiesen werden.

In Code-Snippet 22: Collision Detection wird ein Teil des verwendeten ContactListeners dargestellt. Der Ausschnitt zeigt die Collision Detection zwischen einem Angriffsobjekt und Wilson. Da man nicht weiss, welchen Body man über Fixture A und B bekommt muss die Überprüfung für beide Fälle gemacht werden.

```
private ContactListener mContactListener = new ContactListener() {

	@Override
	public void beginContact(Contact contact) {
		Body bodyA = contact.getFixtureA().getBody();
		Body bodyB = contact.getFixtureB().getBody();

		String userDataBodyA = (String) bodyA.getUserData();
		String userDataBodyB = (String) bodyB.getUserData();

		if (userDataBodyA != null && userDataBodyB != null) {
			if (userDataBodyA.equals(GameOptions.USER_DATA_ATTACK) &&
				userDataBodyB.equals(GameOptions.USER_DATA_WORM)) {

				mAttack.handleAttackingItem(bodyA, contact);
				killWilson(bodyB);

			}

			if (userDataBodyA.equals(GameOptions.USER_DATA_WORM) &&
				userDataBodyB.equals(GameOptions.USER_DATA_ATTACK)) {

				mAttack.handleAttackingItem(bodyB, contact);
				killWilson(bodyA);
			}
		}
	}
};
```

Code-Snippet 22: Collision Detection

8.6.8 Kollisionsmodell

Ein gutes Kollisionsmodell ist für ein Game mit physikalischen Aspekten sehr wichtig. Da man bei Wilson's Adventures per Drag and Drop Gegenstände auf das jeweilige Level bringen kann, können daraus unterschiedlichste Konstruktionen entstehen. Dabei ist es nicht unwahrscheinlich, dass eine instabile Konstruktion während des Baus oder während des Angriffs auf Wilson fällt. Manchmal kann es sogar vorkommen, dass Wilson bewusst ein Teil der Konstruktion wird.
Um bei einem Game dieser Art den Spielspass hoch zu halten ist es wichtig, dass das Kollisionsmodell der einzelnen Gegenstände, inklusive Wilson, sehr genau ist.

Aufgrund des Designentscheids Kollisionsmodell (Seite 93) wurde eine Variante implementiert, welche das Kollisionsmodell mithilfe einer Komposition aus polygenen Shapes erzeugt. In den folgenden Unterkapiteln wird die Erstellung eines komplexen Körpers dargestellt.

8.6.8.1 Physics Body Editor

Der Physics Body Editor[13] unterstützt die Erstellung von komplexen Kollisionsmodellen für unterschiedlichste Physik-Engines. In der übersichtlichen Benutzeroberfläche, kann per Mausklicks ein passendes Kollisionsmodell für importierte Grafiken erstellt werden.

Nachdem eine gewünschte Grafik erstellt und in den Physics Body Editor importiert wurde, kann das Kollisionsmodell beliebig genau durch das Einfügen von Eckpunkten definiert werden.

Abbildung 32: Physics Body Editor Logo

Da der Physics Body Editor in seiner ersten Version speziell für die Erstellung von Box2D Bodys entwickelt wurde, werden alle spezifischen Einschränkungen von Box-2D eingehalten. Die Form wird dadurch automatisch in konvexe Polygone unterteilt, welche maximal acht Eckpunkte besitzen.

In Abbildung 33: Physics Body Editor ist das, durch Punkte definierte, Kollisionsmodell von Wilson ersichtlich. Ebenfalls sieht man die Aufteilung in die einzelnen Polygone.

[13] Entwickelt von Aurelien Ribon (http://code.google.com/p/box2d-editor/)

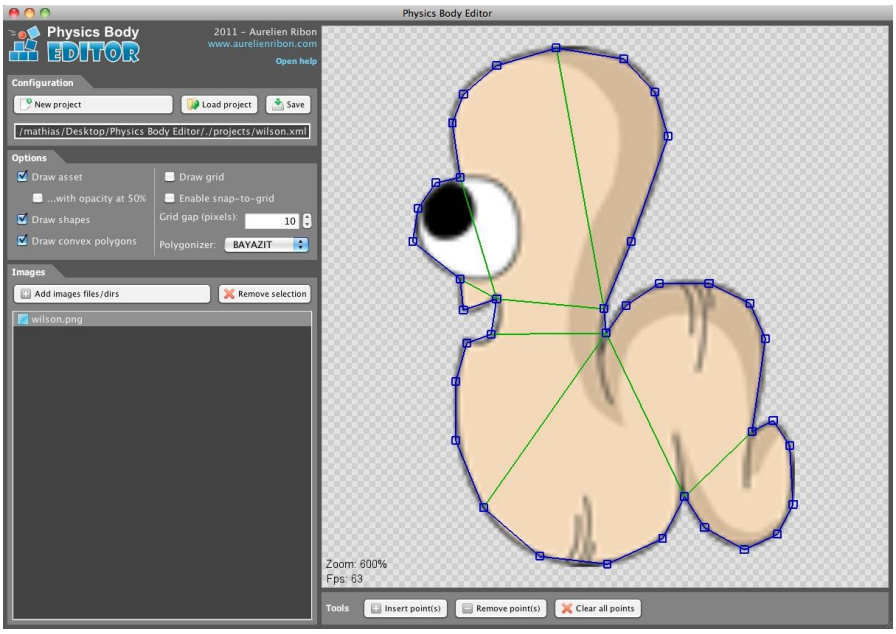


Abbildung 33: Physics Body Editor

Das Kollisionsmodell lässt sich im XML Format exportieren. Darin sind die Polygone mithilfe von Vektoren beschrieben.

Die Definition eines Polygons ist in Code-Snippet 23: Polygon ersichtlich. Ein komplettes Beispiel für ein Body-XML-File ist im Kapitel Body Definitionen aufgelistet.

```
<polygon>
    <vertex x="85.5" y="60.777786"/>
    <vertex x="85.83333" y="68.77778"/>
    <vertex x="76.166664" y="81.44445"/>
</polygon>
```

Code-Snippet 23: Polygon

8.6.8.2 Body-Erstellung

Damit beim Parsing der Level-Definitionen komplexe Körper unterschieden werden können, wurde in der Physic-Entity das Attribut bodyShape eingeführt. Wird dieses Attribut auf den Wert „complex“ gesetzt, muss über ein zweites Attribut (bodyDescription) der Pfad zur XML-Definition des komplexen Körpers angegeben werden. Das bodyDescription-Attribut wird für einfache Shapes (Rechtecke, Kreise usw.) nicht benötigt.

Im folgenden Ausschnitt ist die Definition eines einfachen Körpers (links) sowie eines komplexen Körpers (rechts) zu sehen.

```
<physicEntity
    bodyShape="box"
    bodyType="StaticBody"
    texture="box_800x50.png"
    userData="world"
    x="0"
    y="430" />
```

```
<physicEntity
    bodyDescription="wilson.xml"
    bodyShape="complex"
    bodyType="DynamicBody"
    texture="worm.png"
    userData="worm"
    x="400"
    y="200" />
```

Code-Snippet 24: Physic-Entities

Sobald eine Physic-Entity gelesen wurde, wird mittels der BodyFactory ein passender Body erstellt.

Die BodyFactory Klasse übernimmt das Erstellen der unterschiedlichen Physik-Körper. Dazu gehören die herkömmlichen rechteckigen und kreisförmigen Bodys sowie die komplexen Bodys, welche zuvor mit dem Physics Body Editor im XML Format definiert wurden.

8.6.8.3 BodyFactory

Code-Snippet 25: createBody zeigt die createBody-Methode der Klasse BodyFactory. Zur Unterscheidung der drei Arten von Shapes wird der Parameter pBodyShape verwendet, der aus dem XML-File gelesen wurde.
Für Rechtecke und Kreise wird auf die PhysicsFactory von Box2D zugegriffen. Dabei wird der passende Körper erstellt und zurückgegeben.

Falls es sich um einen komplexen Körper handelt, wird zusätzlich der Parameter pBodyDescription benötigt, dieser enthält den Pfad zum passenden Body-XML-File.

```
public static Body createBody(Context pContext, PhysicsWorld pPhysicsWorld, Sprite
pSprite, BodyShape pBodyShape, BodyType pBodyType,  String pBodyDescription,
FixtureDef pFixtureDef) {
	switch (pBodyShape) {

	case box:
		return PhysicsFactory.createBoxBody(pPhysicsWorld, pSprite,
			pBodyType, pFixtureDef);

	case circle:
		return PhysicsFactory.createCircleBody(pPhysicsWorld, pSprite,
			pBodyType, pFixtureDef);

	case complex:
		return createComplexBody(pPhysicsWorld, pSprite,
			pBodyType,pFixtureDef, pBodyDescription, pContext);
	}
	return null;
}
```

Code-Snippet 25: createBody

Im Falle eines komplexen Bodys wird über die createComplexBody-Methode ein zusammengesetzter Body erstellt. Um einen zusammengesetzten Body zu erhalten, müssen folgende Schritte abgehandelt werden:

1. Die Vektoren aller definierten Polygone müssen eingelesen werden.
2. Die Vektoren müssen skaliert werden.
3. Es muss eine Body-Hülle erstellt werden.
4. Der Body-Hülle werden die einzelnen Polygone hinzugefügt.

```
private static Body createComplexBody(PhysicsWorld pPhysicsWorld, Sprite pSprite,
BodyType pBodyType, FixtureDef pFixtureDef,  String pBodyDescription, Context
pContext) {

	BodyLoader bodyLoader = new BodyLoader(pContext);
	List<Vector2[]> polygons = bodyLoader.getPolygonList(pBodyDescription);
	scaleVertices(polygons, pSprite.getWidth(), pSprite.getHeight());
	 final Body body = createBody((int) pSprite.getX(), (int) pSprite.getY(),
			pBodyType, pPhysicsWorld);
	composeBody(polygons, pFixtureDef, body);
	return body;
}
```

Code-Snippet 26: createComplexBody

Nachdem die Eckpunkte der Polygone über den BodyLoader eingelesen wurden, werden die Vektoren mittels der scaleVertices-Methode auf die tatsächliche Bildgrösse skaliert.

Der Grund für die Skalierung ist auf das generierte XML-File des Physics Body Editors zurückzuführen. Die Vektoren der gesetzten Punkte werden in einer normalisierten Form abgespeichert. Dies bedeutet, dass die Länge und Breite der Grafik auf ein Quadrat (100x100 Pixel) abgebildet wird.

Aus diesem Grund wird der scaleVertices-Methode die Länge und Breite des Sprites mitgegeben, welches später über die generierten Bodys gelegt wird. Nachdem die Ausmasse des Sprites durch die Pixel-Meter-Umrechnungskonstante geteilt wurde, kann jede X und Y-Koordinate aller Vektoren auf die tatsächliche Länge skaliert werden. So wird sichergestellt, dass der zusammengesetzte Body genau die Grösse des Sprites hat.

```
private static void scaleVertices(List<Vector2[]> pPolygons, float pWidth, float
pHeight) {
	final float width = pWidth / GameOptions.PIXEL_TO_METER_RATIO_DEFAULT;
	final float height = pHeight / GameOptions.PIXEL_TO_METER_RATIO_DEFAULT;

	for (Vector2[] v : pPolygons) {
		for (int i = 0; i < v.length; i++) {
			v[i].x = (v[i].x / 100) * width;
			v[i].y = (v[i].y / 100) * height;
		}
	}
}
```

Code-Snippet 27: scaleVertices

Mit den skalierten Vektoren sind nun die Voraussetzungen geschaffen, um die Bodys zu erstellen. Da der komplexe Körper aus mehreren Bodys besteht, wird zuerst über die private createBody-Methode eine Body-Hülle erstellt. Dies wird erreicht, indem zuerst eine BodyDef erzeugt wird. Dieser kann der BodyType sowie die Position mitgegeben werden.

```
private static Body createBody(int pX, int pY, BodyType pBodyType,
	PhysicsWorld pPhysicsWorld) {

	final BodyDef bodyDef = new BodyDef();
	bodyDef.type = pBodyType;
	bodyDef.position.x = pX / GameOptions.PIXEL_TO_METER_RATIO_DEFAULT;
	bodyDef.position.y = pY / GameOptions.PIXEL_TO_METER_RATIO_DEFAULT;
	final Body body = pPhysicsWorld.createBody(bodyDef);
	return body;
}
```

Code-Snippet 28: createBody

Innerhalb der composeBody-Methode werden nun die einzelnen Polygone wieder zusammengesetzt. Dazu wird jeweils ein neues PolygonShape erstellt, auf welchem anschliessend mit der set-Methode die Eckpunkte definiert werden. Das neu erstellte PolygonShape wird nun dem shape-Attribut auf der FixtureDef gesetzt. Nun kann dem Body mithilfe der createFixture-Methode die neue FixtureDef mitgegeben werden, welche nun um das neue Polygon ergänzt ist. Das erstellte PolygonShape wird nun nicht mehr benötigt und kann mittels dispose() für den Garbage Collector freigegeben werden.

```
private static void composeBody(List<Vector2[]> pPolygons, FixtureDef pFixtureDef,
final Body body) {

	for (Vector2[] vertices : pPolygons) {
		final PolygonShape polygonShape = new PolygonShape();
		polygonShape.set(vertices);
		pFixtureDef.shape = polygonShape;
		body.createFixture(pFixtureDef);
		polygonShape.dispose();
	}
}
```

Code-Snippet 29: composeBody

8.6.9 Scoring

Das Scoring-Modell in Wilson's Adventures unterteilt sich in zwei Gebiete. Erstens muss das Freischalten der Levels und Kategorien geregelt sein. Zweitens muss pro Level ein Highscore gespeichert werden können. Zu diesem Zeitpunkt steht jedoch noch nicht fest, wie diese Highscores berechnet werden.

Durch den definierten Aufbau des Games in Level-Kategorien und einzelne Levels ergab sich folgende Freischalte-Logik:

- Ist ein Level bestanden, kann das darauffolgende Level gespielt werden
- Sind alle Levels einer Kategorie bestanden, kann in der nächsten Kategorie weitergespielt werden

Um dieses Scoring-Model zu realisieren, mussten pro Level und Kategorie folgende Informationen gespeichert werden:

- Ist das Objekt bereits spielbar (unlocked)?
- Ist das Objekt bereits erfolgreich bestanden (passed)?
- Wie hoch ist der aktuelle Highscore des Objekts?

8.6.9.1 SQLite

Da die Informationen des Scorings persistent gespeichert werden müssen, wurde eine SQLite Datenbank eingesetzt. Weil sich die Levels und die Kategorien vom Scoring-Modell her nicht unterscheiden, wurde eine SQLite Datenbank mit nur einer Tabelle erstellt. Diese Tabelle beinhaltet folgende Spalten:

- ID: String
- Unlocked: Boolean
- Passed: Boolean
- Score: Integer

Da die Kategorien und Levels in XML-Files ausgelagert sind, ist der jeweilige Filename eindeutig (z.B. farm_001.xml) und kann somit als Primary Key ID in der Tabelle verwendet werden.
Durch den Aufbau dieser Tabelle blieb die dynamische Definition von Kategorien und Levels erhalten. Das heisst konkret, dass immer neue Levels und Kategorien erstellt werden könnten. Dies lässt die Möglichkeit offen, Updates für das Game zur Verfügung zu stellen, wobei der Spieler neue Levels erhält.
Sobald beim Starten des Games die XML-Files geparsed werden, wird ein neuer Datensatz angelegt. Falls bereits ein Eintrag mit dem geparsten File-Namen besteht, wird dieser in der Datenbank gelassen. Somit ist sichergestellt, dass die bereits freigeschalteten Levels weiterhin freigeschaltet bleiben und dass die erreichten Highscores nicht überschrieben werden.

8.6.9.2 ScoreDBHandler

Die Datenbank Zugriffe werden in der ScoreDBHandler Klasse gekapselt. Der ScoreDBHandler stellt nebst der insert-Methode jeweils Getter und Setter für die Spalten Unlocked, Passed und Score zur Verfügung. Die insert Methode ist in Code-Snippet 30: insert() ersichtlich. Dabei werden die neuen Werte in einem ContentValue Objekt gespeichert, das anschliessend in die Datenbank eingefügt wird.

```
public void insert(String pID, boolean pUnlocked, boolean pPassed, int pScore) {
	SQLiteDatabase dbConn = mScoreDb.getWritableDatabase();

	ContentValues contentValue = addValues(pID, pUnlocked, pPassed, pScore);
	dbConn.insertOrThrow(ScoreTable.TABLE_NAME, null, contentValue);

	dbConn.close();
}
```

Code-Snippet 30: insert()

Die Getter und Setter Methoden werden anhand der Spalte Unlocked dargestellt. Die isUnlocked-Methode benötigt eine private Hilfsmethode (getCursorById), welche ein Cursor auf den gewünschten Datensatz zurückgibt. Anschliessend kann der boolean ausgelesen werden.

```
public boolean isUnlocked(String pID) {
	Cursor c = getCursorById(pID);
	int columnIndex = c.getColumnIndex(ScoreTableColumns.UNLOCKED);
	return c.getInt(columnIndex) == 1;
}
```

Code-Snippet 31: isUnlocked()

Die eben erwähnte Hilfsmethode, die für alle Abfrage Operationen auf der Datenbank benötigt wird, ist unterhalb ersichtlich. Dabei wird eine Query auf die Datenbank abgesetzt, welche einen Cursor zurückgibt, der auf den Datensatz mit der gewünschten Id zeigt.

```
private Cursor getCursorById(String pID) {
	SQLiteDatabase dbConn = mScoreDb.getReadableDatabase();
	Cursor c = dbConn.query(ScoreTable.TABLE_NAME, null, ScoreTable.ID + " =
			?", new String[] { pID }, null, null, null);
	c.moveToFirst();
	dbConn.close();
	return c;
}
```

Code-Snippet 32: getCursorById()

Die setUnlocked-Methode erstellt auch ein ContentValue Objekt, womit Unlocked auf true gesetzt wird. Da bei jedem bestandenen Level das nächste freigeschaltet

wird, besteht kein Anspruch auf das Setzen von unlocked auf false, ausser bei der Initialisierung durch die insert-Methode.

```
public void setUnlocked(String pID) {
	ContentValues contentValues = new ContentValues();
	contentValues.put(ScoreTableColumns.UNLOCKED, true);
	update(pID, contentValues);
}
```

Code-Snippet 33: setUnlocked()

Sobald das neue ContentValue Objekt erstellt wurde, wird es über die private update-Methode in die Datenbank eingefügt. Dies geschieht über ein Update auf der Datenbank. Dazu werden die Id und das ContentValue Objekt benötigt.

```
private void update(String pID, ContentValues pContentValues) {
	SQLiteDatabase dbConn = mScoreDb.getWritableDatabase();
	dbConn.update(ScoreTable.TABLE_NAME, pContentValues, ScoreTableColumns.ID
		+ " = ?", new String[] { pID });
	dbConn.close();
}
```

Code-Snippet 34: update()

8.6.9.3 ScoreHandler

Der ScoreHandler kapselt die Scoring Logik, wobei über den ScoreDBHandler auf die Datenbank zugegriffen wird.
Das Prinzip, dass die XML-File-Namen als Primary Key für Levels und Kategorien gelten setzt voraus, dass Levels und Kategorien Auskunft über ihre ID geben können. Dies ist über die abstrakte Klasse AbstractScore gelöst, welche definiert, dass die erbenden Level- und Kategorie-Klassen die Methode getId() überschreiben müssen. Das Id Feld wird beim Parsing der Levels und Kategorien jeweils im Kategorie- und Level-Konstruktor mitgegeben.
Wenn nun der Spieler in das Kategorie – oder Level-Auswahl Menü navigiert, kann für jedes AbstractScore-Objekt über den ScoreHandler ermittelt werden, ob es Unlocked ist. Je nach Ergebnis der Abfrage wird die Kategorie oder das Level als gesperrt oder als spielbar dargestellt. Ebenfalls kann zum Beispiel anhand der Methoden isPassed() und getScore()der aktuelle Highscore angezeigt werden.

```
public boolean isUnlocked(AbstractScore pScore) {
	return mScoreDbHandler.isUnlocked(pScore.getId());
}

public boolean isPassed(AbstractScore pScore) {
	return mScoreDbHandler.isPassed(pScore.getId());
}

public int getScore(AbstractScore pScore) {
	return mScoreDbHandler.getScore(pScore.getId());
}
```

Code-Snippet 35: isUnlocked(), isPassed(), getScore()

Die Hauptaufgabe des ScoreHandlers besteht darin, nach einem erfolgreichen Abschluss eines Levels zu entscheiden, welche Auswirkungen dies auf das Scoring hat. Die beiden Aufgaben sind, wie zu Beginn des Kapitels erwähnt, das Speichern des Highscores sowie das Freischalten des nächsten Levels.

Das Updaten des Scores wird über die updateScores-Methode realisiert. Dabei wird zuerst der alte Score mit dem neuen verglichen. Wenn der neue Score höher ist als der alte – es wurde ein neuer Highscore erzielt – wird die Datenbank auf den neusten Stand gebracht. Wenn der neue Score niedriger ist als der alte, wird der bisherige Highscore nicht verändert.

Sobald der Highscore auf dem Level angepasst wurde, wird der neue Gesamt-Highscore der jeweiligen Kategorie berechnet.

```
private void updateScores(AbstractScore pScore, int pNewScore) {
	AbstractScore container = pScore.getContainer();
	int oldScore = mScoreDbHandler.getScore(pScore.getId());
	if (pNewScore > oldScore) {
		mScoreDbHandler.setScore(pScore.getId(), pNewScore);

		int oldCategoryScore = mScoreDbHandler.getScore(container.getId());
		int newCategoryScore = (oldCategoryScore - oldScore) + pNewScore;
		mScoreDbHandler.setScore(container.getId(), newCategoryScore);
	}
}
```

Code-Snippet 36: updateScores()

Nachdem die Scores angepasst wurden, wird die Freischalte-Logik über die updateLocking-Methode realisiert.

1. Dabei wird zuerst das bestandene Level auf passed gesetzt.
2. Anschliessend wird überprüft, ob das Level ein Folge-Level hat. Falls dies der Fall ist, kann das Folge-Level auf unlocked gesetzt werden.
3. Wenn das bestandene Level kein Folge-Level besitzt wird überprüft ob die Kategorie, in welchem sich das Level befindet, eine Folge-Kategorie hat.
4. Wenn dies der Fall ist, kann über die unlockCategory-Methode die nächste Kategorie und das erste Level der nächsten Kategorie auf unlocked gesetzt werden. Der Spieler hat somit eine neue Kategorie freigeschaltet.

```
private void updateLocking(AbstractScore pScore) {
	AbstractScore container = pScore.getContainer();

	mScoreDbHandler.setPassed(pScore.getId());

	if (!pScore.hasNext()) {
		if (container.hasNext())
			unlockContainer(container.getNext());

	} else {
		mScoreDbHandler.setUnlocked(pScore.getNext().getId());
	}
}
```

Code-Snippet 37: updateLocking()

8.6.10 Sound

8.6.10.1 Einleitung

Hintergrundmusik und Soundeffekte sind wichtige Bestandteile eines Games. Durch akustische Effekte kann ein Spielerlebnis massgeblich beeinflusst werden. So auch bei Smartphone-Games. Eine Hintergrundmusik, welche den Wiedererkennungswert steigert und Soundeffekte, welche die Faszination des Games steigern, können sich enorm auf das ganze Produkt auswirken, vor allem der Negativeffekt.
Obwohl sich das Projektteam dessen bewusst war, wurde nur ein geringer Aufwand dafür betrieben, um passende Sounds zu erstellen. Der Grund dafür ist, dass die Priorität dieser Arbeit nicht auf der Entwicklung eines marktreifen Apps liegt, sondern darin, möglichst viele Bereiche der Android-Game-Programmierung kennen zu lernen.
Jedoch muss auch bei einem marktreifen App entweder professioneller Sound gekauft werden oder man hat selbst die Möglichkeit, Sounds mit guter Qualität zu erstellen.

Das Ausbleiben von professionellen Sounds sollte es jedoch nicht verunmöglichen, Hintergrundmusik und Soundeffekte an passenden Stellen einzubinden. Die dafür notwendige Implementation der AudioHandler-Klasse wird im Kapitel AudioHandler erklärt. Die verwendeten Musikstücke und Soundeffekte sind dabei als Platzhalter für aufwändig kreierte Musik-Kompositionen zu sehen.

8.6.10.2 AndEngine Sound[14]

AndEngine stellt vier Klassen für den Gebrauch mit Sounds zur Verfügung.

- Die Music-Klasse repräsentiert einen Musik-Stream.
- Die Sound-Klasse repräsentiert einen Sound-Effekt.
- Die MusicFactory übernimmt das Laden einer Musikdatei in ein Music Objekt.
- Die SoundFactory übernimmt das Lader eines Soundeffekts in ein Sound Objekt.

Dabei ist die Music-Klasse für Musikstücke und die Sound-Klasse für Soundeffekte verantwortlich. Als Daumenregel sind Musik-Stücke länger als fünf Sekunden und Soundeffekte bestenfalls um die drei Sekunden oder kürzer.

Die wichtigsten Methoden, welche von diesen beiden Klassen angeboten werden, sind folgende:

- play()
- stop()
- pause()
- resume()
- release()

Da der Music und Sound Klasse der MediaPlayer des Android Systems zu Grunde liegt, bieten sie die gleichen Methoden wie der MediaPlayer an.

[14] [LeAnGaPro]

Um Music und Sounds verwenden zu können, müssen diese in der onCreateEngineOptions-Methode der Klasse GameActivity registriert werden.

```
engineOptions.getAudioOptions().setNeedsMusic(true);
engineOptions.getAudioOptions().setNeedsSound(true);
```

Code-Snippet 38: engineOptions

8.6.10.3 AudioHandler

Die AudioHandler-Klasse übernimmt das Laden von Music- sowie Sound-Objekten und kapselt die Steuerung des Sounds.
Wie bereits erwähnt, findet das Laden von Music- und Soundobjekten über die jeweiligen Factorys statt. Die Music und Sound Factorys sind der Textur Factory sehr ähnlich. Im Codesnippet unterhalb ist die loadMusic-Methode des AudioHandlers ersichtlich. Der einzige Unterschied zur loadSound-Methode ist das Looping-Flag, welches hier auf true gesetzt wird. Dies hat zur Folge, dass die Hintergrundmusik immer dauernd wiederholt wird.

```
public void loadMusic() {
	MusicFactory.setAssetBasePath(GameOptions.MFX_PATH_MUSIC);
	try {
		mMusic= MusicFactory.createMusicFromAsset(mEngine.getMusicManager(),
				mContext, GameOptions.MUSIC_MUSIC);
		mMusic.setLooping(true);
	} catch (IOException e) {
		e.printStackTrace();
	}
}
```

Code-Snippet 39: loadMusic()

Wie in den meisten Games ist es möglich, den Sound ein und ausschalten zu können. Ebenfalls ist man es sich gewohnt, dass nach einem Neustart des Games die zuletzt eingestellten Audio-Optionen berücksichtigt werden.
Um Informationen über die Ausführungsdauer eines Apps hinaus speichern zu können, bietet Android das Konzept der Shared Preferences an. Dabei können Key-Value Paare von primitiven Typen gespeichert werden.
Bei der Initialisierung des AudioHandlers wird in der initSharedPreferences-Methode auf die Shared Preferences zugegriffen. Falls das Game zum Ersten Mal startet, wird der Sound standardmässig aktiviert. Wenn bereits ein Eintrag in den Shared Preferences besteht, wird dieser Wert übernommen.

```
private void initSharedPreferences() {
	mAudioOptions = mContext.getSharedPreferences(
				GameOptions.SHARED_PREFERENCES_NAME, 0);
	mAudioEditor = mAudioOptions.edit();
	if (!mAudioOptions.contains(GameOptions.SHARED_PREFERENCES_KEY)) {
		mAudioEditor.putBoolean(GameOptions.SHARED_PREFERENCES_KEY, true);
		mAudioEditor.commit();
	}
}
```

Code-Snippet 40: initSharedPreferences()

Über die setAudioEnabled und isAudioEnabled-Methoden können die Shared Preferences Editiert und abgefragt werden. Diese Methoden werden vom OnClickListener des SoundButtons benötigt.

In Wilson's Adventures wird, solange sich der Spieler im Menü befindet, eine Hintergrundmusik abgespielt. Im Gegensatz dazu werden vereinzelte Soundeffekte nur zu ganz bestimmten Ereignissen abgespielt. Zum Beispiel wenn Wilson von einem Angriffselement getroffen wird.
Aus diesem Grund wird für die Musik eine playMusic() und stopMusic()-Methode angeboten, die vom OnClickListener des Soundbuttons aufgerufen werden.

```
public void playMusic() {
	if (isAudioEnabled())
		mMusic.play();
}

public void stopMusic() {
	if (mMusic.isPlaying())
		mMusic.pause();
}
```

Code-Snippet 41: playMusic() stopMusic()

Ebenfalls werden diese beiden Methoden in den onPauseGame()- und onResumeGame()-Methoden in der GameActivity-Klasse verwendet. Diese Methoden bieten die Möglichkeit, dass die Hintergrundmusik in Bezug auf den Android Lifecycle korrekt gestartet und gestoppt wird.

```
@Override
public synchronized void onPauseGame() {
	super.onPauseGame();
	if (isPlaying())
		showPauseScreen();
	mAudioHandler.stopMusic();
}

@Override
public synchronized void onResumeGame() {
	super.onResumeGame();
	mAudioHandler.playMusic();
}
```

Code-Snippet 42: onPauseGame() onResumeGame()

Für die Soundeffekte steht nur eine playSound-Methode zur Verfügung, da diese nicht gestoppt werden müssen. Jedoch wird auch bei der playSound-Methode über die Shared Preferences überprüft, ob zurzeit Sound Effekte erwünscht sind oder nicht.

8.6.11 Physic Entity Pool

8.6.11.1 Einleitung

Bei der Entwicklung einer Smartphone Applikation muss auf einen sparsamen Memory-Gebrauch geachtet werden. Dies gilt für Business-Apps sowie für Games. Eine Möglichkeit, um die Memory Auslastung zu reduzieren, stellt ein Pool dar.

8.6.11.2 Das Pool-Prinzip

Die Aufgabe eines Pools ist das ressourcensparende Verwalten von Objekten. Dafür stellt ein Pool folgende zwei Methoden zur Verfügung:

- Objekte über den Pool beziehen
- Objekte an den Pool zurückgeben

Die Interaktion mit dem Pool läuft folgendermassen ab. Wenn vom Benutzer des Pools ein Objekt benötigt wird, kann es über den Pool bezogen werden. Dabei muss der Pool überprüfen, ob er noch Objekte zur Verfügung hat. Falls dies der Fall ist, kann das Objekt dem Benutzer gegeben werden. Wenn kein Objekt vorhanden ist, muss zuerst ein neues erstellt werden.
Sobald der Benutzer des Pools das Objekt nicht mehr braucht, kann er es dem Pool zurückgeben. Der Pool hat somit wieder ein verwendbares Objekt, welches er bei der nächsten Anfrage dem Benutzer übergeben kann.

Durch dieses Recycling-Prinzip werden nicht dauernd neue Objekte erstellt und somit wird im Idealfall auch weniger Memory gebraucht. Der Einsatz des Pools ist jedoch mit Vorsicht zu geniessen. Das ressourcen-schonende Prinzip des Pools funktioniert nämlich nur, wenn auch regelmässig Objekte recycelt werden.

8.6.11.3 Einsatz des Pools bei Wilson's Adventures

Die Levels von Wilson's Adventures basieren, bezüglich dem Game-Konzept, grossteils auf statischen Levels, welche durch diverse Tools ergänzt werden können. Ebenfalls sind zu diesem Zeitpunkt keine Levels ausgearbeitet, in welchen Wilson seine Position durch den Einsatz von Lockstoffen verändert. Somit sind alle diese Objekte während eines Levels vom Anfang bis zum Schluss ersichtlich und können somit auch nicht recycled werden.

Das Starten der Angriffswelle bringt jedoch Angriffselemente ins Spiel, welche sich je nach Level-Kategorie bei einer Kollision auflösen. Für die Angriffselemente bietet sich der Einsatz eines Pools an.

Da es sich bei den Angriffselementen um physikalische Objekte handelt, wird für jedes einzelne Angriffselement ein Sprite mit definierter Textur und ein Physik-Body mit passender Grösse verwendet. Aus diesem Grund muss der Pool mit Objekten arbeiten, welche jeweils aus einem Sprite und einem Physik-Body bestehen. Somit ist garantiert, dass das Sprite sowie der Body eines Angriffselements sauber verwaltet werden können.

Die Klasse PhysicEntity bietet mit folgenden vier Methoden den Gebrauch von Objekten mit Sprites und Bodys an:

- getSprite()
- setSprite()
- getBody()
- setBody()

Die Implementation der daraus entstehenden Klasse PhysicEntityPool ist von der GenericPool-Klasse aus der AndEngine abgeleitet. Dabei werden vom Pool PhyiscEntity-Objekte verwaltet. Durch das Überschreiben der folgenden drei Methoden kann der Pool auf die eigenen Ansprüche angepasst werden:

- onAllocatePoolItem()
- onHandleRecycleItem()
- onHandleObtainItem()

In Code-Snippet 43: onAllocatePoolItem() ist die onAllocatePoolItem-Methode ersichtlich. Diese Methode wird aufgerufen, wenn eine PhysicEntity benötigt wird, jedoch zurzeit keine vorhanden ist. Dabei wird ein Sprite sowie ein Physic-Body erstellt und durch den PhysicsConnector verbunden.

```
@Override
protected PhysicEntity onAllocatePoolItem() {

	Sprite sprite = new Sprite(0, 0, mTextureRegion,
			mVertexBufferObjectManager);

	Body body = PhysicsFactory.createCircleBody(mPhysicsWorld, sprite,
			BodyType.DynamicBody, FIXTURE_DEF);
	body.setUserData(GameOptions.USER_DATA_ATTACK);

	PhysicEntity physicEntity = new PhysicEntity(sprite, body);
	mPhysicsWorld.registerPhysicsConnector(
			new PhysicsConnector(sprite, body, true, true));

	return physicEntity;
}
```

Code-Snippet 43: onAllocatePoolItem()

In Code-Snippet 44: onHandleRecycleItem() sieht man die onHandleRecycleItem-Methode. Diese Methode wird aufgerufen, sobald eine PhysikEntity an den Pool zurückgegeben wird.
Dabei wird dem Sprite mitgeteilt, dass es zurzeit auf keine Updates reagieren muss. Ebenfalls wird es unsichtbar gemacht.
Der Body wird über die setAwake und setActive Methoden zwischenzeitlich deaktiviert.

```
@Override
protected void onHandleRecycleItem(final PhysicEntity physicEntity) {

	Sprite sprite = physicEntity.getSprite();
	sprite.setIgnoreUpdate(true);
	sprite.setVisible(false);

	final Body body = physicEntity.getBody();
	body.setAwake(false);
	body.setActive(false);

	mPhysicsEntityHandler.removePhysicEntity(physicEntity);
}
```

Code-Snippet 44: onHandleRecycleItem()

In der letzten Zeile wird die PhysicsEntity auf dem PhysicsEntityHandler gelöscht. Im PhysicsEntityHandler werden alle, zur Zeit aktiven, PhysicsEntitys gespeichert. Der Grund dafür liegt bei der Box2D Collision Detection, welche im Kapitel 7.6.2 beschrieben ist. Bei einer Kollision kann über das Contact Objekt nur auf den betreffenden Body zugegriffen werden. Da jedoch das dazugehörige Sprite von der Scene gelöscht werden muss, kann über den PhysicsEntityHandler das passende Sprite gefunden werden. Die Implementation des PhysicEntityHandlers ist in Code-Snippet 45: PhysicEntityHandler ersichtlich.

```
public class PhysicEntityHandler {

	private List<PhysicEntity> list = new ArrayList<PhysicEntity>();

	public synchronized void add(PhysicEntity physicEntity) {
		list.add(physicEntity);
	}

	public synchronized PhysicEntity getPhysicEntityBy(Sprite sprite) {
		for (PhysicEntity b : list) {
			if (b.getSprite() == sprite)
				return b;
		}
		return null;
	}

	public synchronized PhysicEntity getPhysicEntityBy(Body body) {
		for (PhysicEntity b : list) {
			if (b.getBody() == body)
				return b;
		}
		return null;
	}

	public synchronized void removePhysicEntity(PhysicEntity physicEntity) {
		list.remove(physicEntity);
	}
}
```

Code-Snippet 45: PhysicEntityHandler

Die letzte der drei Methoden des Pools ist in Code-Snippet 46: onHandleObtainItem() dargestellt. Dabei handelt es sich um die onHandleObtainItem-Methode. In dieser Methode hat man die Möglichkeit, letzte Anpassungen an dem Objekt vorzunehmen, bevor es dem Pool-Benutzer übergeben wird. Dabei wird nicht zwischen neuen und recycelten Objekten unterschieden.
Diese Methode bietet sich gut dafür an, um die in der onHandleRecycleItem-Methode getätigten Änderungen wieder rückgängig zu machen. Somit ist das Objekt wieder aktiviert und für den Einsatz als Angriffselement vorbereitet.

```
@Override
protected void onHandleObtainItem(final PhysicEntity physicEntity) {

	Sprite sprite = physicEntity.getSprite();
	sprite.setIgnoreUpdate(false);
	sprite.setVisible(true);

	Body body = physicEntity.getBody();
	body.setAwake(true);
	body.setActive(true);

	mPhysicsEntityHandler.add(physicEntity);
}
```

Code-Snippet 46: onHandleObtainItem()

9 Designentscheide

9.1 Angriffswelle (Collision Detection)

Im Game-Konzept wurde spezifiziert, dass Wilson einem Angriff von oben ausgesetzt wird. Die gesammelten Ideen zur Angriffswelle sind im Kapitel 5.3.1 zu finden.
Eine Variante, welche dem Projektteam von Beginn an gefallen hat, war ein Giftangriff. Die Idee dahinter war, dass der Bauer durch das Feld läuft und dabei ein Schädlingsgift versprüht. Sobald Wilson vom Gift getroffen wird, gilt das Level als nicht bestanden. Eine Skizze vom Giftangriff ist unterhalb zu sehen.

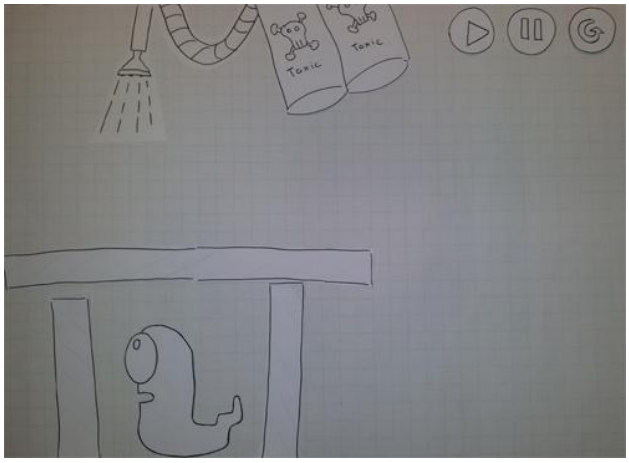


Abbildung 34: Skizze Giftangriff

Im Prototyp wurden die von der Game-Engine angebotenen Möglichkeiten ausprobiert, welche für einen Angriff verwendet werden können. Im Hinblick auf einen Giftangriff musste eine Variante gefunden werden, die es erlaubt Flüssigkeiten zu simulieren.

9.1.1 Partikel Systeme

Während der Realisierung des Prototyps, wurde mithilfe des Partikel Systems von AndEngine ein Giftangriff realisiert. Dazu wurde ein Partikel in Form eines Gifttropfens angefertigt. Das Partikel System wurde an die Giftspritze angehängt. Anschliessend sorgte das Partikel System dafür, dass die Partikel an der richtigen Position erstellt werden und sich korrekt wegen. Ein Screenshot des Prototyps ist in Abbildung 35: Partikel System zu sehen.

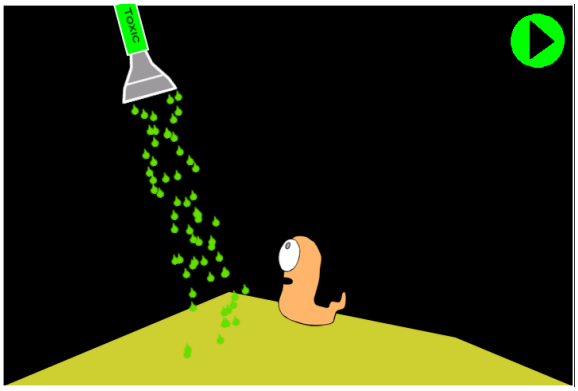


Abbildung 35: Partikel System

9.1.2 Collision Detection (Partikel System)

Wie in Kapitel 7.6 Collision Detection beschrieben, existieren zwei Möglichkeiten um eine Kollision von zwei Objekten zu erkennen.

Den einzelnen Partikeln können unterschiedliche Modifier zugewiesen werden. Anhand dieser Modifier kann eine gravitationsähnliche Bewegung realisiert werden, obwohl die Partikel nicht Teil der Physikwelt sind und somit keine physikalische Repräsentation (Body) besitzen. Da das Partikel System jedoch nur lose Sprites erstellt (ohne Body), kommt eine Collision Detection mithilfe der Physikwelt nicht in Frage.

Aus diesem Grund konnte nur der Ansatz der Shape-Kollision verwendet werden um festzustellen, wann ein Partikel kollidiert. Während der Implementation des Prototyps wurde klar, dass dieser Ansatz nicht zufriedenstellend ist.

Die Shape-Kollision hat einen sehr grossen Nachteil. Das Prinzip der Shape-Kollision beruht auf der Pixelüberlagerung zweier unterschiedlicher Sprites. Enthält jedoch eine der überlagernden Texturen transparente Bereiche (Abbildung 35: Partikel System) funktioniert die Shape-Kollision nicht mehr wie erwartet. Da für die Berechnung der Pixelüberlagerung die gesamte Textur (inklusiv transparenter Bereiche) verwendet wird, werden in der Shape-Kollision folglich alle Sprites als Rechtecke, entsprechend der Texturhöhe und –breite, interpretiert. Bei dem gelben Polygon in Abbildung 35: Partikel System, gilt in der Shape-Kollision das rote Rechteck aus Abbildung 36: Rechteckiges Kollisionsmodell als Kollisionsmodell für den Boden.

Abbildung 36: Rechteckiges Kollisionsmodell

9.1.3 Entscheid

Aufgrund des Nachteils bei der Verwendung eines Partikel Systems als Angriffsvariante wurde eine andere Lösung gesucht.

Die offensichtlichste Lösung ist ein Angriff, welcher aus festen (physikalischen) Elementen besteht. Werden die Angriffselemente in die Physikwelt aufgenommen, kann die Collision Detection der Physik-Engine verwendet werden.
Die neue Version der Angriffswelle wird durch einen Schleuderstreuer dargestellt, welcher am Traktor des Bauers befestigt ist. Dabei werden aus dem Schleuderstreuer Schneckenkörner ausgestreut, welche vom Prinzip her den gleichen Effekt für den Wurm haben. Die gestreuten Schneckenkörner sind Physik-Bodys, welche sich in der Physikwelt automatisch richtig verhalten. Eine Skizze des verwendeten Angriffsmodells ist in Abbildung 37: Skizze Schleuderstreuer zu sehen.

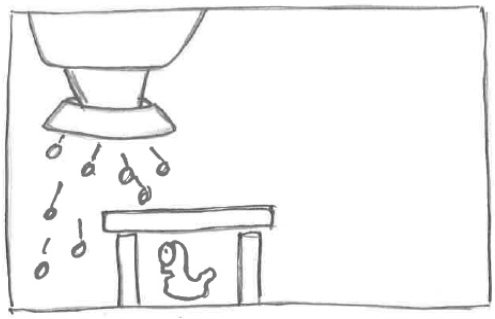


Abbildung 37: Skizze Schleuderstreuer

9.2 Kollisionsmodell

9.2.1 Einleitung

Mit dem Ziel, ein möglichst gutes Kollisionsmodell für beliebig komplexe Figuren zu erhalten, wurden im Prototyp drei Möglichkeiten ausprobiert.

Würden die Level-Elemente, Tools und Spielfiguren nur aus primitiven Formen (Kreis, Linie, Rechteck, Dreieck) bestehen, könnte die Box2D BodyFactory diese Aufgabe übernehmen und würde alle Elemente mit einem perfekten Kollisionsmodell abbilden (Kapitel 7.3.6). Wie bereits erwähnt, sollen jedoch auch komplexere Formen mit einem passenden Kollisionsmodell abgebildet werden können.

In den folgenden Unterkapiteln werden drei unterschiedlichen Varianten vorgestellt.

Da das Kollisionsmodell der Spielfigur bereits ziemlich komplex ist, wurde als Testobjekt jeweils die Grafik von Wilson verwendet. Der Vorteil bei der Verwendung dieser Grafik ist, dass anschliessend auch einfachere Formen problemlos dargestellt werden können. Am Schluss jeder Variante werden jeweils die Vor - und Nachteile aufgelistet, welche schlussendlich für den Entscheid verantwortlich waren.

9.2.2 Polygenes Shape

Der erste Versuch bestand darin, von den primitiven Shapes wegzukommen und mit einem Polygon zu arbeiten. Bei einem Polygon können bis zu maximal acht Eckpunkte definiert werden, welche anschliessend zu einem Shape verbunden werden. Ein mögliches Kollisionsmodell ist unterhalb ersichtlich.

Abbildung 38: Polygones Shape

Der Vorteil dieser Lösung besteht in der Performance. Da nur ein Shape pro Objekt verwendet wird, kann mit wenigen Shapes ein ganzes Level dargestellt werden.

Der Nachteil ist jedoch offensichtlich. Da ein Polygon auf acht Eckpunkte beschränkt ist, kann das Kollisionsmodell für eine komplexe Figur nur grob angenähert werden.

9.2.3 Komposition aus primitiven Shapes

Die zweite Variante bestand aus einer Komposition aus mehreren primitiven Shapes. Dies bedeutet für die Grafik von Wilson, dass zum Beispiel der Kopf mit zwei Kreisen und der restliche Körper mit Rechtecken dargestellt wird. Ein mögliches Kollisionsmodell ist unterhalb ersichtlich.

Abbildung 39: Komposition aus primitiven Shapes

Als Vorteil dieser Variante gilt die Genauigkeit des Kollisionsmodells im Gegensatz zur Lösung mit nur einem Polygon.

Ein Nachteil besteht jedoch in der Erstellung des Kollisionsmodells, denn grundsätzlich könnte mit beliebig vielen primitiven Shapes und durch eine sehr exakte Anordnung ein nahezu perfektes Kollisionsmodell angefertigt werden. Die Erstellung der Shapes und die korrekte Ausrichtung wären jedoch ein enormer Aufwand. Als weiterer Nachteil gilt die Anzahl von Shapes, welche für ein komplexes Objekt erstellt werden müssten.

9.2.4 Komposition aus polygenen Shapes

Der Nachteil der Variante mit einem polygenen Shape war die Begrenzung auf maximal acht Eckpunkte. Wenn das Kollisionsmodell jedoch aus einer Komposition von mehreren polygenen Shapes besteht, können beliebig komplexe Kollisionsmodelle erstellt werden. Die einzige Bedingung dabei ist, dass die einzelnen Polygone, wie bereits erklärt, nur acht Ecken aufweisen dürfen. Ein mögliches Kollisionsmodell ist unterhalb ersichtlich.

Abbildung 40: Komposition aus polygonen Shapes

Der Vorteil dieser Variante besteht in der Genauigkeit des Kollisionsmodells.

Als Nachteile gelten jedoch wieder die Erstellung und Ausrichtung geeigneter Polygone sowie die Menge an Bodys die erstellt werden müssen.

9.2.5 Entscheid

Aufgrund der Anforderung an das Kollisionsmodell, dass beliebig komplexe Körper sehr genau dargestellt werden müssen, konnte die erste Variante nicht verwendet werden.
Der Entscheid zwischen „Komposition aus primitiven Shapes“ und „Komposition aus polygenen Shapes“ wurde, durch das Auffinden eines praktischen Tools, sehr einfach. Mithilfe des Physics Body Editors[15] kann eine beliebige Grafik anhand von selbst definierten Punkten in passende Polygone unterteilt werden. Das Tool übernimmt die mühsame Arbeit der Positionierung der unterschiedlichen Shapes. Aus diesem Grund wurde die Variante „Komposition aus polygenen Shapes“ verwendet.

9.3 Grafiken

Ähnlich wie im Kapitel Sound auf Seite 84 beschrieben, ist auch die Qualität der Grafiken für das Spielerlebnis ein ausschlaggebender Aspekt eines Games. Egal wie gut oder neuartig ein Spielprinzip ist, schlussendlich bestimmt der Spielspass, ob jemandem ein Spiel gefällt oder nicht. Durch schlechte oder unpassende Grafiken kann dieser Spielspass sehr stark gemindert werden.
Die Mitglieder des Projektteams haben nur rudimentäre Kenntnisse im Bereich Gestaltung und Design. Mithilfe dieser grundlegenden Kenntnisse konnten Skizzen und Platzhalter-Grafiken angefertigt werden, welche ihren Zweck die Prototypen erfüllten.
Das Projektteam entschied sich jedoch dafür, einen Grafiker zu kontaktieren, der die endgültigen Grafiken anfertigte. Durch den Einsatz des Grafikers konnte das Spiel im Hinblick auf das Design bereits jetzt auf einen marktreifen Stand gebracht werden.

Abbildung 41: Wilson Skizze

Abbildung 42: Wilson endgültige Grafik

In den oberhalb ersichtlichen Abbildungen sind die anfängliche Skizze sowie die ausgearbeitete Grafik von Wilson dargestellt.

Im Anhang C Grafikverzeichnis ist aufgelistet, welche Grafiken vom Designer erstellt wurden.

[15] Seite 73 Physics Body Editor

9.4 Dynamic Loading

Jedes Spiel besitzt eine Game-Logik, welche das Spiel einzigartig macht. Dabei wird das Game-Konzept mit den entsprechenden Regeln implementiert. In Wilson's Adventures sind dies Dinge wie zum Beispiel der Angriff oder das Drag'n'Drop der verschiedenen Hilfsmittel. Ebenso gehören das Kollisionsmodell sowie die Collision Detection dazu.
Mit der alleinigen Umsetzung der Game-Logik ist das Spiel jedoch noch nicht spielbar. Es müssen Levels erstellt werden, bei welchen schlussendlich die Game-Logik zur Anwendung kommt. Dies kann auf zwei Möglichkeiten geschehen. Entweder wird jedes Level im Code erstellt oder man setzt auf ein dynamisches Prinzip, bei welchem die Levels in externen Files definiert werden.

Der Entscheid, ob das Game auf einer gut erweiterbaren Basis aufbaut oder nicht, wurde vom Projektteam bereits sehr früh getroffen. Dieses Prinzip bietet hauptsächlich Vorteile und unterstützt das Prinzip der Wiederverwendbarkeit sehr stark. Die Alternative, jedes Level im Programmcode zu definieren war somit für das Projektteam uninteressant. Die Vorteile der dynamischen Variante sind unterhalb zusammengefasst:

- Für die Level-Erstellung sind keine Programmier-Kenntnisse nötig.
- Die Level-Erstellung kann völlig unabhängig vom Rest des Spiels vorgenommen werden.
- Level-Definitionen und Programm-Code sind sauber getrennt.
- Das Prinzip, dass die Levels extern definiert sind und eingelesen werden, kann wiederverwendet werden, unabhängig von der Game-Logik.

Auch die Unterteilung der Levels in verschiedene Kategorien ist in externe Files ausgelagert und kann somit jederzeit für ein anderes Spielprinzip wiederverwendet werden.

9.4.1 File-Format

Bezüglich der Erstellung von externen Files mit Level- und Kategorie-Definitionen wurde auf das bewährte XML-Format gesetzt. Grund dafür war, dass die meisten Level Editoren auch auf das XML-Format setzen. Ebenfalls ist das Parsing von XML-Files in Java sehr angenehm.

9.4.2 XML-Generierung

Wenn man sich für den dynamischen Ansatz der Level-Erstellung im XML-Format entscheidet, gibt es zwei Möglichkeiten. Entweder verwendet man einen bestehenden Level Editor, aus welchem man das erstellte Level als XML exportieren kann oder man definiert eigene XML-Tags.

Das Projektteam hat sich aufgrund folgender Punkte für ein eigenes XML-Format entschieden:

- Die Vorteile eines Editors konnten nicht effizient genug genutzt werden. Konkret bedeutet dies, dass die XML-Files sowieso angepasst werden müssen um alle benötigten Informationen mitzugeben.

- Durch das definierte Spielprinzip sind die Levels einfach aufgebaut. Dies bedeutet, sie können zwar mehrere Elemente beinhalten, jedoch wird für die Ausrichtung von Objekten kein grafischer Editor benötigt.

9.5 Level-Kategorien

Die Aufteilung der Levels in Level-Kategorien ist ein vielfach verwendetes Prinzip bei Games aus dem Casual Genre und entspricht den Best-Practices für dieses Genre. Eine Listenansicht oder aufwändigere Level-Auswahlen würden für ein Spielprinzip mit kurzen Levels nicht passen. Das Prinzip der Level-Kategorien ist sich ein Spieler somit bereits gewöhnt. Er kann sich gut orientieren und hat klare Abschnitte, die er während des Games durchlaufen kann.
Ebenfalls bietet dies auch dem Entwickler die Möglichkeit das Game logisch zu unterteilen. Für Wilson's Adventures bedeutet dies, dass neue Angriffe und Tools ausgearbeitet und anschliessend in einer neuen Kategorie eingebaut werden können. Zusätzlich ist durch dieses Prinzip die Storyline gut gegliedert und kann problemlos erweitert werden.

9.6 Namensgebung: Wilson's Adventures

Nach der Definition der Storyline war klar, dass das Spiel aus einer Hauptfigur besteht, nämlich einem männlichen Wurm. Dieser männliche Wurm ist fester Bestandteil der einzelnen Levels. Ebenfalls gibt es einen weiblichen Wurm, welcher jedoch nur im Zusammenhang mit der Storyline vorkommt. Um die Storyline realistischer und mitreissender zu machen, wurden anschliessend passende Namen für die beiden Würmer gesucht. Dazu hat das Projektteam eine Umfrage gestartet, bei welcher sich die beiden Namen Wilson und Winnie durchgesetzt haben. In dieser Abstimmung wurde ebenfalls der Spielname Wilson's Adventures ermittelt.

10 Ergebnisse

Das Ergebnis dieser Bachelorarbeit ist nebst der Game-Studie ein Spiel für die Android Plattform. Wilson's Adventures baut auf der ursprünglichen Game-Idee (Seite 34) auf und enthält viele Ansätze aus dem Game-Konzept (Seite 37).
Das Game kommt im Casual Stil daher und ist deshalb von der Menüführung intuitiv und bei den meisten Spielern bekannt. Das Grafik-Konzept wurde von einem Grafiker erstellt und ist daher sehr ansprechend. Musik und Soundeffekte können durch das ganze Spiel hindurch gesteuert werden. Die Scoring-Logik ermöglicht das schrittweise Freischalten der Levels.
Wilson's Adventures bietet zwei Level-Kategorien an, welche beide unterschiedliche Schauplätze darstellen. Auf dem Schauplatz Farm muss Wilson vor gefährlichen Schneckenkörnern beschützt werden. In den Levels der Kategorie Wilder Westen muss sich Wilson gegen einen Sandsturm behaupten.
Der Spieler hat die Möglichkeit Hintergrund-Informationen zur Storyline zu erfahren. Ebenfalls kann über das Hilfe-Menü eine Spiel-Anleitung angesehen werden.

Im folgenden Kapitel ist die GUI-Map von Wilson's Adventures dargestellt. Darin sind alle unterschiedlichen Screens des Spiels ersichtlich. Vom Start des Spiel über den Android-Launcher, bis hin zu einem Level.

10.1 GUI-Map

Abbildung 43: GUI-Map

11 Ausblick

11.1 Einleitung

Wilson's Adventures bietet zum momentanen Zeitpunkt alle im Kapitel Ergebnisse erwähnten Features und ist voll funktionsfähig. Bevor es auf Google Play gestellt werden könnte, müssten noch kleine Anpassungen gemacht werde. In diesem Kapitel soll kurz erläutert werden, was für eine Veröffentlichung noch überarbeitet werden müsste.

11.2 Grafiken

Unabhängig vom Einsatz des Grafikers wurde zu Beginn des Projekts definiert, dass das Game auf eine Auflösung von 800x480 optimiert wird. Das Skalieren auf höhere Auflösungen wird dabei von der AndEngine übernommen. Leider gibt es dadurch teilweise Grafiken, deren Konturen nicht scharf gezeichnet werden. Dies hat zur Auswirkung, dass um gewisse Grafiken ein feiner Rahmen sichtbar wird, welcher nicht zum Bild gehört. Dieser Bug wird derzeit im AndEngine Forum diskutiert. Dem Projektteam ist dieses Problem vor allem bei der Verwendung eines Tablets aufgefallen.

Wenn Wilson's Adventures auf den Android-Market gestellt wird und dieser Bug noch nicht gelöst wurde, muss allenfalls ein anderer Ansatz umgesetzt werden. Eine Möglichkeit wäre die Skalierung selber zu implementieren oder die von Android zur Verfügung gestellte Variante zu verwenden. Dies hätte jedoch den Nachteil, dass für alle unterschiedlichen Auflösungen Grafiken bereitgestellt werden müssten.

11.3 Audio

Das Einbauen von passenden Musik- und Soundeffekten ist ebenfalls ein Punkt, welcher vor der Veröffentlichung überdacht werden muss. Da dem Projektteam die Erfahrung im Aufzeichnen von Sounds fehlt, müssten diese allenfalls von extern beschafft werden.

11.4 Spielumfang

Der wohl wichtigste Aspekt ist der Spielumfang. Damit das Spiel mit anderen Titeln auf Google Play mithalten kann, müssten noch einige Levels erstellt werden, welche die Spieler etwas mehr fordern.
Da bereits jetzt die Möglichkeit besteht Highscores für die verschiedenen Levels und Kategorien zu speichern, fehlt nur noch ein entsprechendes Modell, das den Erfolg des Spielers in Punkte abbildet.

12 Projektmanagement

12.1 Zweck

Dieses Dokument soll einen Überblick über die Zeitplanung, Organisation, Zuständigkeiten und über das Qualitätsmanagement im Projekt Android Game geben.

12.2 Projekt Übersicht

Einerseits soll eine Studie zur Game-Entwicklung auf der Android-Plattform durchgeführt werden, welche die unterschiedlichen Technologien und ihre Eigenschaften sowie verschiedene Game-Genres diskutiert.
Andererseits soll ein Game-Konzept entworfen und für die Android-Plattform implementiert werden.

12.3 Projektteam

	
Marco Pfiffner	**Mathias Fasser**
Projektmitarbeiter	Projektmitarbeiter

Tabelle 12: Projektteam

12.4 Management Abläufe

12.4.1 Projektumfang

Die Bachelorarbeit dauert 2x 360 Stunden und ist auf 16 Wochen beschränkt.

12.4.2 Abgabe

Die Dokumentation zum Projekt soll den Vorgaben der HSR und des IFS entsprechen und folgende Punkte enthalten:

- Technologie-Studie (Technischer Bericht)
- Game-Konzept
- Software Engineering Dokumente
- Glossar
- Literaturverzeichnis
- Projektmanagement
- Persönlicher Bericht

12.5 Meilensteine

Da zu Beginn der Arbeit eine Studie der Game-Technologien gemacht werden muss, kann dieser Teil nicht agil realisiert werden. Die Implementation des Games wird jedoch durch eine agile Vorgehensweise realisiert. Die unterschiedlichen Phasen: Inception, Elaboration, Construction und Transition werden in unterschiedliche Iterationen aufgeteilt. Die genaue Zeitplanung befindet sich unterhalb.

Projektplan

Version 1.0 22.02.2012

KW	SW	MS	Datum	Arbeiten	Phase	Spezielles
8	1		24.02.2012	Planung: -Projektplan -Projektmanagement	Inception I1	Kick-Off Meeting
9	2		02.03.2012	Infrastruktur: -Server (Git, Redmine, Jenkins)	Inception I1	
10	3	MS1	09.03.2012	Game-Entwurf -Papier-Prototypen -Storyline Anforderungen	Elaboration E1	
11	4	MS2	16.03.2012	Technologie-Studie - Physik-Engine	Elaboration E2	
12	5		23.03.2012	Einarbeitung in Game-Technologie	Elaboration E3	
13	6		30.03.2012	Prototypen	Elaboration E3	
14	7	MS3	06.04.2012	Prototypen	Elaboration E3	
15	8		13.04.2012	Game-Implementation / Grafiken	Construction C1	
16	9		20.04.2012	Game-Implementation / Grafiken	Construction C1	
17	10	MS4	27.04.2012	Game-Implementation / Grafiken	Construction C1	
18	11		04.05.2012	Game-Implementation / Grafiken	Construction C2	
19	12		11.05.2012	Game-Implementation / Grafiken	Construction C2	
20	13	MS5	18.05.2012	Game-Implementation / Grafiken	Construction C2	
21	14		25.05.2012	Testen / Überarbeitung Dokumentation	Transition T1	
22	15	MS6	01.06.2012	Testen / Überarbeitung Dokumentation	Transition T1	
23	16		08.06.2012	Überarbeitung Dokumentation	Transition T2	
24	17		15.06.2012	Überarbeitung Dokumentation	Transition T2	Abgabe

Milestones:
MS1: Game-Konzept
MS2: Technologie-Studie
MS3: Prototyp
MS4: Game-Architektur
MS5: Game
MS6: Review & Test

Abbildung 44: Projektplan

12.5.1 MS1: Game-Konzept

Zu Beginn der Arbeit soll ein Game-Konzept entworfen werden. Folgende Punkte sind Teil des Konzepts:

- Storyline
  - Schriftliche Beschreibung der Storyline
  - Skizzierung der involvierten Personen, Gegenstände, usw.
- Game-Play
  - Genaue schriftliche Beschreibung des Game-Ablaufs
  - Skizzierung des Game-Ablaufs mit allen möglichen Screens und Übergängen

12.5.2 MS2: Technologie-Studie

Anhand des Game-Konzepts wurde festgelegt wie das Game funktioniert und wie die verwendeten Grafiken aussehen. Anhand dieser Informationen soll mittels einer Studie ermittelt werden, welche Technologie am besten dafür geeignet ist, um dieses Game umzusetzen.

12.5.3 MS3: Prototyp

Bei diesem Meilenstein sollen wichtige Punkte der unbekannten Game-Technologie ausprobiert werden. Wenn möglich sollten Teile der Prototypen in der Construction Phase verwendet werden können.

12.5.4 MS4: Game-Architektur

Im Meilenstein MS4 soll die Game-Architektur implementiert werden. Dies beinhaltet die Entwicklung der Android App, in welcher anschliessend das Game mittels der gewählten Game-Technologie implementiert wird. Am Ende dieses Meilensteins sollen die Hauptkomponenten des Games implementiert sein. Das bedeutet, dass die Game-Struktur in grundlegenden Zügen besteht und anschliessend erweitert werden kann.

12.5.5 MS5: Game

Nachdem im MS4 die Grundlagen geschaffen wurden, geht es hier darum, mittels der bestehenden Grundkomponenten das Game zu erweitern. Dabei soll das Game-Konzept bereits zu einem Grossteil fertiggestellt werden. Ebenfalls sollen Erweiterungen im Bereich des Dynamic Loading und der Menüs stattfinden. Ebenfalls sollen bereits einige Levels designet werden, damit die Implementation getestet werden kann.

12.5.6 MS6: Review & Test

Während der gesamten Implementation wird, wo sinnvoll, mit Unit-Tests getestet. Im letzten Meilenstein geht es darum, das Game ausgiebig mittels Systemtests zu testen. Dabei wird das Gameplay auf mehreren Android Geräten getestet. Erkannte Schwachpunkte werden bereinigt.

12.6 Risikomanagement

Eine ausführliche Risikoanalyse befindet sich im Anhang E Risikomanagement.

12.7 Besprechungen

Jeden Mittwoch findet eine Besprechung an der HSR statt. Dabei werden das Projektteam und Herr Rudin teilnehmen.

Ziel der Besprechungen:

- Präsentation der erzielten Resultate
- Beurteilung der Lösungen
- Besprechung der weiteren Vorgehensweise

Zusätzlich zu den regelmässigen Meetings können auch noch weitere Besprechungen stattfinden.

12.8 Räumlichkeiten

Das Projektteam arbeitet an den zur Verfügung gestellten Laborarbeitsplätzen (Raum 1.206).

12.9 Zeiterfassung

Die Zeiterfassung wird mit Redmine (Web-Tool) erfasst und verwaltet. Die Unifiied Process Disziplinen entsprechen dabei den Aktivitäten in Redmine. Folgende Aktivitäten wurden erfasst:

- Business Modeling
- Requirements
- Design
- Implementation
- Test
- Deployment
- Configuration & Change Management
- Project Management
- Environment
- Study

Die Iterationen wurden in Redmine als Versionen abgebildet. Folgende Versionen wurden angelegt:

- Inception I1 – Kick-Off
- Elaboration E1 – MS1
- Elaboration E2 – MS2
- Elaboration E3 – MS3
- Construction C1 – MS4
- Construction C2 – MS5
- Transition T1 – MS6
- Transition T2 – Abgabe

Zusätzlich wurden folgende Ticket-Kategorien erstellt:

- Game-Logik
- Game-Technologie-Studie
- Gameplay
- Multimedia
- Storyline

Zu Beginn jeder Iteration wird eine Iterations-Planung erstellt. Dabei werden neue Tickets erfasst, welche die spezifischen Arbeiten und Ziele während dieser Iteration darstellen. Zu jedem Ticket wird jeweils eine Zeitschätzung gemacht. Anschliessend kann auf diese Tickets die benötigte Zeit verbucht werden. Somit kann am Ende jeder Iteration festgestellt werden, welche Arbeiten abgeschlossen wurden und welche Arbeiten in die neue Iteration übernommen werden müssen

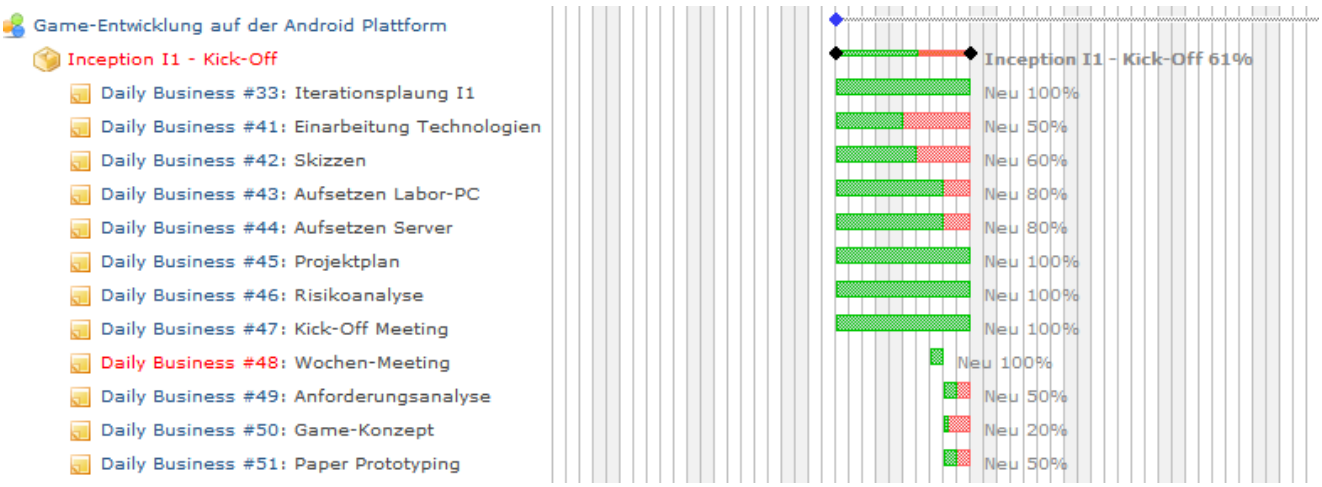


Abbildung 45: Redmine

.

12.10 Infrastruktur

12.10.1 Hardware und Software

Gegenstand / Service	Bereitgestellt durch	Support
2x Labor-PCs	IFS	
HTC Desire	Projektteam	
Samsung Galaxy S2	Projektteam	
Google Nexus S	IFS	
HTC one X	Projektteam	
Private Laptops	Projektteam	
Virtueller Server • GIT • Redmine • Jenkins	HSR	Projektteam
Dropbox • Dateiaustausch	Dropbox.com	Projektteam

Tabelle 13: Infrastruktur

12.10.2 Backup

Die Dokumente und der Sourcecode der Bachelorarbeit werden auf unterschiedlichen Servern abgelegt, diese sorgen für das Backup der Dateien.

- Dropbox (Binary-Files)
- GIT-Server (Sourcecode)

12.10.3 Verwendete Tools

- Eclipse 3.7 Indigo
- Google ADT Plugin
- Android SDK
- Find Bugs (Eclipse Plugin)
- State of Flow Metrics (Eclipse Plugin)
- STAN4J – Structure Analaysis for Java (Eclipse Plugin)
- Enterprise Architect
- Pencil (Pencil Project)
- Dropbox
- GIT
- Redmine
- Jenkins
- Microsoft Office
- Physics Body Editor
- Adobe Creative Suite

12.11 Qualitätsmanagement

12.11.1 Versionsmanagement

Für die Versionenverwaltung wird GIT (Sourcecode) und Dropbox (Binary-Files) verwendet. Somit ist der Zugriff auf die aktuellsten Versionen gewährleistet. Sämtliche Änderungen sind nachvollzieh- und umkehrbar. Ausserdem ist der Autor ersichtlich.

12.11.2 Code-Review

Beim Code-Review wird darauf geachtet, dass die gängigen „Programmier-Paradigmen“ eingehalten sowie das Wissen aus SE1 und SE2 angewandt werden. Der geschriebene Code wird wöchentlich zwischen den Teammitgliedern besprochen und refactored.

12.11.3 Code-Analyse

Der Code wird mit den beiden Code-Analyse Tools FindBugs und CheckStyle überprüft.

12.11.4 Unit-Tests

Die Projekt-Arbeit wird, wo sinnvoll, anhand von Unit-Tests getestet.

12.11.5 Systemtests

Nach den Meilensteinen in der Construction-Phase sowie in der Transition-Phase werden manuelle System-Tests durchgeführt. Bei den Systemtests wird die Usability (Gameplay) auf mehreren Android Geräten getestet.

12.11.6 Architekturentscheide

Architekturentscheide werden im Projektteam gemeinsam getroffen und allenfalls mit den Betreuern besprochen.

12.11.7 Metriken

Die Code-Metriken werden mithilfe von unterschiedlichen Metrik-Tools (STAN4J, State of Flow) analysiert.

13 Iterationsassessment

13.1 Inception I1

- Dauer: 2 Wochen

Aufgaben

Daily Business #33: Iterationsplaung I1
Daily Business #41: Einarbeitung Technologien
Daily Business #42: Skizzen
Daily Business #43: Aufsetzen Labor-PC
Daily Business #44: Aufsetzen Server
Daily Business #45: Projektplan
Daily Business #46: Risikoanalyse
Daily Business #47: Kick-Off Meeting
Daily Business #48: Wochen-Meeting
Daily Business #49: Anforderungsanalyse
Daily Business #50: Game-Konzept
Daily Business #51: Paper Prototyping

Abbildung 46: Tickets Inception I1

Ausstehende Tätigkeiten

Ticket #43 (Aufsetzen Labor-PC) und #44 (Aufsetzen Server) werden in einer späteren Iteration fertig gestellt. Folgende Tätigkeiten sind noch auszuführen:

- Aufsetzen Buildserver (Maven / Ant)
- Konfiguration von Eclipse (Maven- / Ant-Plugin)

13.2 Elaboration E1

- Dauer: 1 Woche

Aufgaben

Daily Business #34: Iterationsplaung E1
Daily Business #52: Technologiestudie
Daily Business #53: Anforderungsanalyse
Daily Business #54: Game-Konzept
Daily Business #55: Ausarbeiten Storyline
Daily Business #56: Wochen-Meeting
Daily Business #57: Grafik-Design

Abbildung 47: Tickets Elaboration E1

Ausstehende Tätigkeiten

Die Dokumente Anforderungsanalyse, Game-Konzept, Technologiestudie sowie Storyline wurden in einer ersten Version verfasst. Diese werden zu einem späteren Zeitpunkt überarbeitet und ergänzt.

13.3 Elaboration E2

- Dauer: 1 Woche

Aufgaben

Daily Business #35: Iterationsplaung E2
Daily Business #58: Studie Physik-Engine
Daily Business #59: Einarbeitung in Game-Technologie
Daily Business #60: Wochen-Meeting
Daily Business #61: Überarbeitung Anforderungsanalyse
Daily Business #62: Überarbeitung Game-Konzept
Daily Business #63: Überarbeitung Technologie-Studie

Abbildung 48: Tickets Elaboration E2

Da die Technologiestudie schon sehr fortgeschritten ist, kann sich das Projektteam schon früher als geplant in die Game-Technologie einarbeiten (Ticket #59).

Ausstehende Tätigkeiten

Die Einarbeitung in die Game-Technologie (Ticket #59) wird in der nächsten Iteration fortgesetzt.

13.4 Elaboration E3

- Dauer: 3 Wochen

Aufgaben

Daily Business #36: Iterationsplaung E3
Daily Business #64: Wochen-Meeting
Daily Business #65: Überarbeitung Dokumentation
Daily Business #66: Studie Physik-Engine
Daily Business #67: Einarbeitung in Game-Technologie
Daily Business #68: Prototyp I
Daily Business #69: Prototyp II
Daily Business #70: Grafik-Konzept

Abbildung 49: Tickets Elaboration E3

In dieser 3 wöchigen Iteration wird in der ersten Woche am Ticket „Einarbeitung in Game-Technologie“ weitergearbeitet. Dabei gilt es vor allem festzustellen, wie die benötigten Grafiken in das Game eingebunden werden. Somit ist die Grundlage geschaffen, um an den Prototypen in den zwei verbleibenden Wochen zu arbeiten. Ebenfalls kann aufgrund dieses Entscheids ein Grafik-Konzept für den Grafiker erstellt werden, in welchem definiert wird, welche Grafiken mit welcher Auflösung benötigt werden.

Ausstehende Tätigkeiten

Ticket #70 (Grafik-Konzept) konnte noch nicht vollends ausgearbeitet werden. In der nächsten Iteration wird zusammen mit dem Designer das Konzept verfeinert.

13.5 Construction C1

- Dauer: 3 Wochen

Aufgaben

Daily Business #37: Iterationsplaung C1
Daily Business #72: Wochen-Meeting
Daily Business #73: Überarbeitung Dokumentation
Daily Business #74: Definition XML Konfigurationsdateien
Daily Business #75: Definition Angriff / Auswertung
Daily Business #76: Definition Aufbau HUD
Daily Business #77: Implementation Dynamic Loading / XML Parsing
Daily Business #78: Implementation HUD / Toolbox
Daily Business #79: Implementation Angriff
Daily Business #80: Level Auswertung / Speicherung Spielstände
Daily Business #81: Kollisionsmodell Wurm
Daily Business #82: Soundkonzept
Daily Business #83: Sounds erstellen
Daily Business #84: Grafiken
Daily Business #85: Code Review

Abbildung 50: Tickets Construction C1

In dieser 3 wöchigen Iteration geht es darum, die Game-Architektur bereits zu einem Grossteil fertig zu stellen. Ebenfalls soll das dynamische Level-Loading aus XML Files funktionieren. Die Engine spezifischen Tätigkeiten wie Angriff, Kollisionsmodell und die Erstellung komplexer Körper sollte in die Architektur integriert werden.

Am Ende der Iteration C1 sollte eine primitive, jedoch spielbare Version des Spiels vorliegen. In dieser ersten Version geht es darum, dass ein Level von einem XML File eingelesen werden kann und dieses dargestellt wird. Anschliessend sollte die Möglichkeit bestehen, den Wurm durch Gegenstände zu schützen und eine Angriffswelle auszulösen, welche darüber entscheidet ob das Level bestanden ist oder nicht.

Ausstehende Tätigkeiten

Ticket #82 (Soundkonzept) und #83 (Sounds erstellen) wurden in dieser Iteration nicht bearbeitet. Da Soundeffekte noch nicht zwingend benötigt werden, können diese auch zu einem späteren Zeitpunkt eingefügt werden. Ebenfalls konnte das Ticket #80 (Level Auswertung) nicht ganz fertig gestellt werden.

13.6 Construction C2

- Dauer: 3 Wochen

Aufgaben

Daily Business #38: Iterationsplaung C2
Daily Business #86: Wochen-Meeting
Daily Business #87: Dokumentation
Daily Business #88: Zwischenpräsentation
Daily Business #89: Code Review
Daily Business #90: Implementation Particle System
Daily Business #91: Implementation Toolbox
Daily Business #92: Implementation Menüs
Daily Business #93: Überarbeitung Soundkonzept
Daily Business #94: Sounds erstellen
Daily Business #95: Planung Features

Abbildung 51: Tickets Construction C2

In der Iteration Construction C2, welche ebenfalls 3 Wochen dauert, wird einerseits die Zwischenpräsentation abgehalten, andererseits sollen alle restlichen Arbeiten am Code abgeschlossen werden. Am Ende der Iteration C2 soll eine spielbare Version von Wilson's Adventures vorliegen. Dazu müssen unter Anderem noch die unterschiedlichen Menüs sowie das Verhalten der Toolbar implementiert werden.

Ausstehende Tätigkeiten

Die aktuelle Version von Wilson's Adventures ist zwar spielbar, jedoch bestehen erst zwei unterschiedliche Levels und an vielen Stellen sind noch die Platzhalter-Grafiken des Projektteams, welche in den letzten vier Wochen durch die entsprechenden Grafiken des Designers ersetzt werden müssen.

Ebenfalls stehen noch diverse Refactorings sowie Ergänzungen an der Dokumentation an.

13.7 Transition T1

- Dauer: 2 Wochen

Aufgaben

Daily Business #39: Iterationsplaung T1
Daily Business #96: Dokumentation
Daily Business #97: Refeactoring
Daily Business #98: Test
Daily Business #99: Bugfixing
Daily Business #100: Meeting mit Grafiker
Daily Business #101: Wochen-Meeting

Abbildung 52: Tickets Transition T1

Da Herr Rudin nach dieser Woche nicht mehr anwesend sein wird, geht es in erster Linie darum die Dokumentation auf Vordermann zu bringen und diese soweit wie möglich fertigzustellen. Hauptziel ist es, gemeinsam eine endgültige Struktur zu definieren und alle grossen Kapitel möglichst auszuschreiben.

Ebenfalls muss ein Bug im Zusammenhang mit der AndEngine behoben werden. Ansonsten soll der Code vor allem refactored und getestet werden.

Ausstehende Tätigkeiten

Die Abgabe Dokumente (Poster und Broschüre) wurden in einem ersten Entwurf festgehalten. Diese Entwürfe gilt es nun zu überarbeiten und druckfertig zu machen.

13.8 Transition T2

- Dauer: 2 Wochen

Aufgaben

Daily Business #40: Iterationsplaung T2
Daily Business #103: Dokumentation
Daily Business #104: Code Review
Daily Business #105: Testing
Daily Business #106: Vorbereitung Abgabe
Daily Business #107: Grafiken

Das Ziel dieser Iteration ist die Arbeit sauber abzuschliessen. Dazu müssen die obligatorischen Abgabe-Dokumente, wie z.B. Poster, Broschüre und CD fertiggestellt werden. Ebenfalls soll das Game mit einigen Levels erweitert werden, dass die Besucher der Präsentation die Möglichkeit haben, ein paar Minuten zu spielen. Wo dies noch nicht geschehen ist, werden die Grafiken vom Designer ergänzt.

Ausserdem wird ein ausführlicher Systemtest durchgeführt und protokolliert.

14 Persönliche Berichte

14.1 Marco Pfiffner

Die gelungene Studienarbeit sowie die gute Zusammenarbeit mit Herrn Rudin haben dazu geführt, dass wir uns für eine Art Fortsetzungsarbeit der SA entschieden haben.

Das Ziel der Bachelorarbeit sollte es nun sein, unsere Kenntnisse im Bereich des Mobile-Developments, im speziellen der Android-Programmierung, zu erweitern. Da in der Studienarbeit bereits eine Business-Applikation von der Anforderungsanalyse bis zur Integration entwickelt wurde, wollten wir diesmal einen etwas anderen Aspekt des Android-Frameworks kennen lernen. Weil die Game-Programmierung an der HSR nicht unterrichtet wird, uns jedoch sehr interessiert, haben wir uns entschieden im Rahmen der Bachelorarbeit unseren Horizont etwas zu erweitern und erste Erfahrungen in der Game-Entwicklung zu sammeln.

Zu Beginn der Arbeit war es etwas schwierig für uns. Wir wussten nicht genau, wie man am besten an die Entwicklung eines Games heran geht, welche Bereiche speziell zu beachten sind und wie die üblichen Vorgehensweisen aussehen.

In der Folge haben wir zusammen diverse Spielideen diskutiert und von Hand skizziert. Ziel war es, ein möglichst vielseitiges und einfach erweiterbares Konzept zu finden, da wir schlecht abschätzen konnten, was während dieser Bachelorarbeit alles umgesetzt werden kann.

Nach diversen Diskussionen und etlichen Bleistiftspitzen haben wir uns für Wilson, den sympathischen kleinen Wurm, entschieden. Aspekte wie die Physikwelt, das Kollisionsmodell sowie die lustige Story waren für diesen Entscheid tragend. Ebenfalls hatten wir ein einfach zu erweiterndes Konzept gefunden, für das wir bereits früh viele Ideen hatten.

Als wir die grobe Spielidee gefunden hatten, ging es darum die funktionalen und nichtfunktionalen Anforderungen zu definieren sowie das Game-Konzept mit den unterschiedlichen Ideen zu dokumentieren. Mit Hilfe der Studie konnte anschliessend eine passende Game-Engine evaluiert werden.

Die ersten Eindrücke der AndEngine stimmten mich zunächst sehr skeptisch. Nach einigen Tutorials und mit der Hilfe des Lehrbuchs [LeAnGaPro] konnten wir jedoch die Prototypen wunschgemäss umsetzen.

Der Umgang mit der AndEngine erwies sich aber während des ganzen Projekts als nicht ganz einfach, da relativ wenig Dokumentation zu finden ist und das Lehrbuch auf der alten Version der Engine basiert.

Abschliessend darf ich auf 16 lehrreiche Wochen und eine sehr spannende, wenn auch etwas stressige Zeit zurückschauen. Die Zusammenarbeit mit meinem Team-Kollegen verlief, wie gewohnt, ohne Probleme. Ebenfalls habe ich den erneut guten Kontakt zu unserem Betreuer, Prof. Hans Rudin, sehr geschätzt.

Ich habe gelernt, dass in der Game-Entwicklung die erste Phase des Projekts sehr wichtig ist. Ohne ein detailliertes Game-Konzept, wird es später schwierig die Architektur des Games zu konzipieren. Dabei sollte man gerade auch die „einfachen“ Dinge wie den File I/O sowie die verschiedenen Input-Möglichkeiten für den Spieler nicht vernachlässigen. Ebenfalls ist es wichtig sich in die Rolle des Spielers versetzen zu können und das Konzept kritisch zu überdenken.

Es wird sich zeigen, ob Wilson’s Adventures in absehbarer Zeit auf dem Android Market landet oder ob wir gar ein Nachfolge-Projekt umsetzen werden.

14.2 Mathias Fasser

Während der Semesterarbeit hatte ich die Möglichkeit, mich mit einer Business-Applikation in die Android-Programmierung einzuarbeiten. Ebenfalls bot sich mir dabei die Möglichkeit, nebst den Android-Technologien auch die Chancen und Schattenseiten der Smartphone Märkte kennen zu lernen.

Computerspiele haben mich seit jeher interessiert. Früher nur als Hobby, etwas später aus Faszination an neuen Technologien und den Ideen der Spieleentwickler. Die Herausforderung während der Bachelorarbeit auf den Android-Game-Markt hinzuarbeiten und gleichzeitig lang ersehnte Einblicke in die Game-Programmierung zu bekommen, war für mich eine vielversprechende Kombination.

Die kreative Arbeit, die vor allem in den ersten Wochen sehr stark und während der ganzen Arbeit immer wieder auftauchte, machte mir sehr viel Spass. Es war interessant zu sehen, wie anfängliche Handskizzen zu einem durchdachten Spielprinzip heranwuchsen, welches Potenzial bietet.
Nebst Kenntnissen über die Game-Programmierung erlangte ich während der anschliessenden Studie Einblick in das Angebot an Game-Engines für die Android Plattform. Der Entscheid bezüglich der zu verwendeten Engine, der auf umfassenden Recherchen basierte, zahlte sich während der ganzen Arbeit aus. Diese Vorgehensweise beim Einsatz mit einer neuen Technologie werde ich, wenn möglich, auch in Zukunft anwenden.
Während den Prototypen beschäftigte ich mich intensiv mit der AndEngine und der Physic-Engine Box2D um die einzelnen Teile des definierten Game-Konzepts umzusetzen. Die grössten Herausforderungen stellten dabei das Kollisionsmodell und die dazugehörige Collision Detection dar.
Da Marco Pfiffner und ich ein eingespieltes Projektteam sind, arbeiteten wir oft zusammen, was uns gegenseitig Einblicke in alle aktuellen Arbeiten gab. Ebenfalls konnten Probleme durch diese Arbeitsweise gut diskutiert und effizient gelöst werden.

Da die parallel entwickelten Prototypen bereits viele wiederverwendbare Implementationen enthielten, stellten sie eine stabile Grundlage für das Projekt dar. Aufgrund der guten Fortschritte, welche wir während des Projektes erzielen konnten blieb viel Zeit für Diskussionen, wobei das entstehende Produkt – Wilson's Adventures – in richtig marktreife getrieben wurde.
Für die Unterstützung, die wir vom Betreuer erhielten, bin ich sehr dankbar. Bei den wöchentlichen Meetings wurden jeweils aktuelle Schwerpunkte vorgestellt und besprochen. Auch die Zusammenarbeit mit dem Designer war eine spannende Erfahrung, welche für ein Game und Programme mit dem Schwerpunkt auf dem User-Interface unumgänglich ist.

Als Lehre aus der Semesterarbeit hatte das Testen der Software bereits während der Arbeit eine hohe Priorität. Dies wurde mittels JUnit und Systemtests umgesetzt. Ebenfalls wurde dieses Mal die Dokumentation fortlaufend erweitert, was uns gegen Ende der Arbeit Vorteile verschaffte.

Obwohl ich anschliessend an mein Studium nicht hauptberuflich als Game-Entwickler arbeiten werde, bin ich mit der Themenwahl der Bachelorarbeit und mit dem daraus entstandenen Game sehr zufrieden. Durch meine Bachelorarbeit lernte ich die Game-Programmierung kennen. Dazu zähle ich nebst der direkten Arbeit mit der Game – und Physic-Engine auch das Prinzip des Dynamic-Loadings sowie das Einbinden von Grafiken und Sound als auch das Entwerfen des Scoring-Modells. Nebst der konkreten Implementation des Game-Konzepts, konnte ich auch meine allgemeinen Programmier- und Softwarearchitektur-Kenntnisse erweitern, was ich als einen guten Abschluss des Studiums empfinde. Ebenfalls war das Gefühl ein eigenes Produkt zu entwerfen allgegenwärtig. Ich freue mich darüber, die damit verbundenen Überlegungen, Kenntnisse und Herausforderungen während meiner Bachelorarbeit gemacht zu haben.

Ich hoffe, dass ich auch weiterhin die Zeit finde, um mich mit der Game-Programmierung auseinander zu setzen. Ob ich meine Android-Market-Erfahrungen mit Wilson's Adventures oder mit einer neuen Game-Idee machen werde, wird sich noch zeigen.

15 Glossar

Begriff	Definition
Android Developers	developer.android.com Portal für Android Programmierer
Android Launcher	
Android Market	Heisst neu Google Play
IFS	Institut für Software (HSR)
Freeware	Kostenlos verwendbare Software
Google Code	Google's offizielle Developer Seite
Google Play	Marktplatz zum Erwerben / Herunterladen von Android Apps
Library	Sammlung von Programmfunktionen
Open Source	Quelloffene Software (Programmcode frei verfügbar)
Sourceforge	Webportal zur Verwaltung von OpenSource-Projekten

Tabelle 14: Glossar

16 Verzeichnisse

16.1 Quellen

Referenz	Quelle
[BeAnGa]	Mario Zechner: Beginning Android Games ISBN: 978-1-4302-3042-7
[GameCareerGuide]	http://www.gamecareerguide.com/features/529/what_is_a_game_.php
[LeAnGaPro]	Rick Rogers: Learning Android Game Programming ISBN: 978-0-321-76962-6
[SA]	Marco Pfiffner, Mathias Fasser: Studienarbeit, CENTRADO Android App
[SanEl]	Sandi Elezovic: Designer
[WikiDEComputerSpiel]	http://de.wikipedia.org/wiki/Computerspiel
[WikiENBox2D]	http://en.wikipedia.org/wiki/Box2d

Tabelle 15: Quellen

16.2 Abbildungen

16.3 Code-Snippets

16.4 Tabellen

17 Anhang

A) Aufgabenstellung

Aufgabenstellung Bachelorarbeit für Herrn Mathias Fasser und Herrn Marco Pfiffner „Game-Entwicklung auf der Android-Plattform“

1. Auftraggeber, Betreuer und Experte

Bei dieser Arbeit handelt es sich um ein Thema, das von den Studierenden eingebracht wurde.

Ansprechpartner Auftraggeber:

Betreuer HSR: Prof. Hans Rudin, HSR, hrudin@hsr.ch; +41 55 222 49 36
Experte: Daniel Hiltebrand, Crealogix

2. Studierende

Diese Arbeit wird als Bachelorarbeit an der Abteilung Informatik durchgeführt von

- Mathias Fasser, mfasser@hsr.ch
- Marco Pfiffner, m1pfiffn@hsr.ch

3. Ausgangslage

In ihrer Studienarbeit haben sich Herr Fasser und Herr Pfiffner mit der Entwicklung einer Client-Applikation in die Programmierung der Android-Plattform eingearbeitet. Daraus entstand der Wunsch, sich vertieft mit dieser Plattform auseinanderzusetzen und ein Game auf Android zu entwickeln. Die Studierenden haben einen entsprechenden Vorschlag eingereicht, der in dieser Arbeit umgesetzt werden soll.

Den Betreuer interessieren bei dieser Arbeit ,neben dem eigentlichen Game, die Möglichkeiten (Art der Spiele, Technologien, ...) und Grenzen der Gameprogrammierung auf der Android-Plattform.

4. Aufgabenstellung

Diese Arbeit umfasst zwei Teile:

1. Eine Studie zur Gameentwicklung auf der Android-Plattform
2. Die Entwicklung eines Games auf der Android-Plattform

Der genaue Inhalt der Studie wird zwischen Betreuer und Studierenden im Verlauf der Arbeit festgelegt. Die Gameentwicklung ist nachfolgend im Wortlaut des Vorschlages der Studierenden beschrieben:

Vorschlag der Studierenden zur Gameentwicklung

Es soll ein Game für die Android Plattform entwickelt werden. Dabei wird in einer ersten Phase das Gamekonzept erstellt. Dadurch soll der genaue Spielablauf definiert und die beinhalteten Elemente skizziert werden. Ebenfalls wird in dieser Phase die Storyline festgelegt, was schlussendlich konkrete Auswirkungen auf das ganze Game hat.

In einer zweiten Phase wird eine geeignete Technologie evaluiert. Der nächste Schritt besteht in der Entwicklung des Games.

Das Game sollte aus der Familie der „Casual Games“ stammen. Dies bedeutet, dass das Game jederzeit gespielt werden kann und dadurch in kurze, nicht zusammenhängende Levels aufgeteilt ist. Das Game soll für ein breites Publikum geeignet sein und somit keine Vorkenntnisse oder spezielle Fertigkeiten voraussetzten.

Beim Game geht es in erster Linie darum, ein Objekt vor herunterfallenden Gegenständen zu schützen. Dies kann erreicht werden, indem das Objekt in einen geschützten Bereich transportiert wird oder das Objekt mit Gegenständen geschützt wird. Dies wird durch Einsatz einer Physik-Engine ermöglicht.

Die Architektur dieses Games soll die Möglichkeit bieten, dass das Game beliebig erweitert werden kann. Dies ist einerseits durch die Levels möglich, andererseits durch das Hinzufügen von neuen Elementen, welche der Spieler einsetzen kann.

Ebenfalls sollte das Game grafisch ansprechend sein.

In dieser Bachelorarbeit geht es hauptsächlich darum, sich mit dem Gamedesign zu beschäftigen. Ob das Spiel schlussendlich auf den Android Markt gestellt wird, wird am Schluss entschieden.

5. Zur Durchführung

Mit Betreuer finden wöchentliche Besprechungen statt. Zusätzliche Besprechungen sind nach Bedarf durch die Studierenden zu veranlassen.

Alle Besprechungen sind von den Studierenden mit einer Traktandenliste vorzubereiten, die Besprechung ist durch die Studierenden zu leiten und die Ergebnisse sind in einem Protokoll festzuhalten, das den Betreuern und dem Auftraggeber per E-Mail zugestellt wird.

Für die Durchführung der Arbeit ist ein Projektplan zu erstellen. Dabei ist auf einen kontinuierlichen und sichtbaren Arbeitsfortschritt zu achten. An Meilensteinen gemäss Projektplan sind einzelne Arbeitsresultate in vorläufigen Versionen abzugeben. Über die abgegebenen Arbeitsresultate erhalten die Studierenden ein vorläufiges Feedback. Eine definitive Beurteilung erfolgt auf Grund der am Abgabetermin abgelieferten Dokumentation.

6. Dokumentation

Über diese Arbeit ist eine Dokumentation gemäss den Richtlinien der Abteilung Informatik zu verfassen (siehe https://www.hsr.ch/Allgemeine-Infos-Diplom-Bach.4418.0.html). Die zu erstellenden Dokumente sind im Projektplan festzuhalten. Alle Dokumente sind nachzuführen, d.h. sie sollten den Stand der Arbeit bei der Abgabe in konsistenter Form dokumentieren. Alle Resultate sind vollständig auf CD/DVD in 3 Exemplaren abzugeben. Der Bericht ist ausgedruckt in doppelter Ausführung abzugeben.

7. Termine

Siehe auch Terminplan auf https://www.hsr.ch/Termine-Diplom-Bachelor-und.5142.0.html.

Montag, den 20. Februar 2012	**Beginn der Bachelorarbeit,** **Ausgabe der Aufgabenstellung durch die Betreuer**
8. Juni 2012	Abgabe Kurzbeschreibung und A0-Poster. Vorlagen stehen unter den allgemeinen Infos Diplom-, Bachelor- und Studienarbeiten zur Verfügung.
15. Juni 2012, 12:00	**Abgabe** der Arbeit an den Betreuer bis 12.00 Uhr. Fertigstellung des A0-Posters bis 12.00 Uhr. Abgabe der Posters im Abteilungssekretariat 6.113.
15. Juni 2012	HSR-Forum, Vorträge und Präsentation der Bachelor- und Diplomarbeiten, 13 bis 18 Uhr
6.8. - 25.08.2012	Mündliche Prüfung zur Bachelorarbeit

8. Beurteilung

Eine erfolgreiche Bachelorarbeit zählt 12 ECTS-Punkte pro Studierenden. Für 1 ECTS Punkt ist eine Arbeitsleistung von 30 Stunden budgetiert (Siehe auch Modulbeschreibung der Bachelorarbeit https://unterricht.hsr.ch/staticWeb/allModules/19419_M_BAI.html.).

Für die Beurteilung ist der HSR-Betreuer verantwortlich.

Gesichtspunkt	Gewicht
1. Organisation, Durchführung	1/6
2. Berichte (Abstract, Mgmt Summary, technischer u. persönliche Berichte) sowie Gliederung, Darstellung, Sprache der gesamten Dokumentation	1/6
3. Inhalt*)	3/6
4. Mündliche Prüfung zur Bachelorarbeit	1/6

*) Die Unterteilung und Gewichtung von 3. Inhalt wird im Laufe dieser Arbeit mit den Studierenden festgelegt.

Im Übrigen gelten die Bestimmungen der Abteilung Informatik für Bachelorarbeiten.

Rapperswil, den 19. Februar 2012

Prof. Hans Rudin
Institut für Software
Hochschule für Technik Rapperswil

B) Nutzungsvereinbarung

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Vereinbarung

1. Gegenstand der Vereinbarung

Mit dieser Vereinbarung werden die Rechte über die Verwendung und die Weiterentwicklung der Ergebnisse der Bachelorarbeit *„Game-Entwicklung auf der Android-Plattform* von Herrn Mathias Fasser und Herrn Marco Pfiffner unter der Betreuung von Prof. Hans Rudin geregelt.

2. Urheberrecht

Die Urheberrechte stehen den Studierenden, Herrn Mathias Fasser und Herrn Marco Pfiffner, zu.

3. Verwendung

Das entwickelte Spiel basiert auf einer Idee der Studierenden, Herrn Mathias Fasser und Herrn Marco Pfiffner. Die Weiterentwicklung und allfällige Vermarktung bleibt daher den Studierenden vorbehalten.

Die anderen Ergebnisse der Arbeit (Studie, technische Lösungen usw.) dürfen sowohl von den Studierenden wie von der HSR nach Abschluss der Arbeit verwendet werden.

Rapperswil, den 25.4.12 .....
Der Student: Mathias Fasser

Rapperswil, den 25.4.12 .....
Der Student: Marco Pfiffner

Rapperswil, den 25.4.12 .....
Der Betreuer der Bachelorarbeit:
Prof. Hans Rudin

C) Grafikverzeichnis

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Android Game
Grafikverzeichnis

Bachelorarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Frühlingssemester 2012

Autoren:	Marco Pfiffner, Mathias Fasser
Betreuer:	Prof. Hans Rudin
Experte	Daniel Hiltebrand
Gegenleser:	Dr. Daniel Keller
Version:	1.0

1 Inhaltsverzeichnis

2 Einleitung

Dieses Dokument gibt Auskunft über die Entstehung der Grafiken, welche Wilson's Adventures beinhaltet. Die Entstehung wird über die Phasen hinweg aufgezeigt. Ebenfalls wird im Kapitel Resultat beschrieben, wie der endgültige Stand des Games bezüglich der Grafiken aussieht. Im Kapitel Grafiker wird Sandi Elezovic vorgestellt. Er hat dem Projektteam dabei geholfen, das Grafik-Konzept von Wilson's Adventures umzusetzen.

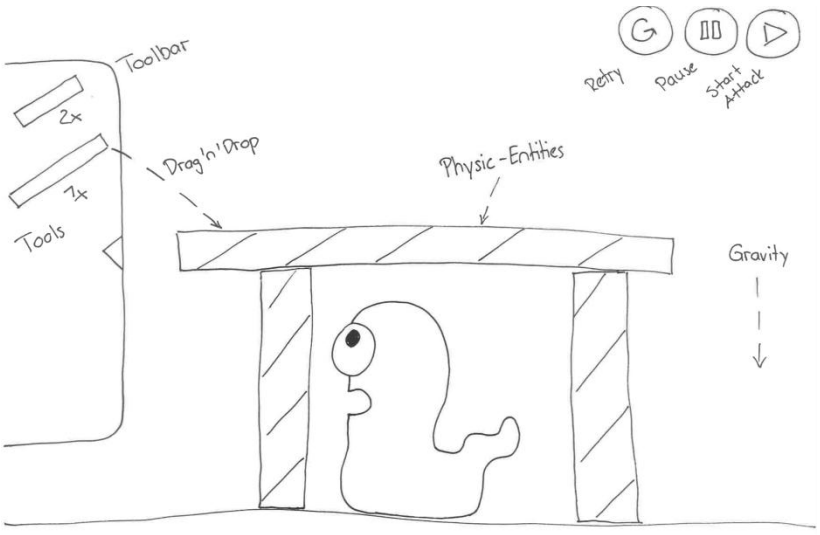
3 Phasen

Durch die Aufteilung in die vier Projekt-Phasen kann die Entstehung der Grafiken in einer klaren und strukturierten Weise dargestellt werden.

3.1 Inception

Während der Inception-Phase hat das Projektteam mittels eigens erstellten Skizzen die Game-Idee konkretisiert. Dabei ging es hauptsächlich darum sich klar zu werden wie das Spielprinzip aussieht.

Ebenfalls wurde mittels Skizzen eine geeignete Hauptfigur für das Spiel definiert. Mithilfe dieser Skizzen entstanden anschliessend das Game-Konzept und die Storyline.



Bereits im Verlauf der Inception hat sich das Projektteam mit Sandi Elezovic getroffen. Das Ziel des Meetings war es herauszufinden, ob er interessiert wäre, um Grafiken für diese Bachelorarbeit zu erstellen. Zu diesem Zeitpunkt war noch völlig unklar wie weit das Projektteam überhaupt kommt. Es konnte also keine abschliessende Aussage über die Menge an benötigten Grafiken oder über konkrete Grafiken getroffen werden. Ebenfalls konnte kein genauer Termin abgemacht werden, an dem die Grafiken benötigt werden.
Auch von der Seite des Grafikers konnten keine fixen Arbeitszeiten angegeben werden. Aus diesen Gründen wurden folgende Abmachungen für eine Zusammenarbeit definiert:

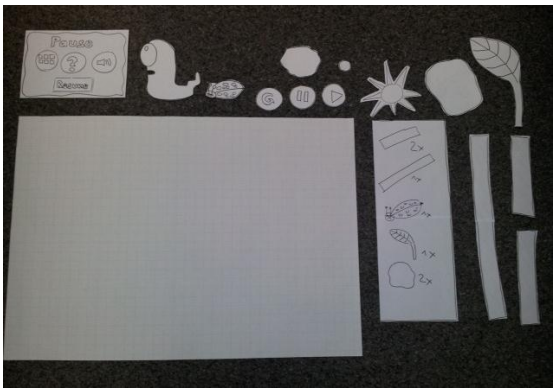
- Das Projektteam führt eine Liste mit den benötigten Grafiken.
- Die Grafiken müssen vom Projekteam jeweils priorisiert werden.
- Anschliessend arbeitet Sandi Elezovic, je nach Zeit und Lust, an den Grafiken mit der höchsten Priorität.
- Zu gegebener Zeit werden Meetings veranstaltet, bei welchen das Projektteam sehr eng mit dem Grafiker zusammenarbeitet. Dabei kann anhand von Skizzen besprochen werde, wie die endgültige Grafik aussehen sollte. Ebenfalls kann bei diesen Meetings über die Dimensionen der Grafiken diskutiert werden.

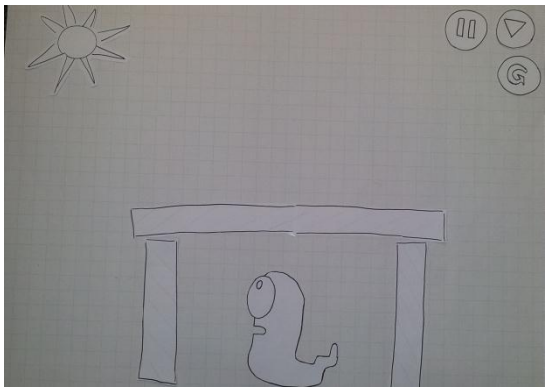
Durch diese Abmachungen wurden die vielen Unsicherheiten zwischen Projektfortschritt und der Zusammenarbeit zwischen Projektteam und Sandi Elezovic in einen möglichst geregelten Ablauf gebracht. Ob und wie viele Grafiken schlussendlich von Sandi Elezovic designet werden war dadurch noch nicht klar. Es war jedoch sichergestellt, dass immer diejenigen Grafiken erstellt werden, welche das Projektteam aktuell am dringendsten benötigt.

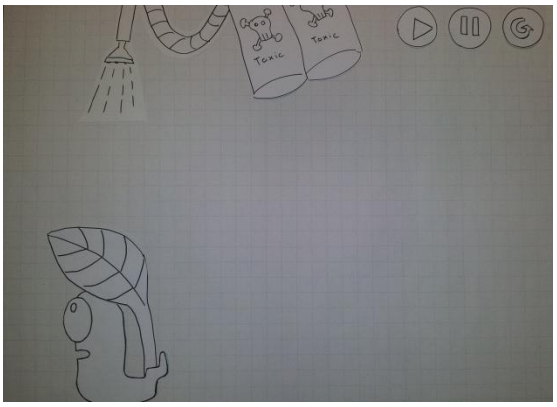
3.2 Elaboration

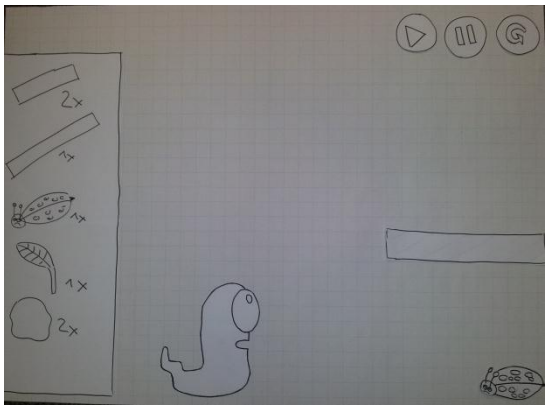
Zu Beginn der Elaboration-Phase wurden grafische Elemente aus dem Game-Konzept vom Projektteam skizziert und ausgeschnitten. Dadurch hatten die Projektmitglieder die Möglichkeit, auf einfache Art und Weise, sich gegenseitig neue Ideen vorzustellen.

Ebenfalls konnte somit auch die Ausrichtung der Buttons und der Toolbar einfach veranschaulicht werden.

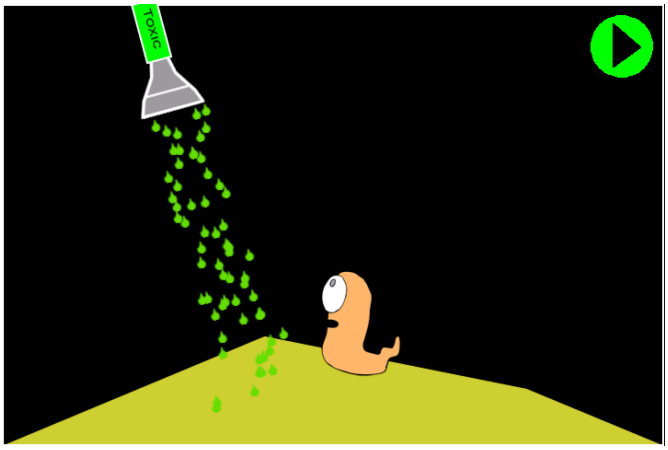








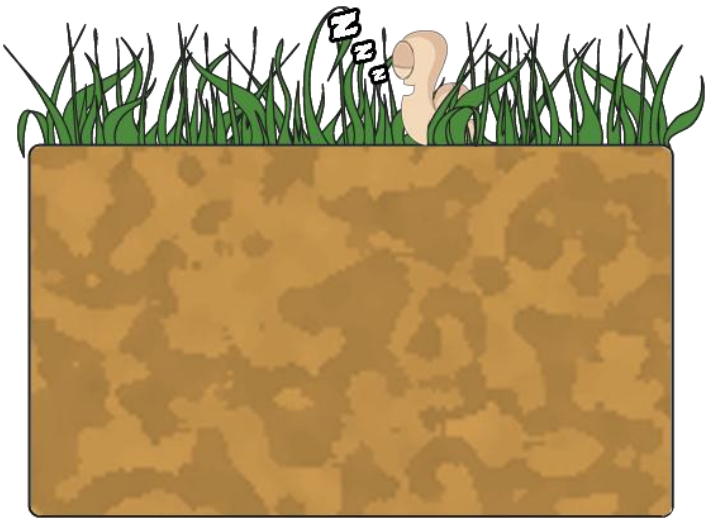
Sobald die Arbeit an den Prototypen begann, wurden die Skizzen digitalisiert um geeignete Platzhalter Grafiken verwenden zu können.



3.3 Construction

Während der Construction-Phase arbeitete Sandi Elezovic an den Elementen, welche Ingame[1] benötigt wurden. Dazu gehörten Wilson, sowie die Tools. Der Screenshot unterhalb zeigt einen Stand des Spiels, bei welchem Wilson bereits von Sandi Elezovic angefertigt wurde, der Rest jedoch noch vom Projektteam stammt.

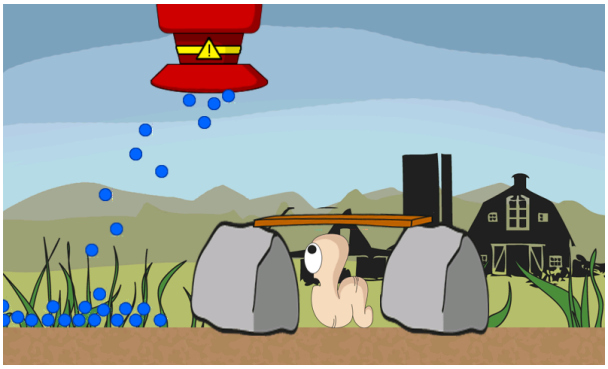
Im Verlauf der Construction-Phase wurde auch im Rahmen eines eintägigen Meetings intensiv mit dem Grafiker zusammen gearbeitet. Dabei entstanden auch bereits Menü-Hintergründe und gewisse Buttons.

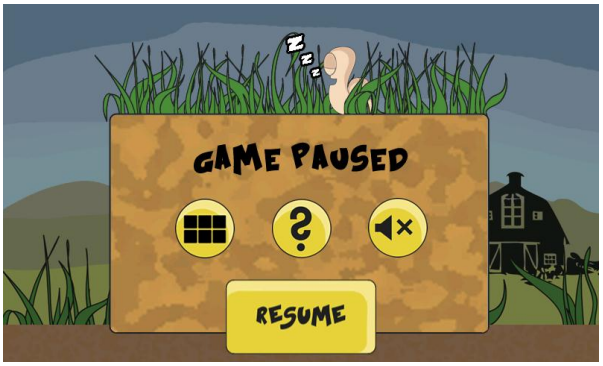


[1] Während des Spielverlaufs

3.4 Elaboration

Während der Elaboration wurden die restlichen Menü-Elemente von Sandi Elezovic fertig gestellt. Hintergründe für die Levels und die Menüs sowie das Logo, das auf dem Splash-Screen und dem Haupt-Menü verwendet wird, wurde wiederum während einem eintägigen Workshop angefertigt.





4 Resultat

Zum Zeitpunkt der Abgabe stammen alle in Wilson's Adventures verwendeten Grafiken von Sandi Elezovic. Das Projektteam freut sich, dass sich die Grafiken dadurch bereits auf einem marktreifen Stand befinden.

Die Zusammenarbeit mit Sandi Elezovic war sehr angenehm. Durch seine Flexibilität konnte in den ganztägigen Meetings wichtige grafische Elemente diskutiert werden, was dem Projektteam wichtig war.

Durch die Vergabe von Prioritäten an die Grafiken wurde sichergestellt, dass immer diejenigen Grafiken angefertigt wurden, welche in Bezug auf die Programmlogik wichtig waren und seitens des Designers keine Grafiken erstellt wurden, welche noch gar nicht benötigt werden. Die Anpassung des Grafikers an den aktuellen Fortschritt des Projekts hat dem Projektteam viele Vorteile gebracht und führte dazu, dass schlussendlich das ganze Grafik-Konzept von Sandi Elezovic umgesetzt werden konnte.

5 Grafiker

Das Projektteam kennt Sandi Elezovic bereits seit sieben Jahren. Damals gingen Sandi Elezovic, Mathias Fasser und Marco Pfiffner gemeinsam in die Berufsschule. Zum Zeitpunkt der Arbeit hat Sandi Elezovic das gestalterische Propädeutikum erfolgreich bestanden und wurde nun von der Zürcher Hochschule der Künste (ZHdK[2]) aufgenommen.
Das Projektteam würde sich über eine weitere Zusammenarbeit freuen.

[2] http://www.zhdk.ch/

D) Soundverzeichnis

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Android Game
Soundverzeichnis

Bachelorarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Frühlingssemester 2012

Autoren:	Marco Pfiffner, Mathias Fasser
Betreuer:	Prof. Hans Rudin
Experte	Daniel Hiltebrand
Gegenleser:	Dr. Daniel Keller
Version:	1.0

1 Soundverzeichnis

Da das Sound-Konzept während der Arbeit nicht so hoch priorisiert war, sind sämtliche Sounds in Wilson's Adventures als Platzhalter zu verstehen.

Natürlich durfte trotz Mangel an qualitativ guter bzw. geeigneter Sounds diese Komponente nicht im Game fehlen. Deshalb wurden folgende Sounds, welche für die nichtkommerzielle Verwendung frei zur Verfügung stehen, verwendet:

Sound	Quelle
/music/music.wav	http://www.freesound.org/people/lucasgonze/sounds/90102/
/sound/applause.wav	http://www.partnersinrhyme.com/soundfx/applause_sounds/applause10wav.shtml
/sound/boo.wav	http://www.partnersinrhyme.com/soundfx/applause_sounds/boo2wav.shtml
/sound/scream.wav	http://www.partnersinrhyme.com/soundfx/human_sounds/human_scream05_wav.shtml

E) Risikomanagement

Risikomanagement

Risiko Analyse	Version 1.0
Projektname: Android Game-Entwicklung	
Betreuer: Hans Rudin	
Projektmitglieder: Mathias Fasser, Marco Pfiffner	

Risiko Bewertungen							
Risk ID	Risiko	Auswirkung	Massnahme zur Verhinderung/Verminderung	maximaler Schaden in h	Wahrschein-lichkeit des Eintreffens	Gewichteter Schaden in h	Vorgehen beim Eintreffen
R01	Ausfall eines Projektmitglieds	Umfang muss angepasst werden.	keine Massnahmen	**360**	1%	**3.6**	Umfang anpassen -bei fortgeschrittenem Projektzeitpunkt kann die Anzahl der Levels minimal gehalten werden -ansonsten muss das Game in einfacherer Version entwickelt werden
R02	Krankheit/Teilausfall eines Projektmitgliedes	Mehraufwand für den Teampartner Umfang muss angepasst werden.	keine Massnahmen	**40**	10%	**4.0**	Umfang anpassen -bei fortgeschrittenem Projektzeitpunkt kann die Anzahl der Levels minimal gehalten werden -ansonsten muss das Game in einfacherer Version entwickelt werden
R03	Aufwand wurde falsch eingeschätzt	Projektziele können nicht vollständig realisiert werden	Projektplan sorgfältig erstellen	**60**	20%	**12.0**	Umfang anpassen -bei fortgeschrittenem Projektzeitpunkt kann die Anzahl der Levels minimal gehalten werden -ansonsten muss das Game in einfacherer Version entwickelt werden
R04	Storyline lässt sich mangels Kreativität nicht wie gewünscht ausarbeiten	Verzug bei der Realisierung / Storyline ist nicht so ansprechend wie erwartet	Einplanung von genügend Zeit für kreative Arbeit	**40**	15%	**6.0**	Storyline wird kurz gehalten, dafür mehr Wert auf Game selber gelegt
R05	Game-Technologie (Fachwissen) kann sich nicht schnell genug angeeignet werden	Verzug bei der Realisierung	Einplanung von genügend Zeit für Aneignung des Fachwissens	**50**	30%	**15.0**	Umfang anpassen -bei fortgeschrittenem Projektzeitpunkt kann die Anzahl der Levels minimal gehalten werden -ansonsten muss das Game in einfacherer Version entwickelt werden
R06	Erstellung von Grafiken ist zu aufwendig (fehlende Kenntnisse)	Projekt kann in ursprünglicher Form nicht realisiert werden / Grafiken sind nicht ansprechend designed	Eventuell abstrakte Variante des Games vorziehen / Einplaung von genügend Zeit für Grafik-Design	**50**	30%	**15.0**	Auf abstrakte Variante umsteigen / Grafikdesign auslagern
R07	Game-Konzept lässt sich schlecht umsetzen (Steuerung, Story usw.)	Gameplay ist für den Kunden nicht zufriedenstellend	-bekannte (Physik)-Engines verwenden -bewährte Input-Mechanismen verwenden -Hardware-Fragmentierung beachten -Während der Arbeit die Usability testen	**60**	10%	**6.0**	verwendete Physik-Engine anpassen / (Physik-Engine wechseln)
R08	Antwortzeiten von Betreuer/HSR sind länger als angenommen	Verzögerung des Projekttermin	Termine frühzeitig vereinbaren/ planen	**20**	5%	**1.0**	Anliegen besser kommnunizieren
R09	Hardwareausfall eines Projektmitgliedes	Arbeitsplatz muss neu eingerichtet werden / Verzögerungen	Hardware pflegen	**8**	20%	**1.6**	Ausweichmöglichkeit suchen / AP neu einrichten
R10	Ausfall virtueller Server	Verzögerung der Projekttermine	keine Massnahmen	**50**	1%	**0.5**	Ausweichmöglichkeit suchen / Backup
	Total Kosten in Arbeitspaketen enthalten					**64.7**	

F) Qualitätssicherung

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Android Game
Qualitätssicherung

Bachelorarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Frühlingssemester 2012

Autoren:	Marco Pfiffner, Mathias Fasser
Betreuer:	Prof. Hans Rudin
Experte	Daniel Hiltebrand
Gegenleser:	Dr. Daniel Keller
Version:	1.0

1 Inhaltsverzeichnis

2 Einleitung

Dieses Dokument zeigt die Umsetzung des Qualitätsmanagements aus dem Projektmanagement Kapitel der Dokumentation.

3 Code-Review

Während den wöchentlichen Code-Reviews hat sich das Projektteam mit den aktuellen Implementationen befasst. Dadurch konnten bereits viele Probleme behoben und Unschönheiten bereinigt werden. Ebenfalls wurde sichergestellt, dass die definierten Codestyles eingehalten wurden. Durch die enge Zusammenarbeit des Projektteams kam es durchaus vor, dass mehrere Code-Reviews pro Woche abgehalten wurden.

4 Code-Analyse

Dank den beiden Tools FindBugs und CheckStyle konnten kleinere Probleme und nicht mehr verwendeter Code entfernt werden.

5 Unit-Tests

Da das Unit-Testing von Games aufgrund der starken Abhängigkeit von der Benutzeroberfläche erschwert wird, konnten nur gewisse Teile des Programmcodes per Unit-Test getestet werden. Die Interaktionen mit der Benutzeroberfläche wurden daher mit umfänglichen Systemtestes abgedeckt.

Für die Packages Application , DB und Physics war es jedoch möglich mit geeigneten Mocking-Objekten gewisse Implementationen zu testen.

6 Systemtests

Im Kapitel Testgeräte sind die zur Verfügung stehenden Android Devices aufgelistet, welche für die Systemtests eingesetzt werden konnten.
Die folgenden drei Kapitel GUI, Game und Android Lifecycle stellen die Aufteilung der Systemtests in zusammengehörige Bereiche dar.

6.1 Testgeräte

Folgende Android Geräte standen für das Testen der App zu Verfügung.

Nr.	Hersteller	Typ	Android Version	Auflösung	dp
1	htc	Nexus S	2.3.6	480x800 px	hdpi
2	Samsung	Galaxy S2 GT-I9100	2.3.3	480x800 px	hdpi
3	htc	One X	4.0.3	1280x720 px	xhdpi
4	Samsung	Galaxy Tab	3.1	1280x800 px	yhdpi

6.2 01_GUI

6.2.1 Test Ziel

Mithilfe der GUI Tests sollen alle Szenen und alle darin enthaltenen Grafiken überprüft werden. Dabei gilt es auf die Proportionen und Schärfe der einzelnen Grafikelemente zu achten.

6.2.2 Testbedingungen

Um das GUI zu testen ist es wichtig, Geräte mit verschiedenen Auflösungen zu verwenden. Ebenfalls wird mit einem Tablet getestet.

6.2.3 Testgeräte

Mit den Testgeräten 2, 3 und 4 werden drei verschiedene Auflösungen abgedeckt.

6.2.4 Ergebnisse

Die Testergebnisse vielen wie erwartet aus. Dies bedeutet, dass auf den Geräten 3 und 4, welche eine sehr hohe Auflösung haben, nicht alle Grafiken scharf dargestellt werden. Ebenso die Textur des Soundbuttons.
Bezüglich des Grafik-Bugs sind die bisher festgestellten Probleme, wie erwartet aufgetreten. Alle Probleme welche im Zusammenhang mit den Grafiken auftraten, sind im Kapitel Ausblick dokumentiert.

6.2.5 Massnahmen

Es mussten keine Massnahmen getroffen werden. Die Massnahmen, welche bei einer Publikation getätigt werden müssen, sind im Kapitel Ausblick festgehalten.

6.3 02_Game

6.3.1 Test Ziel

Das Ziel dieses Systemtests ist es, Wilson's Adventures in seinem vollen Umfang zu testen. Dazu gehören alle möglichen Interaktionen die einem Benutzer Angeboten werden. Ebenfalls viele mögliche Kombinationen, welche daraus entstehen können. Wichtig dabei ist, dass alle Levels durchgespielt werden und somit auch die daraus entstehenden Auswirkungen getestet werden.

Während dem ganzen Systemtest ist auf folgende Schwerpunkte zu achten:

- Funktioniert die Freischalte-Logik der Levels?
- Wie verhält sich die Ladezeit der Levels?
- Treten Exceptions auf?
- Wie verhält sich das Memory?
- Verhalten sich die Sounds wie erwartet?

Um all diese Schwerpunkte zu testen wurden drei Abläufe definiert, wie ein Benutzer die Levels durchspielen kann. Während jedem dieser Tests wird nebst anderen Interaktionen ein Level erfolgreich beendet. Alle Testfälle sind unterschiedlich und decken alle Scenes ab. Mit diesen drei Tests kann über alle verfügbaren Levels iteriert werden. Somit wird das ganze Spiel durchgespielt und alle möglichen Interaktionen abgedeckt.

6.3.2 Testbedingungen

Damit eine klare Aussage über die Schwerpunkte Ladezeit und Memory getroffen werden kann, wird hier mit allen zur Verfügung stehenden Testgeräten getestet, welche alle unterschiedliche Hardware Anforderungen mit sich bringen.

6.3.3 Testgeräte

Es werden alle vier Testgeräte verwendet.

6.3.4 Ergebnisse

Die Testergebnisse vielen sehr positiv aus. Bezüglich Freischalte-Logik, Ladezeit, Exceptions und Sounds sind keine Probleme aufgetreten.
Der Memory-Verbrauch war während den Tests auch nicht problematisch. Es wurde jedoch festgestellt, dass der Verbrauch an Memory anstieg und nicht immer freigegeben wurde. Dies ist ein Aspekt, welcher genauer untersucht werden muss, sobald Wilson's Adventures mit einer hohen Anzahl an Levels auf den Android-Markt gestellt wird.

6.3.5 Massnahmen

Aufgrund des positiven Testergebnisses wurden keine Massnahmen getroffen. Die Memory-Auslastung wird, wie im Kapitel Ergebnisse erwähnt, vor einer Publikation überarbeitet.

6.4 03_Game_Android_Lifecycle

6.4.1 Test Ziel

In diesen Tests wird Wilson's Adventures im Zusammenhang mit dem Android Lifecycle betrachtet. Dabei gilt es festzustellen, ob sich das Spiel bei Unterbrüchen korrekt verhält und anschliessend wieder wie erwartet im Spielablauf fortfährt.

6.4.2 Testbedingungen

Es werden drei Möglichkeiten getestet, die jeweils zu einem Unterbruch führen:

- Eingehender Anruf
- Locking / Unlocking
- Home-Button

Dabei wird unterschieden ob sich der Spieler im Menü oder auf der LevelScene befindet. Innerhalb der LevelScene wird unterscheiden ob der Angriff zurzeit läuft oder nicht. Alle Tests werden ebenfalls mit und ohne Musik durchgeführt. Somit kann auch sichergestellt werden, dass sich die Musik erwartungsgemäss verhält.

6.4.3 Testgeräte

Diese Tests können mit einem Gerät durchgeführt werden, da die Hardware keinen Einfluss auf den Android Lifecycle hat. Es wurde Testgerät 2 ausgewählt.

6.4.4 Ergebnisse

Alle Test-Ergebnisse fielen erfolgreich aus.

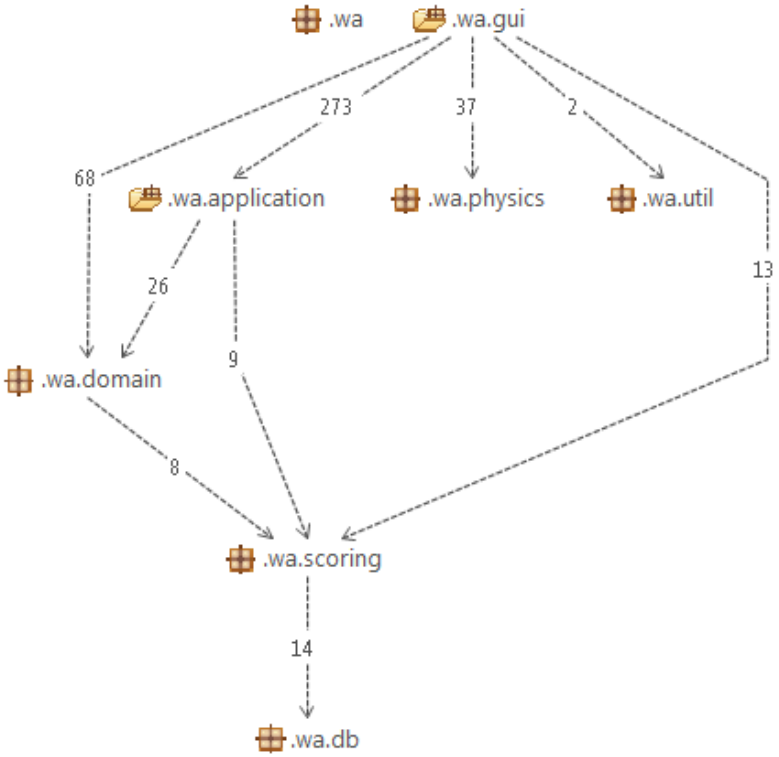
Die MenuScene ist bei allen Tests nach dem Unterbruch wieder erschienen. Bei den Tests der LevelScene wurde, unabhängig ob der Angriff lief oder nicht, das Pausenmenü eingeblendet. Wenn man anschliessend weitergespielt hat, wurde das Spiel an der richtigen Stelle fortgesetzt.
Die Musik hat sich ebenfalls an den Android-Lifecycle angepasst (Wilson's Adventures im Vordergrund: Musik an, Wilson's Adventures im Hintergrund: Musik aus) und verhält sich somit wie gewollt.

6.4.5 Massnahmen

Es mussten keine Massnahmen getroffen werden.

7 Metriken

Dank STAN4J konnten auf der Ebene der Top-Level Packages alle bidirektionalen Assoziationen aufgelöst werden.



Ausserdem wurden folgende Metriken mithilfe von STAN4J bestimmt:

Metrik	Wert
ELOC	3800
Libraries	2
Packages	16
Units	64
Units / Package	4
Methods / Class	4.7
Fields / Class	4.7
ELOC / Unit	58

G) Zeitauswertung

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Android Game
Zeitauswertung

Bachelorarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Frühlingssemester 2012

Autoren:	Marco Pfiffner, Mathias Fasser
Betreuer:	Prof. Hans Rudin
Experte	Daniel Hiltebrand
Gegenleser:	Dr. Daniel Keller
Version:	1.0

1 Zeitauswertung

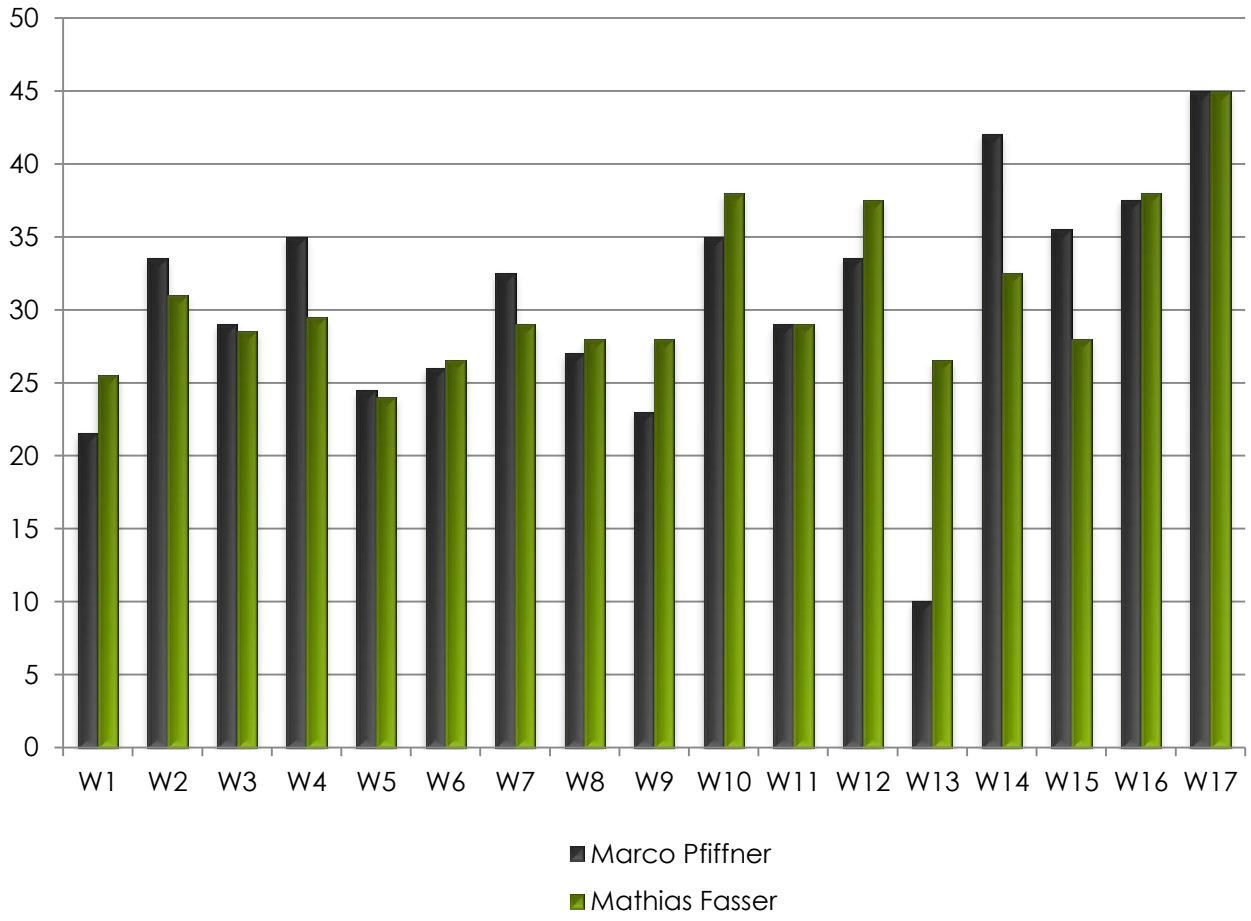
Dieses Dokument zeigt die aufgewendeten Stunden von Marco Pfiffner und Mathias Fasser in aufbereiteter Form. Die Stunden wurden, wie im Projektmanagement-Kapitel erläutert, in Redmine erfasst.

In der folgenden Tabelle ist der Gesamtaufwand beider Projektmitglieder aufgelistet:

Mitglied	Stunden
Marco Pfiffner	519.5
Mathias Fasser	524.5
Total	**1044.0**

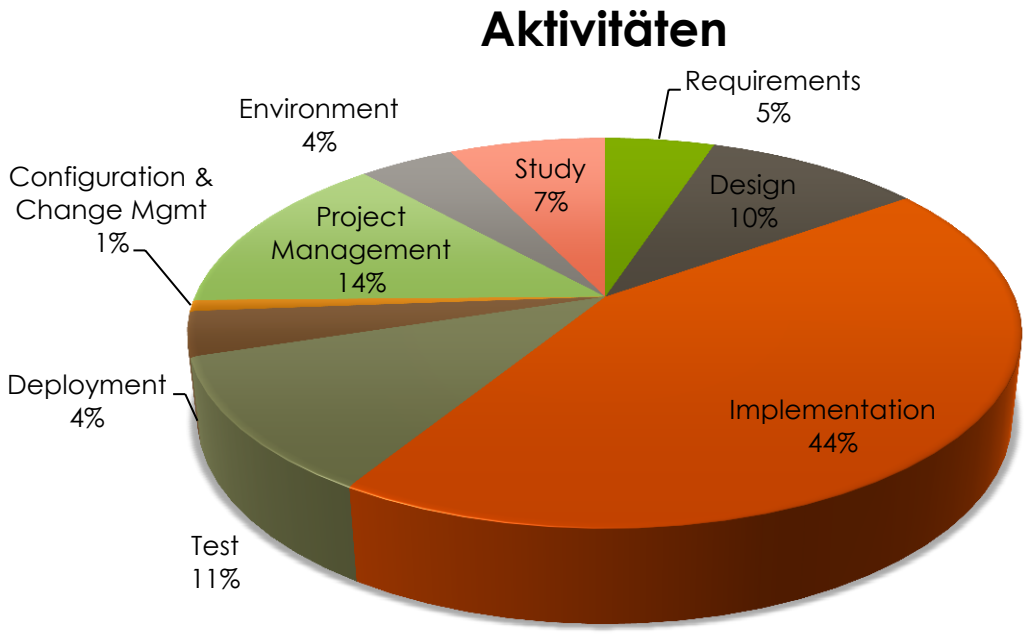
1.1 Wochenübersicht

In der Wochenübersicht ist zu sehen, dass während dem gesamten Projekt wöchentlich mehr als die vorgesehenen 20 Stunden gearbeitet wurde. Das Wochenmittel beider Projektmitglieder liegt bei ungefähr 31 Stunden.



1.2 Aktivitäten

Im untenstehenden Kuchendiagramm sind die prozentualen Anteile der unterschiedlichen Aktivitäten am Gesamtaufwand ersichtlich.



H) Abstract Broschüre

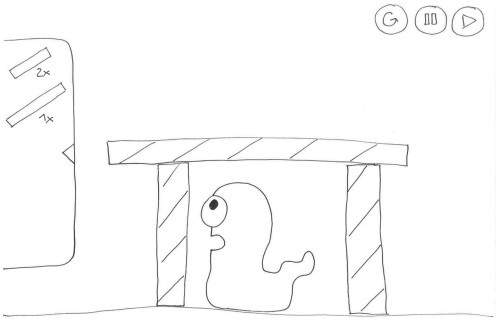

Mathias Fasser


Marco Pfiffner

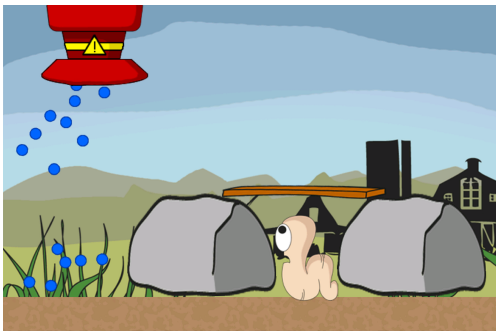
Diplomanden	Mathias Fasser, Marco Pfiffner
Examinator	Prof. Hans Rudin
Experte	Daniel Hiltebrand, Crealogix E-Business AG, Bubikon, ZH
Themengebiet	Software

Gameprogrammierung auf der Android Plattform

"Wilson's Adventures"



Skizze Spiel-Idee



Screenshot Wilson's Adventures

Ausgangslage: Der Einzug des Smartphones als treuer Begleiter vieler Menschen eröffnete eine neue Plattform für die Softwareentwicklung. Die Tatsache, dass heutzutage jeder Smartphone Besitzer eine Spielkonsole mit sich trägt, eröffnete auch für die digitale Spielindustrie neue Wege. Vor allem für kleinere Teams von Game-Entwicklern bietet sich heute eine erstklassige Möglichkeit um auf dem Markt mitzuwirken. Die bereits in der Semesterarbeit gesammelten Erfahrungen mit Android und die Faszination gegenüber der Spieleentwicklung, bewegten das Projektteam dazu, ein Game auf der Android Plattform zu entwickeln.

Vorgehen/ Technologien: Zu Beginn der Arbeit wurde ein Gamekonzept entwickelt, welches die Rahmenbedingungen des Games abstecken sollte. Dabei wurde der grobe Ablauf des Games mit den Interaktionen des Spielers, sowie die Storyline definiert. In einer Studie wurden grundlegende Information über die Game- Entwicklung gesammelt. Ebenfalls wurde eine passende Game-Engine evaluiert, mit welcher das definierte Game-Konzept umgesetzt werden konnte. Die Wahl fiel auf die AndEngine. Anschliessend wurde das Game mittels der AndEngine und der Physik- Engine Box2D implementiert. Während der Arbeit wurden mit mehreren Android Geräten Systemtests durchgeführt.

Ergebnis: Das Ergebnis dieser Bachelorarbeit ist ein Android-Game, namens Wilson's Adventures, welches zur Kategorie der Casual Games gehört. Der Spieler muss dabei in übersichtlichen Levels die Hauptfigur des Games, Wilson, beschützen. Per Drag'n'Drop können Gegenstände auf dem Level platziert werden, damit eine Angriffswelle von oben den Wurm nicht verletzt. Sofern Wilson von den Angriffselementen nicht getroffen wurde, gilt das Level als bestanden. Wilson's Adventures basiert auf Android Version 2.2. Konfigurationen und Level-Definitionen wurden in XML-Files ausgelagert und werden während dem Spiel eingelesen. Durch ein Kollisionsmodell, das beliebig komplexe Körper darstellen kann, und der Box2D Collision Detection wurde die Game- Logik implementiert. Zusätzlich wurde mittels SQLite ein Scoring- Modell realisiert. Ob die wenigen verbleibenden Schritte bis zur Marktreife noch gemacht werden, steht zu diesem Zeitpunkt noch nicht fest. Auf jeden Fall bilden viele der implementierten Komponenten eine gute Grundlage, um neue Game-Ideen zu verwirklichen.

andengine Box2D

Verwendete Game- und Physik-Engine

I) Poster

Gameprogrammierung auf der Android Plattform

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz


Mathias Fasser


Marco Pfiffner

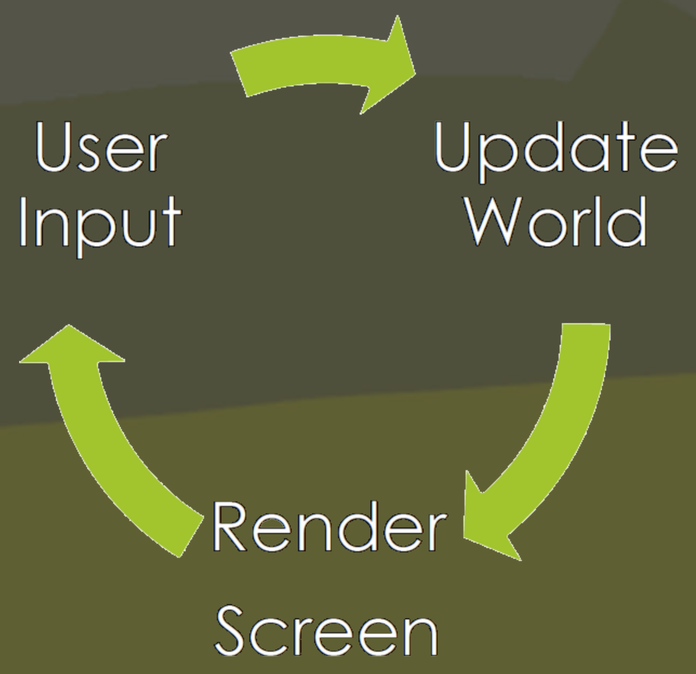
Bachelorarbeit
Frühjahrssemester 2012
Themengebiet Software

Betreuer: Prof. Hans Rudin
Experte: Daniel Hiltebrand
Gegenleser: Dr. Daniel Keller

Überblick:

- Game-Idee
- Game-Konzept
- Studie
- Evaluation Engine
- Implementation

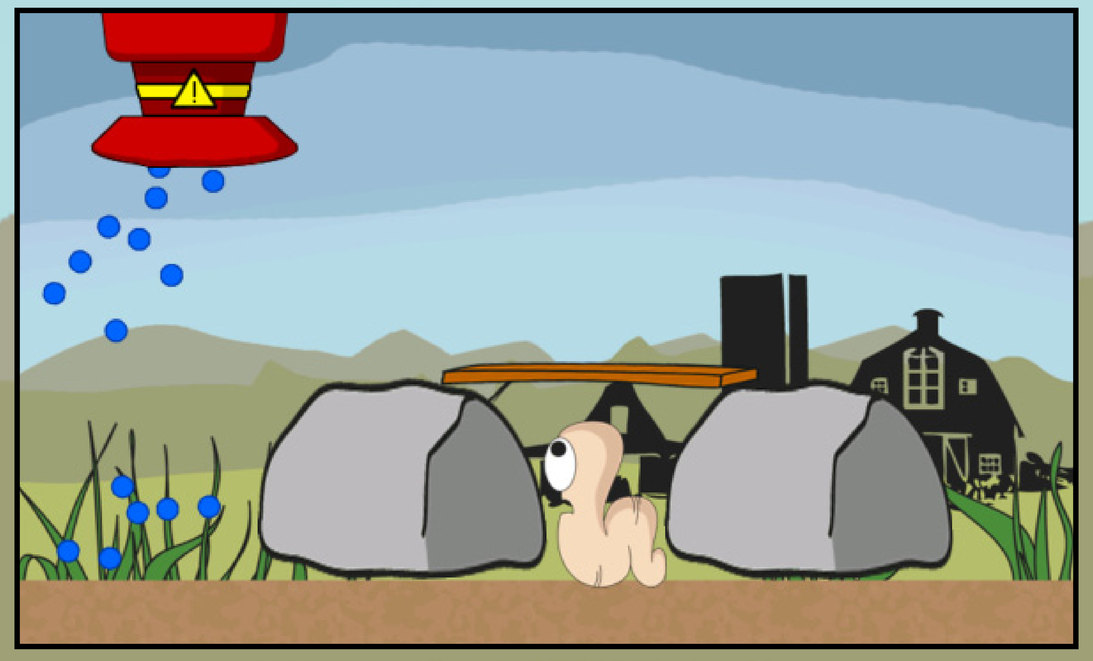
Game-Loop:

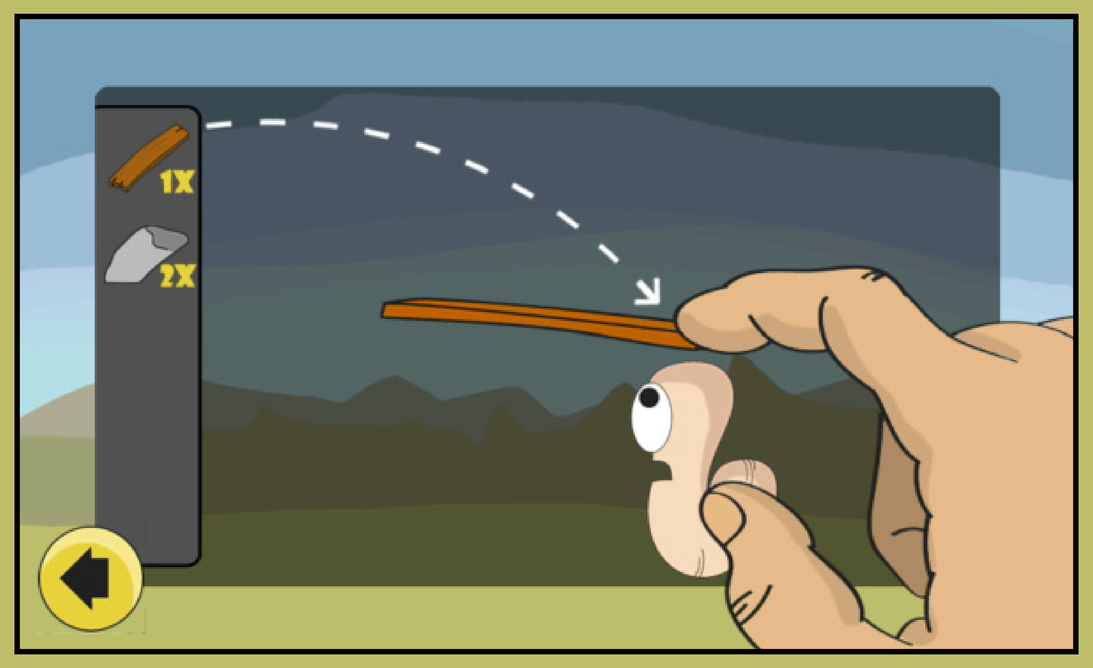


andengine

Box2D

Wilson's Adventures





Spielidee:

- Physik Game
- Drag‘n‘Drop
- Angriff
- Story

Features:

- Scoring-Modell
- Collision Detection
- Erweiterbarkeit

Kollisionsmodell: