Inhaltsverzeichnis

1 Einleitung

Im Zuge eines Projektes im Studiengang Mobile Computing sollte ein Spiel entwickelt werden. In den folgenden Kapiteln werden unter anderem allgemeine einleitende Worte zum Spiel abgegeben. Außerdem werde ich das implementierte Konzept erläutern.

1.1 Zielstellung

Das zu entwickelnde Spiel soll auf mobilen Android Geräten laufen. Dabei soll mit einer Spieleengine gearbeitet werden. Für dieses Projekt wurde sich für LibGDX entschieden. Ein Grund dafür ist zum einen die umfassende Dokumentation, zum anderen gibt es durch die weite Verbreitung viele Einarbeitungsbeispiele. Die Art des Spiels wurde von mir auf Adventure festgelegt. Hauptprinzip des Adventures ist eine zu Grunde liegende Geschichte. Durch Erkundung und das Lösen von Rätseln wird diese Geschichte im Adventure erlebt. Wie für Adventures üblich, ist auch dieses Spiel für Einzelspieler gedacht. Als Subgenre sei Action-Adventure zu nennen, da Shooter-Elemente eingebaut werden sollen. Das Spiel soll in 2D dargestellt werden.

1.2 Story

Das Spiel, welches den ausgeklügelten Namen ProjectMcAdventure trägt, soll auf einem Planeten spielen. Die Story, sowie das Design sind an den bestehenden Klassiker Metroid angelehnt, welchen ich in meiner Kindheit auf dem GameBoy Advance konsumiert habe. Auf diesem Planeten wurde eine unbekannte Lebensform entdeckt, welche von Weltraumpiraten gefangen und gezüchtet werden sollen. Um sie zu stoppen, wurde ein Kämpfer in einem speziellen Anzug zu diesem Planeten geschickt.

1.3 Spielmechanik

Das Ziel des Spieles ist die Tötung der Anführerin der Piraten. Man kann den Kämpfer mit Hilfe von Steuerelementen, ähnlich des GBA's steuern. Die Ansicht ist von der Seite. Man kann also nach links und rechts und nach oben springen und nach

unten fallen. Das Spiel an sich ist in verschiedene Räume aufgeteilt. Der Spieler kann sich von einem Raum in den nächsten Raum durch Tore/Portale bewegen. Diese Portale muss er mit der entsprechenden Waffenversion aufschießen. Hat er die falsche Version, geht die Tür nicht auf. Im Laufe des Spieles soll es dem Spieler möglich gemacht werden, stärkere Waffenversionen freizuschalten um neue Wege durch vorher verschlossene Türen erkunden zu können.

1.3.1 Spieler

Der Charakter soll die Möglichkeit haben, wärend des Spielens zu speichern. Wenn er stirbt, gelangt er an den zuletzt gespeicherten Punkt. Der Charakter hat auserdem verschiedene Ressourcen zu Verfügung. Dazu zählen Gesundheit und Raketen. Man startet mit 0 Raketen und 100 Gesundheit. Nimmt der Spieler im Laufe des Spieles Gesundheitstränke auf erhöht sich seine maximale Gesundheit um 100. Würde der Charakter sterben und man hätte nach Erhalt des Trankes gespeichert, würde er mit 200 Gesundheit starten. Um seine Gesundheit wieder aufzufüllen, muss der Spieler Feinde töten. Wird ein Feind getötet, droppt er ein Item, welche die Gesundheit auffüllt, aber nicht erhöht. Der Spieler kann sowohl normale Kugeln, als auch Raketen abfeuern. Der Sinn einer Rakete ist der höhere Schaden und das Potenzial brüchige Mauern kaputt zu machen, durch die, unter anderem, neue Räume entdeckt werden können.

1.3.2 Feinde

Feinde sollen im Spiel in allen Variationen auftreten können. Sie können fliegen oder sind fest mit dem Boden verbunden. Untereinander können Feinde kollidieren, verletzen sich aber nicht selbst. Wird der Spieler von einem Feind getroffen, verliert er eine bestimmte Anzahl an Gesundheit. Weiterhin wird der Spieler ab diesem Zeitpunkt für 3 Sekunden unverletzlich, damit er sich aus der gefährlichen Situation befreien kann. Feinde können ausschließlich durch Kugeln oder Raketen sterben.

2 Konzept

Das Spiel wurde versucht im Model View Controller zu erstellen. Im folgenden wird das Muster in Bezug auf unser Projekt vergestellt.

2.1 MVC und Projektstruktur

Beim Beginn des Spieles startet LibGDX den Controller, welcher der View sagt, das der GameScreen gerendert werden soll. Der Controller nimmt die Informationen aus den Modellen und übergibt sie der View (Was soll wo im Screen gezeichnet werden?). Ab da aktualisiert dann unser Controller das Modell für jeden Frame mit der abgelaufenen Zeit und einer abstrakten Darstellung der Eingaben des Spielers. Das Modell würde die Position vom Charakter und aller anderen Objekte basierend auf diesen Informationen aktualisieren. Abbildung 1 zeigt eine Übersicht aller wichtigen Klassen.

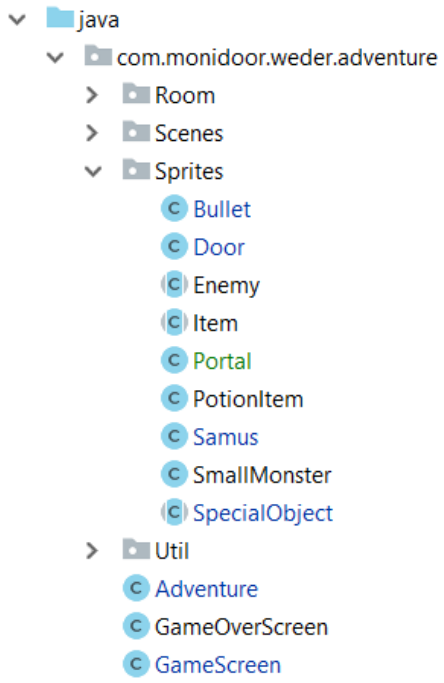

Abbildung 1: Klassendarstellung im Explorer

Adventure erbt von der Klasse Game und enthält den sogenannten SpriteBatch, welcher über alle zu zeichnenden Elemente verfügt. Damit kann die Klasse Adventure als View angesehen werden. GameScreen und GameOverScreen bilden den Controller. Dabei wird vom Interface Screen geerbt. Hier wird entschieden welche Objekte wann und auf welche weise gerendert werden. Die zu rendernten Objekte findet man im Paket Sprites. Jedes Model erbt hier von der Klasse Sprite. Alle zu verwendeten Ressourcen werden im Ordner Assets im Android-Abteil des LibGDX Projektes gespeichert.
Die View wird innerhalb der Screenklassen zusammengesetzt. Die Informationen für die Map befindet sich in einem .tmx File, welches mit dem Tiled Map Editor erstellt wurde. Die Map des Spieles ist eine sogenannte Tiled-Based Map. Das bedeuted, dass das Spielfeld aus kleinen (16x16 in diesem Fall) wiederkehrenden Kachelstrukturen zusammengesetzt ist. Um die dem Spieler zur Verfügung stehenden Ressourcen anzuzeigen, wurde ein sogennnter Hud geschrieben, welcher alle Informationen tabellarisch anzeigt (momentan nur Gesundheit). Damit der Nutzer des Spieles seinen Charakter steuern kann, wurde ein Controller eingeführt, welcher verschiedene Tasten auf den Bildschirm darstellt und gegebenenfalls Tastendrücke abfängt um sie auszuwerten. Dabei sind die Pfeiltasten relativ selbsterklärend. Mit A springt man und mit B kann man schißen. Diese Klasse und die Klasse für den Hud befinden sich im Paket Scenes.

2.2 Laden der Map

Der Tiled Map Editor bietet einem die Möglichkeit, eine Map optisch zu gestalten. Weiterhin kann man auf der Karte Objekte definieren. Je nach Objektebene werden in der Klasse WorldCreator die Objekte evaluiert und im Box2D Raum dargestellt. Ein Raum besitzt dabei physikalische Eigenschaften und eine Menge an Körpern. Zu jedem Objekt in der .tmx wird ein neuer Körper in der Welt erstellt, welcher von unseren beiden Controllern zum rendern freigegeben werden kann. Als Beispiel nehme man alle Objekte der Ebene Door. Wird durch den WorldCreator die Ebene Door abgearbeitet, wird für jedes gefundene Rechteck ein neues Objekt Door angelegt (Siehe: Abbildung 1). Innerhalb der Klasse Door, werden die Eigenschaften des Körpers definiert. Jede Modellklasse besitzt eine update und eine render Methode. Alle erstellten Objekte vom Typ Door können nun pro Zeiteinheit in den Controllern zum rendern, bzw aktualisieren aufgerufen werden.

2.3 Kollionshandling

Bei diesem Spiel ist das Erkennen von Kollisionen und das damit verbundene Ausführen spezieller Funktionen für die kollidierenden Objekte sehr wichtig. Zu diesem Zweck stellt LibGDX die Möglichkeit bereit Flaggen auf die erzeugten Körper zu setzen. Alle existierenden Flaggen sind in Adventure.class implementiert. Unser erstellten Körper in der Klasse Door würden die Flagge DOOR-BIT erhalten. Die Flagge wird CategoryBit genannt. Alle anderen Flaggen unter MaskBits bestimmen die Kollidierbarkeit. Würde man dort ein BULLET-BIT festlegen, würde jedes Mal, wenn der Körper einer Kugel eine Tür berührt, eine Kollidierevent aufgerufen werden. Voraussetzung ist wiederum, das der Körper einer Kugel mit dem entsprechenden CategoryBit belegt wurde. Zum Abfangen der Kollisionsevents dient die Klasse WorldContactListener, welche ein Interface ContactListener implementiert. Mit einem switch-case Statement können nun einfach alle Kollisionen abgefragt werden.

3 Vorhandene Funktionen

Der Spieler kann mit Objekten in seiner Umgebung kollidieren. Feinde können ebenfalls mit allen Objekten kollidieren. Feinde bewegen sich selbstständig und verschwinden, sobald eine Kugel sie berührt. Wird der Spieler vom Feind getroffen, erhält dieser ein Knock-Back und 3 Sekunden Schadensimmunität. Außerdem verliert der Spieler pro Treffer Lebenspunkte. Die Lebenspunkte werden in einem Hud angezeigt. Es wurden Objekte zum Eruegen von Items verfasst. Bei Kollision mit einem Gesundheitstrank, gewinnt der Spieler Lebenspunkte. Der Spieler ist in der Lage Türen für Portalen aufzuschießen. Sinken die Lebenspunkte des Spielers unter 0, stirbt er und es erscheint ein Game Over Screen. Bei einmaligem Druck auf den Screen, startet das Spiel erneut. Additiv sei zu erwähnen, dass für Item, Feind und Charakter Sprites eingefügt wurden. Die letzetn beiden Erwähnungen haben auserdem eine Laufanimation.

4 Fehlende Funktionen

Feinde haben keine Gesundheit. Deshalb können sie mit nur einem Schuss getötet werden, was das Spiel zu einfach machen würde. Die Anordnung der Räume wurde implementiert, jedoch ist es nicht möglich ein Portal zu betreten ohne das das Spiel abschmiert. Das könnte unter anderem daran liegen, wie die Welt erzeugt wird. Ein Raum muss komplett neu gezeichnet werden, inklusive Feinde und Charakter. Eine Lösung habe ich noch nicht funktionstüchtig machen können. Eine Option zum Speichern gibt es auch noch nicht.