

Developerhandbuch - Battletanks

Armin Bernstetter, Stefan Ernst, Nicolas Fella

21. Dezember 2016

Inhaltsverzeichnis

| | |
|---|----------|
| 1 Pflichtenheft | 2 |
| 1.1 Einleitung/Aufgabenstellung | 2 |
| 1.2 Zielbestimmung | 2 |
| 1.2.1 Musskriterien | 2 |
| 1.2.2 Abgrenzungskriterien | 3 |
| 1.2.3 Kann-Kriterien | 3 |
| 1.3 Einsatz | 3 |
| 1.3.1 Anwendungsbereiche | 3 |
| 1.3.2 Zielgruppen | 3 |
| 1.4 Umgebung | 3 |
| 1.4.1 Software | 3 |
| 1.4.2 Hardware | 3 |
| 1.5 Funktionalität | 3 |
| 1.6 Benutzeroberfläche | 4 |
| 1.7 Qualitätsziele | 5 |
| 2 Dokumentation | 6 |
| 2.1 Projektaufbau | 6 |
| 2.1.1 LlbGDX | 6 |
| 2.1.2 Einrichten des Projekts | 6 |
| 2.1.3 Projektstruktur | 6 |
| 2.2 Code | 7 |
| 2.2.1 Klasse Battletanks | 7 |
| 2.2.2 Package Screens | 8 |
| 2.2.3 Package Entity | 11 |
| 2.2.4 Package Utility | 12 |
| 2.3 Dateien/Assets | 13 |
| 2.3.1 Die Maps | 13 |
| 2.3.2 Textures/Grafiken | 13 |
| 2.3.3 Skin/Fonts | 13 |
| 2.3.4 Sounds | 13 |
| 2.4 Tests | 14 |

Kapitel 1

Pflichtenheft

1.1 Einleitung/Aufgabenstellung

Mit dem Projekt 'Battletanks' soll ein Desktop-basiertes Multiplayerspiel erstellt werden. In diesem Spiel können bis zu vier Spieler mit je einer Spielfigur in einem Top Down 2D Spielfeld gegeneinander antreten, indem sie gegnerische Spielfiguren abschießen. Das Spielfeld kann aus einer externen Datei eingelesen werden.

Das Programm wird für Windows, Linux und Mac OS in Java entwickelt unter Einbezug des Frameworks libGDX und damit auch des Build-Management-Automatisierungs-Tools Gradle. Die Benutzeroberfläche des Spiels soll hierbei übersichtlich gestaltet werden, sodass auch Benutzer ohne jegliches Vorwissen 'Battletanks' spielen können.

1.2 Zielbestimmung

1.2.1 Musskriterien

Battletanks Game

- Lokales Versus-Multiplayer Spiel
- Bis zu vier Spieler zur selben Zeit
- Unabhängig von CPU Geschwindigkeit
- Graphische Benutzeroberfläche

User

- Muss ein Spiel starten können
- Kann Spielfigur im Menü auswählen
- Hat eine von vier verschiedenen Tastenbelegungen
- Kann eine Arena laden
- Kann eine Spieldauer festlegen

GUI

Menü

- Per Maus bedienbar
- Spiel starten
- Spielfigurauswahl mit Informationen über die Figuren
- Arena laden
- Eingabe der Spieldauer

Spiel

- Enthält Spielfeld und Spielfiguren (Über Tastatur steuerbar)
- Top-Down 2D Grafik
- Zeitanzeige

- Anzeige von Informationen über die Spielfiguren (Leben, Anzahl der Abschüsse)

Spielfigur

- Hat eindeutiges Vorne und Hinten
- Kann sich in 8 Richtungen bewegen (45° Drehung möglich)
- Festgelegte Lebenspunkte
- Besitzt Waffe (Schussfrequenz, Schaden)
- Schadensreduzierung
- Kann Waffe nach vorne abfeuern
- Bekommt Lebenspunkteabzug falls von gegnerischer Spielfigur getroffen
Abgezogene Lebenspunkte berechnen sich aus Schaden der gegn. Waffe und der Schadensreduzierung

Arena/Spielfeldgeometrie

- Spielfiguren bewegen sich auf Spielfeld
- Information über Spielfeld wird aus externer Datei eingelesen
- Spielfeld enthält Hindernisse

1.2.2 Abgrenzungskriterien

- Keine Netzwerkverbindung benötigt
- Spiel läuft auf Windows, Linux und Mac OS
- Keine 3D Grafik
- Maximal vier Spieler. Einem Spieler ist genau eine Tastenbelegung zugeordnet.

1.2.3 Kann-Kriterien

- Upgrades (Speed-, Damageboost; Streuende Waffen usw)

1.3 Einsatz

1.3.1 Anwendungsbereiche

Das Spiel bietet eine Freizeitbeschäftigung, die wenig bis kein Vorwissen und einen benutzerdefinierten Zeitaufwand benötigt.

1.3.2 Zielgruppen

Personen, die interessiert sind an einem unterhaltsamen und kompetitiven Computerspiel, in dem sie gegen 1-3 Mitspieler antreten können.

1.4 Umgebung

1.4.1 Software

- Betriebssystem: Windows, Linux oder Mac
- Java 8

1.4.2 Hardware

Beliebiger Computer

1.5 Funktionalität

Typische Anwendung: Das Spiel wird gestartet. Im Menü können die Spieler ihre Spielfiguren auswählen, die Arena aus einer externen Datei einlesen und eine Spieldauer eingeben. Anschließend treten die Spieler gegeneinander an. Die Spieler bewegen ihre Figur mithilfe einer festen Tastenbelegung bestehend aus

vier Tasten über ein Spielfeld. Durch Drücken einer zusätzlichen Taste kann die Spielfigur feuern und gegnerische Spielfiguren abschießen, die sich im Schussfeld befinden. Falls die Lebenspunkte einer Spielfigur aufgebraucht sind, wird diese in einer festgelegten/zufälligen (?) Position 'wiederbelebt' und der jeweilige Spieler steigt wieder ins Spiel ein. Der Spieler, der nach Ablauf der Zeit die meisten Abschüsse hat, gewinnt das Spiel.

1.6 Benutzeroberfläche

Benutzeroberfläche bedienbar durch Maus- und Tastatur. Menü zur Auswahl von Spielfiguren, Spieldauer und Arena. Spielfeld zum Ausführen des Spiels.

Menu Screen Entwurf:

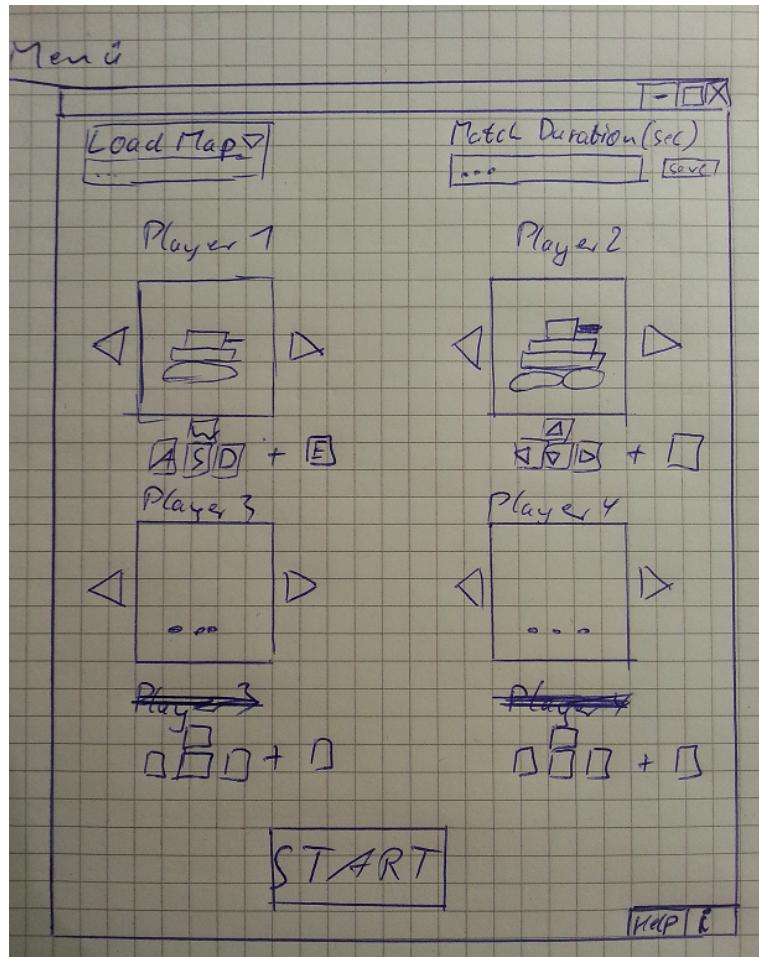


Abbildung 1.1: Spielfigur aus Vorschau aller verfügbaren Figuren auswählen. Information über jeweilige Spielfigur verfügbar. Tastenbelegung unter Vorschau angezeigt. Default: Keine Figur ausgewählt, Anzahl Spieler bestimmt durch Anzahl ausgewählter Figuren. Match Duration in Sekunden eingeben

Game Screen Entwurf:

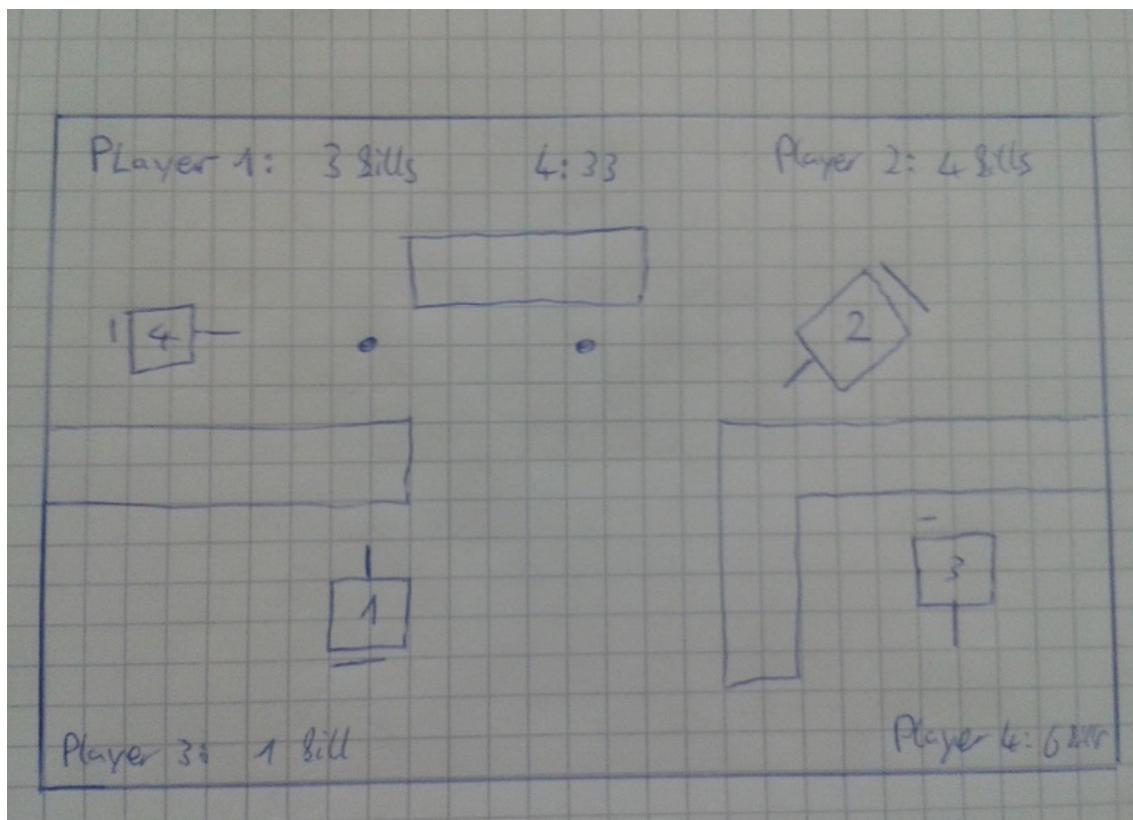


Abbildung 1.2: Anzeige der verbleibenden Spielzeit mittig am oberen Spielfeldrand; Anzeige der Abschüsse und Tode der Spieler in den Ecken; Hindernisse, die nicht durchfahren werden können; Spielfiguren feuern Projektil

1.7 Qualitätsziele

- Spiel läuft flüssig und von der CPU Geschwindigkeit unabhängig
- Einfache Bedienung durch den Benutzer
- Figuren leicht steuerbar

Kapitel 2

Dokumentation

2.1 Projektaufbau

2.1.1 LibGDX

libGDX (<http://libgdx.badlogicgames.com/>) ist ein cross-platform Java-Framework für Spieleentwicklung. Es unterstützt Windows, Linux, Mac OS X, Android, Blackberry, iOS und HTML5. Es wurde von **Mario Zechner** entwickelt. Es erlaubt dem Programmierer, seinen Code einmal zu schreiben und anschließend auf verschiedenen Plattformen anzuwenden. LibGDX bietet direkten Zugang zum Dateisystem, Eingabe- und Audiogeräten sowie OpenGL durch ein kombiniertes OpenGL ES 2.0 und 3.0 interface. Zusätzlich bietet LibGDX APIs für viele Aufgaben in der Spielentwicklung wie das Rendern von Sprites und Text, UI Erstellung, Soundeffekte und vieles Weiteres.

libGDX benutzt Gradle, ein Build-Management-Automatisierungs-Tool

2.1.2 Einrichten des Projekts

Um ein libGDX Projekt zu erstellen wird die GDX-Setup App benötigt, die auf der Seite von libGDX zum Download zur Verfügung steht. Anschließend kann man festlegen, für welche Plattformen man entwickeln will und welche Extensions (wie z.B. um Freetype Fonts einlesen/benutzen zu können) man verwenden möchte. Diese können auch nachträglich noch geändert werden durch Änderungen an den build.gradle Dateien. Im Fall von Battletanks wurde sich auf eine Desktopanwendung beschränkt und die Extensions 'Tools' und 'Freetype' verwendet.

Die Entwicklung geschah größtenteils mit der IDE Eclipse und Java 8, zur Version Control wurde ein Repository auf dem von der Universität Würzburg bereitgestellten Repository Manager <https://gitlab2.informatik.uni-wuerzburg.de/> benutzt.

2.1.3 Ordnerstruktur

Battletanks_SWP Dieser Projektordner beinhaltet die von Gradle verwendete Wrapper und Dateien, einen Ordner textures mit allen verwendeten Grafiken, runnable jar Dateien für das Programm Hiero <https://github.com/libgdx/libgdx/wiki/Hiero> und einen Texture Packer um die Grafiken in einem Texture Atlas zusammenzufassen.

Zusätzlich befindet sich hier noch ein Ordner mit Benutzer- und Developerhandbuch.

Battletanks-desktop Das desktop Projekt beinhaltet nur eine Klasse DesktopLauncher, die das Spiel als 'Lightweight Java Game Library'-Applikation startet, die eine neue Instanz von BattleTanks übergeben bekommt.

Damit der DesktopLauncher auf die richtigen Assets zugreifen kann musste vor der ersten Ausführung das Working Directory in den Run Configurations zum Ordner 'Battletanks-core/assets' geändert werden.

Battletanks-core Dieser Ordner beinhaltet den gesamten Code des Projekts (siehe 2.2 Code), Java Docs sowie alle verwendeten Assets. Darunter befinden sich Soundeffekte, Fonts, die verwendeten Tiled-Maps, ein UI Skin, Fonts für Menü und Game sowie die Grafiken der Panzer, der Hintergrund und weitere Dateien.

2.2 Code

Architekturdiagramm:

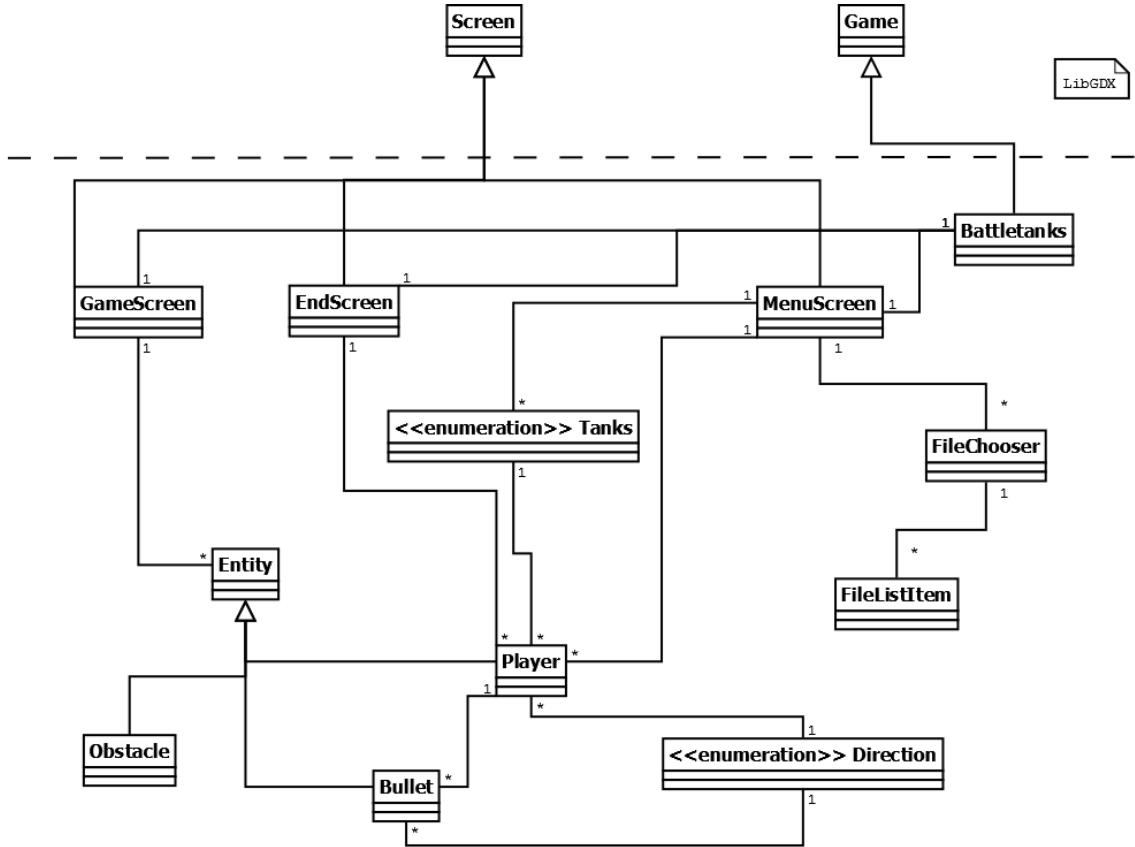


Abbildung 2.1: test

2.2.1 Klasse Battletanks

Die Klasse BattleTanks ist die Hauptklasse. Sie erbt von der LibGDX Klasse Game und verwaltet die Screens, den TextureAtlas sowie die Preferences. Die Methoden showMenu, showGame und showEnd setzen den vom Spiel angezeigten Screen.

2.2.2 Package Screens

Alle Klassen in diesem Package implementieren das LibGDX interface Screen. Jeder Screen besitzt eine Methode reset, um z.B. beim mehrmaligen Start neuer Spiele nicht jedes Mal neue Screen Instanzen erstellen zu müssen.

MenuScreen

Screenshot Menü:



Abbildung 2.2: test

Der MenuScreen besitzt zwei Stages, auf denen das Menü dargestellt wird. Die Hintergrundstage beinhaltet nur das Hintergrundbild, welches an die Fenstergröße angepasst und bei Bedarf gestreckt/gestaucht wird. Das gesamte Menü wird durch die zweite Stage dargestellt, die eine Table mainTable beinhaltet, auf der sich alle UI Elemente befinden. Diese werden jeweils von Methoden erstellt, die nach dem Schema 'createUIElement()' benannt sind und werden anschließend von der Methode 'create()' auf der mainTable angeordnet.

Zur Darstellung aller UI Elemente wird der im Ordner assets/data befindliche Skin uiskin.json verwendet. (Siehe 2.4.3 Skin/Fonts)

Wurden alle notwendigen Informationen vom Benutzer eingegeben, wird durch Aufruf der showGame() Methode der Klasse BattleTanks das Spiel gestartet indem zu einem GameScreen gewechselt wird.

GameScreen

Screenshot Game:



Abbildung 2.3: test

Im GameScreen befindet sich die gesamte Spiellogik. Der Konstruktor bekommt eine Liste von Spielern, die Spieldauer und den Dateipfad der ausgewählten Map übergeben. Letzterer wird in der Methode loadMap verwendet, um die Map aus der .tmx Datei einzulesen (Siehe 2.3.1 Die Maps).

Die Methode render wird jeden Frame aufgerufen und aktualisiert das gesamte Spiel, also alle Entities, die sich auf dem Spielfeld befinden, alle Textanzeigen und die Map.

Kollisionserkennung Grundsätzlich können sich die Player jeden Frame beliebig bewegen. Bevor die neuen Positionen allerdings gezeichnet werden, findet die Kollisionserkennung und -auflösung statt. Zur Erkennung besitzt jede Entity ein Kollisionsrechteck, welches verwendet wird, um zu überprüfen, ob sich zwei Entities überschneiden. Falls dies der Fall ist, wird die Überlappung in x und y Richtung berechnet. Je nachdem, welcher Wert kleiner ist, wird die Kollision in der entsprechenden Richtung wieder aufgelöst. Hierbei dient die Geschwindigkeit der beiden Entities, um den Typ der Kollision festzustellen. Falls eine der beiden Entities steht (Beispiel: Player-Obstacle-Kollisionen), wird die sich bewegende Entity ausgebremst und an den Rand der anderen zurückgesetzt. Ein weiterer Kollisionstyp findet statt, falls beide Entities aufeinander zufahren. Dies wird aufgelöst, indem die Entities jeweils zur Hälfte der Überlappung zurückgesetzt werden. Zuletzt könnte es noch sein, dass die eine Entity langsamer war als die andere, sodass letztere der anderen hinten auffährt. In diesem Fall wird die schnellere ausgebremst und an den Rand der anderen zurückgesetzt. Zusätzlich wird die Geschwindigkeit der versetzten Entities in der entsprechenden Richtung auf 0 gesetzt, sodass sie bei einer weiteren Kollision nicht als Kollisionsverursacher wieder entlang dieser Richtung verschoben werden können.

Damit die Player zuerst von den Wänden weggeschoben werden, findet als Erstes die Kollisionserkennung zwischen Playern und Obstacles und anschließend die zwischen Playern und Playern statt. Die Kollisionen zwischen Bullets und Entities werden zuletzt überprüft und löschen die Bullets (mit Schaden bei Playern).

EndScreen

Screenshot Scoreboard:

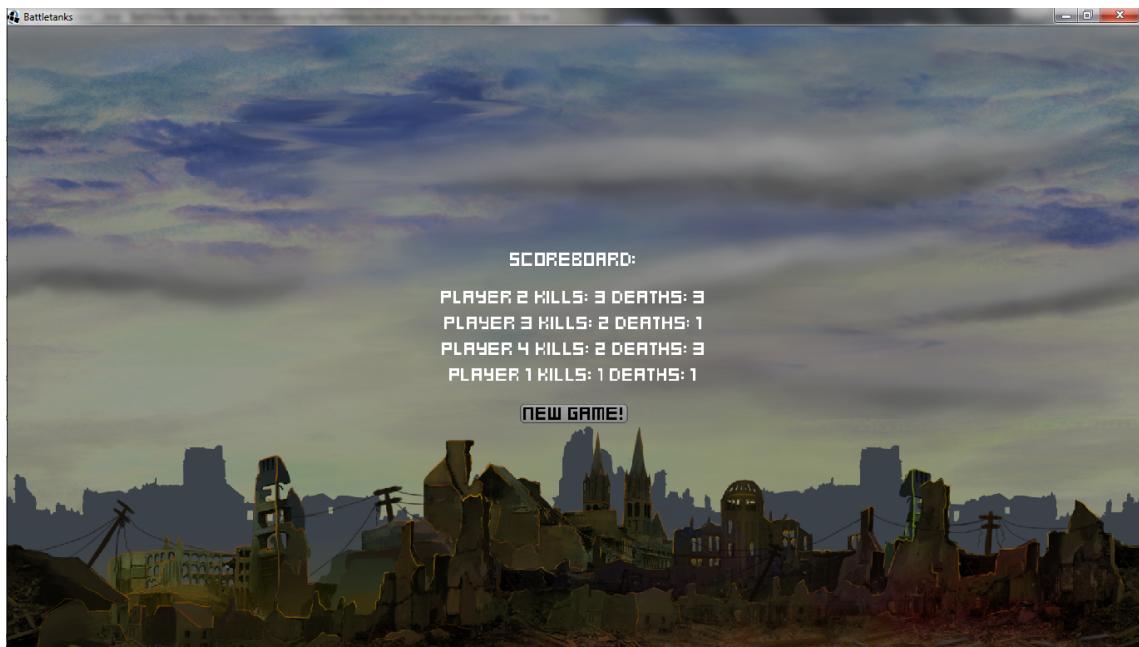


Abbildung 2.4: test

Der EndScreen erscheint sobald die eingebaute Zeit abgelaufen ist. Er ist ähnlich aufgebaut wie der MenuScreen mit main stage und background stage und bekommt eine Liste an Playern übergeben. Diese werden mithilfe des internen PlayerComparator zuerst absteigend nach Kills, aufsteigend nach Deaths und danach aufsteigend nach PlayerNumber sortiert und in Form eines Scoreboards dargestellt. Zudem besitzt EndScreen einen Button, mit dem ein neues Spiel gestartet werden kann.

2.2.3 Package Entity

Entity Die Klasse Entity implementiert das LibGDX Interface Disposable. Dies bedeutet, dass Instanzen dieser Klasse am Ende ihrer Lebenszeit manuell disposed werden müssen, da es sonst zu Memory Leaks kommen kann. Jede Instanz einer Entity besitzt eine Position im Koordinatensystem, eine Höhe und Breite sowie eine 2D Sprite, die aus dem TextureAtlas ausgelesen wird. Zusätzlich hat jede Instanz ein 'collisionRectangle' und eine Koordinate, die zur Kollisionsberechnung verwendet werden.

Player Die Klasse Player erbt von Entity. Sie stellt eine Spielfigur mit einem Tank (siehe Enum Tanks), einer Tastenbelegung, einer Playernumber, einer Bewegungsgeschwindigkeit und einer 2D Sprite 'Gun' dar. Die Player eines Spiels werden bereits im MenuScreen erstellt wenn der 'Lock Tank Choice'-Button gedrückt wird und werden dann an den GameScreen übergeben.

Die Methode update() verarbeitet u.a. mit Aufruf mehrerer Untermethoden die Benutzereingaben und bewegt die Spielfigur bzw erzeugt eine neues Projektil. Zudem kümmert sich diese Methode um den Respawn des Spielers.

Bullet Die Klasse Bullet erbt von Entity. Bullets gehören zu einer Player Instanz. Dies wird benötigt um zu berechnen, welcher Player einen anderen abgeschossen hat.

Obstacle Die Klasse Obstacle erbt ebenfalls von Entity. Sie stellt ein Hindernis auf der Map dar.

Enum Direction Direction ist ein Enum, das die 8 möglichen Richtungen darstellt, in die sich eine Spielfigur bewegen kann.

Enum Tanks Das Enum Tanks erstellt die fünf Spielfiguren. Jede davon besitzt einen Damage-, Armor- und HealthPoints-Wert sowie eine ReloadTime/Schussfrequenz.

2.2.4 Package Utility

Screenshot FileChooser:

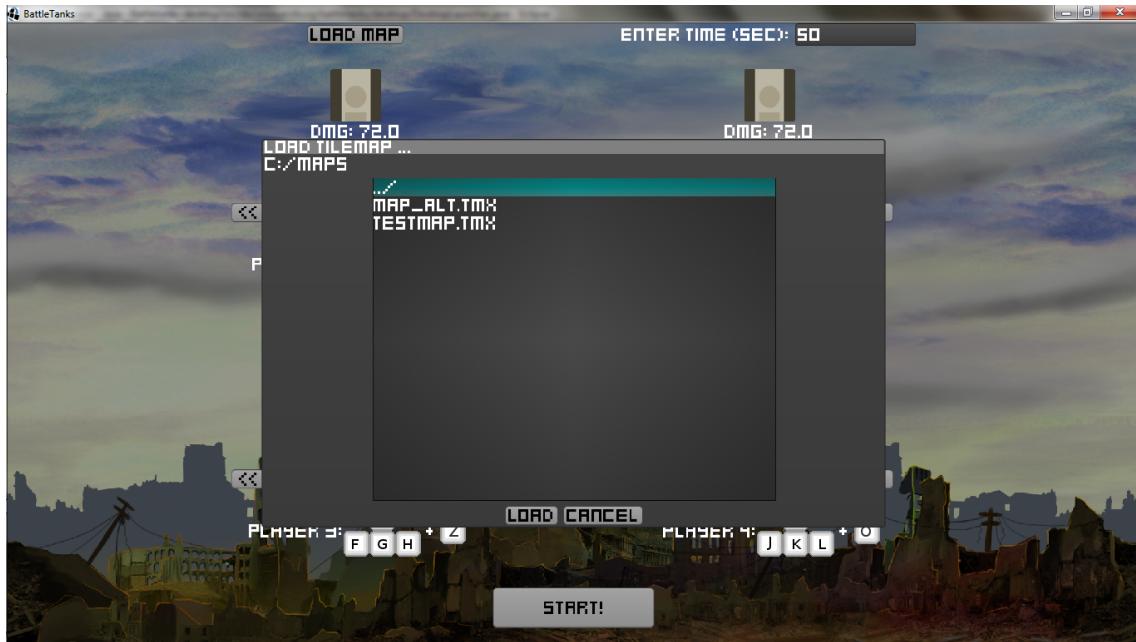


Abbildung 2.5: test

FileChooser Der FileChooser wurde benötigt, da LibGDX kein Fenster zur Auswahl einer Datei enthält und basiert grundlegend diesem Entwurf (<http://www.java-gaming.org/index.php?topic=35471.0>). Er erbt von der Klasse Dialog, welche bereits Methoden zum Hinzufügen von Buttons, Ausgeben eines Resultats und Anzeigen eines Dialogfensters bietet. Der FileChooser enthält Buttons zum Laden und zum Abbrechen der Mapauswahl. Des Weiteren wird dem Dialogfenster eine ScrollPane hinzugefügt, die wiederum aus einer Liste von FileListItems besteht. Die Elemente der Liste entsprechen den Ordner und Dateien wobei letztere nach Dateiendung .tmx gefiltert werden und Ordner vor Dateien angezeigt werden. Beim Klick auf eins der Elemente wird entweder der Ordner gewechselt und dessen Inhalt angezeigt oder die Datei ausgewählt. Zum Erstellen eines FileChoosers wird eine statische Methode verwendet, in der die result Methode so überschrieben wird, dass das Ergebnis der result Methode eines selbst definierten ResultListeners, dessen Interface im FileChooser enthalten ist, übergeben wird. Dieser ResultListener kann außerdem mit einer weiteren Methode gesetzt werden um festzulegen, was beim schließen des Fensters passieren soll. Die ausgewählte Datei kann zum Schluss mit einer weiteren Methode abgefragt werden.

FileListItem Diese Klasse wird benötigt um die Liste von Ordner und Dateien im FileChooser zu realisieren. Ein FileListItem entspricht grundsätzlich einem Ordner bzw. einer Datei, enthält jedoch zusätzlich noch einen Namen, dem bei Ordner noch ein '/' zur Unterscheidung angehängt wird. Die toString Methode wird außerdem so überschrieben, dass hierbei der Name zurückgegeben wird.

2.3 Dateien/Assets

2.3.1 Die Maps

Zum Einlesen von Maps wird eine Datei im TMX Format (<http://doc.mapeditor.org/reference/tmx-map-format/>) benötigt. Diese enthält die erstellte Map mit Verweis auf Grafiken in Form eines Tilesets und kann aus mehreren Layers bestehen. Die Größe der Map ist egal, da das Spiel entsprechend skaliert wird. Zu beachten ist jedoch, dass die Tanks standardmäßig eine Größe von 40x40 Pixel haben und in den Ecken spawnen müssen. Der Rest der Map kann beliebig gestaltet werden. Damit die Obstacles jedoch eingelesen werden können, muss eine Objektebene namens 'objects' existieren, die alle Hindernisse als rechteckige Objekte enthält. Diese werden dann intern in Obstacles übersetzt, während die restlichen Layers nur als Hintergrund dienen. Es bietet sich außerdem an, die Tilemap mithilfe eines Editors, wie zum Beispiel dem Tiled Map Editor (<http://www.mapeditor.org/>) zu erstellen.

2.3.2 Textures/Grafiken

Die meisten der verwendeten Grafiken wurden auf <http://opengameart.org/> gefunden. Der Hintergrund stammt von <https://mobilegamegraphics.com/product/airplane-dogfight-assets-free-assets/>. Sämtliche im Game verwendeten Textures wurden erstellt von <http://kenney.nl/assets/topdown-tanks>. Dies beinhaltet die Textures aus denen die Default-Map besteht, die Bullets sowie Panzer.

2.3.3 Skin/Fonts

Der verwendete Skin sowie alle zugehörigen Dateien befinden sich im Ordner assets/data im core Projekt. Ausgenommen des im Menü und EndScreen verwendeten Fonts pixel.fnt stammen die verwendeten Dateien aus dem LibGDX Tests Repository <https://github.com/libgdx/libgdx/tree/master/tests/gdx-tests-android/assets/data> und werden in der LibGDX Dokumentation als default Dateien empfohlen. Zur näheren Information über die Funktionsweise von Skins siehe <https://github.com/libgdx/libgdx/wiki/Skin>

Der Font pixel.fnt stammt ebenfalls von <http://kenney.nl/assets/kenney-fonts> und wurde mithilfe des Programms Hiero von einem TrueTypeFont in einen BitMapFont umgewandelt.

2.3.4 Sounds

Sowohl der Schuss-Soundeffekt als auch die Hintergrundmusik wurden auf <http://www.freesound.org/> gefunden.

Soundeffekt: <http://www.freesound.org/people/ShawnyBoy/sounds/166191/>

Musik: <http://www.freesound.org/people/Airwolf89/sounds/346455/>

2.4 Tests