

Battletanks - Developerhandbuch

Universität Würzburg Informatik Softwarepraktikum Wintersemester 2016/17

Armin Bernstetter, Stefan Ernst, Nicolas Fella

3. Februar 2017

Inhaltsverzeichnis

1 Pflichtenheft	3
1.1 Einleitung/Aufgabenstellung	3
1.2 Zielbestimmung	3
1.2.1 Musskriterien	3
1.2.2 Abgrenzungskriterien	4
1.2.3 Kann-Kriterien	4
1.3 Einsatz	4
1.3.1 Anwendungsbereiche	4
1.3.2 Zielgruppen	4
1.4 Umgebung	4
1.4.1 Software	4
1.4.2 Hardware	4
1.5 Funktionalität	5
1.6 Benutzeroberfläche	5
1.7 Qualitätsziele	6
2 Dokumentation	7
2.1 Projektaufbau	7
2.1.1 libGDX	7
2.1.2 Einrichten des Projekts	7
2.1.3 Ordnerstruktur	7
2.2 Code	8
2.2.1 BattleTanks	8
2.2.2 Package Screens	9
2.2.3 Package Entity	12
2.2.4 Package Utility	13
2.3 Dateien/Assets	14
2.3.1 Die Maps	14
2.3.2 Textures/Grafiken	14
2.3.3 Skin/Fonts	14
2.3.4 Sounds	14
2.4 Tests	15
2.4.1 JUnit Tests	15
2.4.2 Testprotokoll	15

1 Pflichtenheft

1.1 Einleitung/Aufgabenstellung

Mit dem Projekt 'Battletanks' soll im Rahmen des Softwarepraktikums Informatik der Universität Würzburg im Wintersemester 2016/17 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 geln. 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 '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:

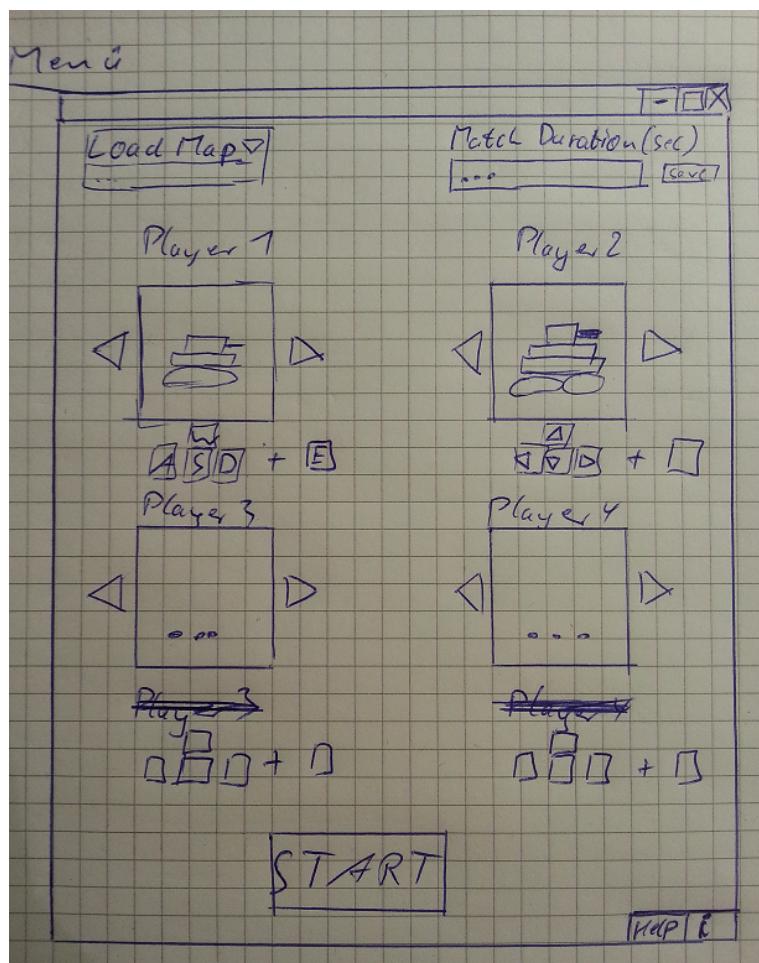


Abbildung 1.1: Spielfigur aus Vorschau aller verfügbaren Figuren auswählen. Information über jeweilige Spielfigur verfügbar. Tastenbelegung unter Vorschau angezeigt. Default: Keine Figur ausgewählt, Anzahl Spieler bestimmt durch Anzahl ausgewählter Figuren. Match Duration in Sekunden eingeben

Game Screen Entwurf:

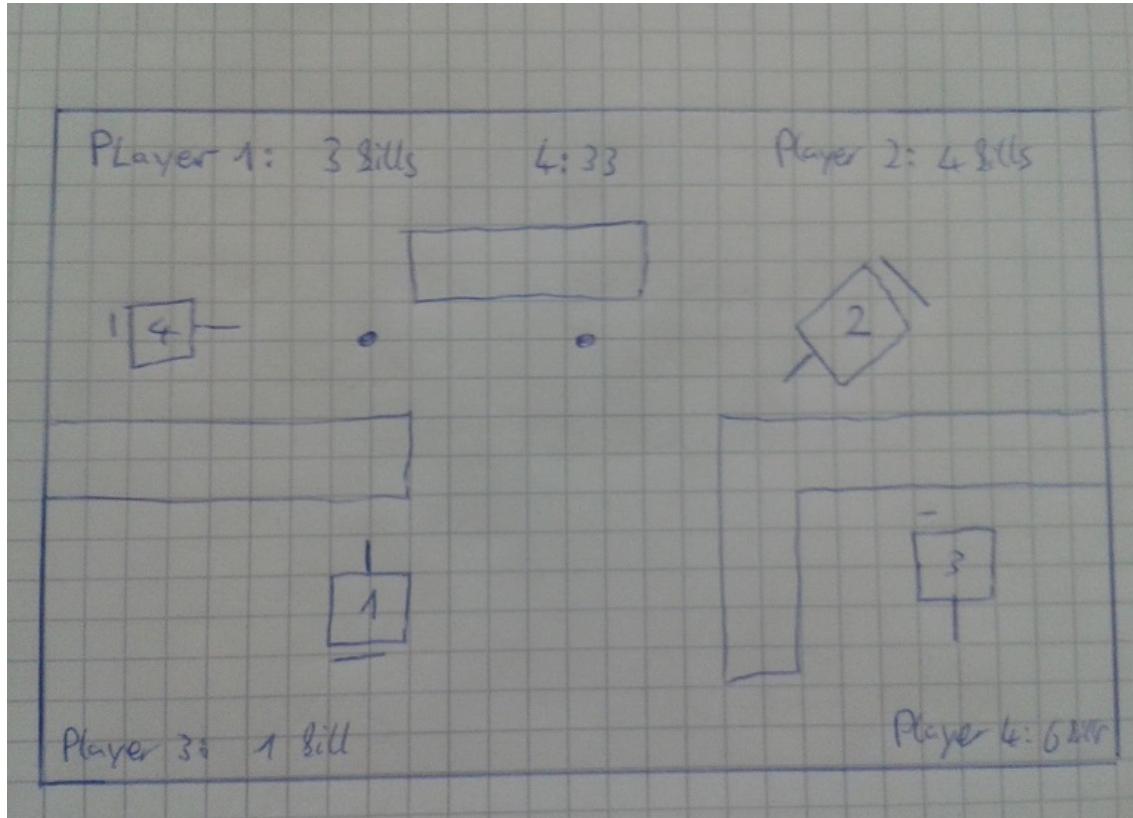


Abbildung 1.2: Anzeige der verbleibenden Spielzeit mittig am oberen Spielfeldrand; Anzeige der Abschüsse und Tode der Spieler in den Ecken; Hindernisse, die nicht durchfahren werden können; Spielfiguren feuern Projektil

1.7 Qualitätsziele

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

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 (<https://twitter.com/badlogicgames>) entwickelt und erlaubt dem Programmierer, seinen Code einmal zu schreiben und anschließend auf verschiedenen Plattformen anzuwenden. LibGDX bietet direkten Zugang zum Dateisystem, zu Eingabe- und Audiogeräten sowie zu 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 einzulesen/benutzen zu können) man verwenden möchte. Diese können auch nachträglich noch geändert werden durch Änderungen an den build.gradle Dateien. Siehe hierzu die libGDX Wiki (<https://github.com/libgdx/libgdx/wiki>). 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 verwendeten Wrapper- und sonstigen Dateien, einen Ordner 'textures' mit allen verwendeten Grafiken, runnable jar Dateien für das Programm Hiero <https://github.com/libgdx/libgdx/wiki/Hiero> zur Umwandlung von TrueTypeFonts in BitMapFonts 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 TiledMaps, ein UI Skin, Fonts für Menü und Game sowie die Grafiken der Panzer, der Hintergrund und weitere Dateien.

2.2 Code

Architekturdiagramm:

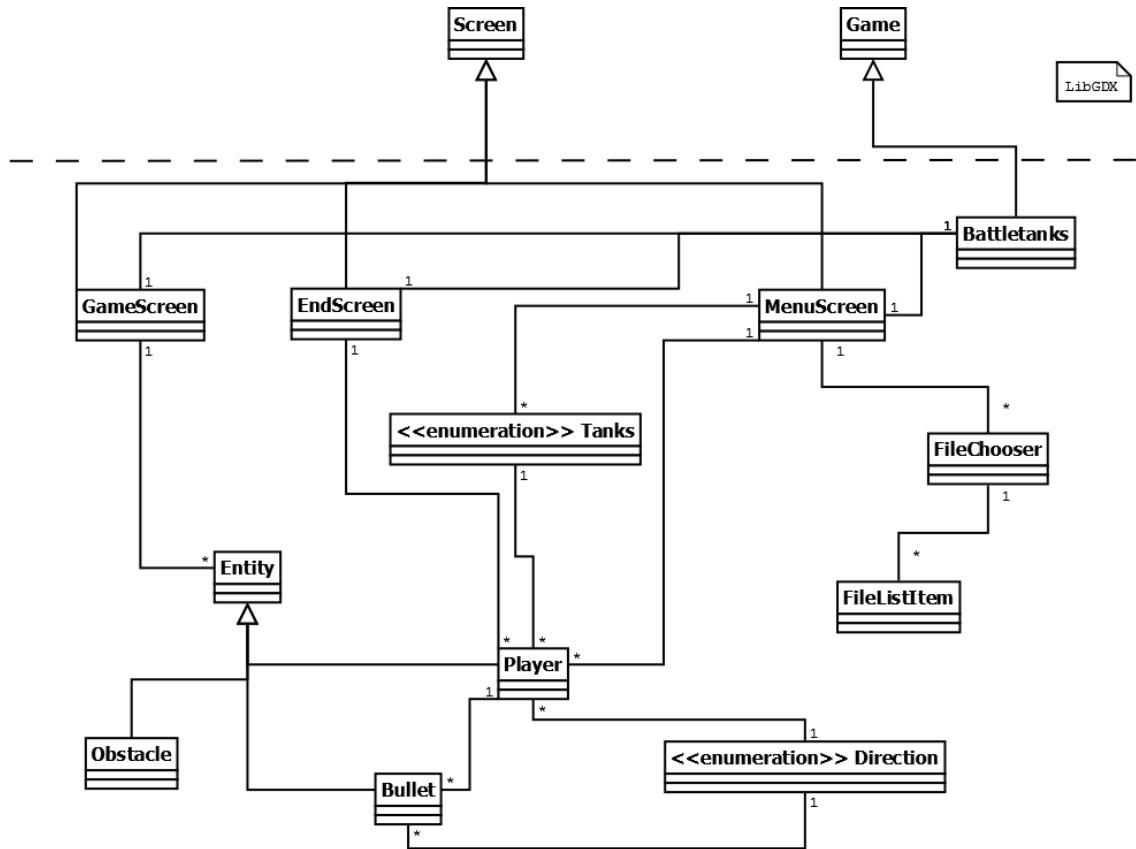


Abbildung 2.1: Übersicht über die Klassenstruktur des Projekts

2.2.1 BattleTanks

Die Klasse **BattleTanks** ist die Hauptklasse. Sie erbt von der LibGDX Klasse **Game** und verwaltet die Screens, den TextureAtlas sowie die Preferences/(Vor-)Einstellungen. Die Methoden **showMenu**, **showGame** und **showEnd** setzen den von der **BattleTanks** Instanz 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: 1: Map Laden, 2: Zeit Eingeben, 3: Spielfigur auswählen, 4: Tastenbelegung des jew. Spielers, 5: Spiel starten

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 2.3.3 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: 1: Anzeige von Abschüssen und Toden, 2: Verbleibende Spielzeit, 3: Panzer mit Kanone nach unten ausgerichtet, 4: Projektil des blauen Panzers

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 in jedem 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 dazu, 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:

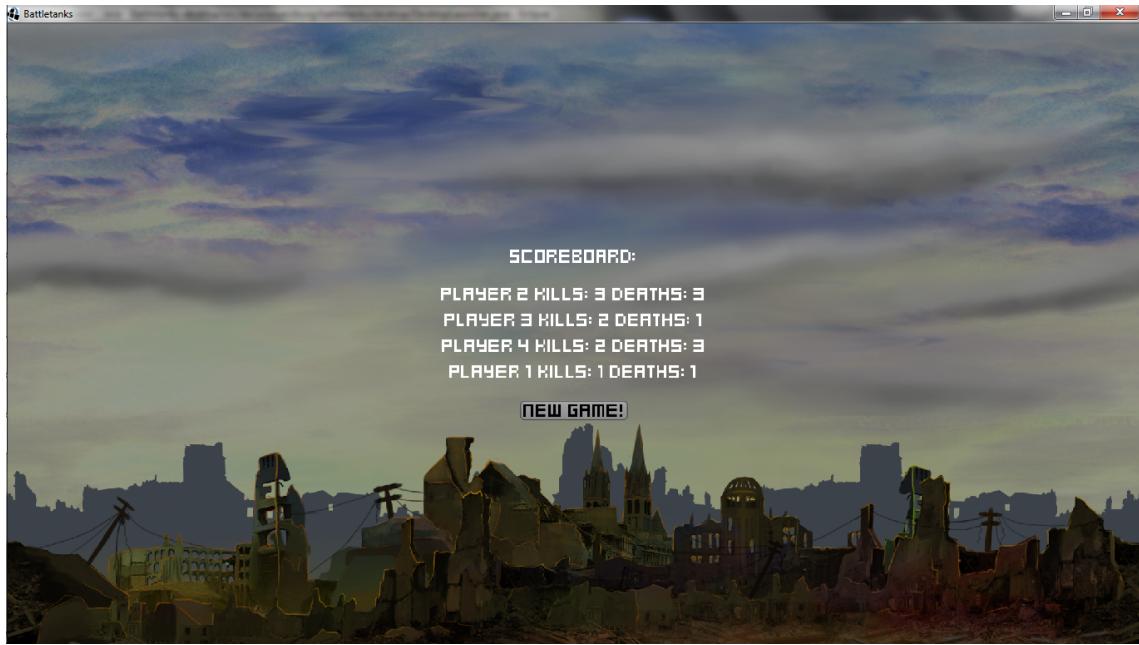


Abbildung 2.4: Spielende und Starten eines neuen Spiels

Der **EndScreen** erscheint sobald die eingegebene Zeit abgelaufen ist. Er ist ähnlich aufgebaut wie der **MenuScreen** mit einer 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.

ErrorScreen

Der **ErrorScreen** erscheint zum aktuellen Stand des Programmes nur, wenn eine Map geladen wurde, auf der die Spielfiguren unmittelbar auf Hindernissen spawnen. Er ist an das Design des **EndScreen** angelehnt und zeigt nur eine Fehlermeldung, die beim Aufruf des Konstruktors übergeben werden kann sowie einen 'New Game' Button.

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 Spielernummer, 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 ein 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:

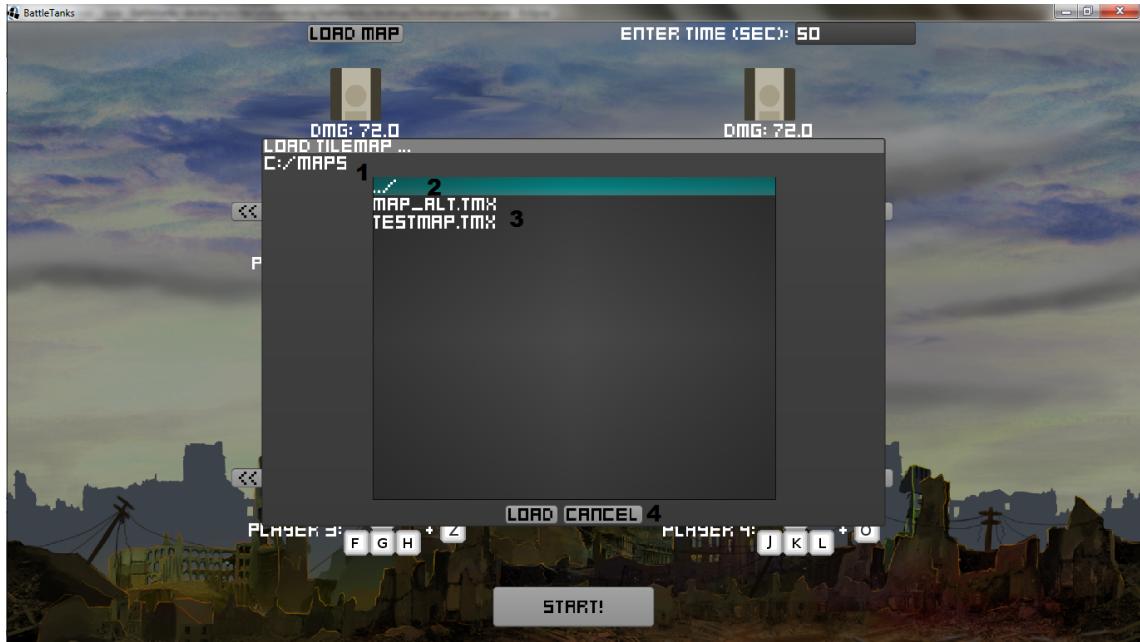


Abbildung 2.5: 1: Anzeige des aktuellen Ordnerpfades, 2: Wechseln in den vorherigen Ordner, 3: Die zur Auswahl stehenden Maps, 4: Ausgewählte Map laden oder Abbrechen

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 Ordnern und Dateien wobei letztere nach Dateiendung .tmx gefiltert werden und Ordner vor Dateien angezeigt werden. Beim Klick auf eines der Elemente wird entweder der Ordner gewechselt und dessen Inhalt angezeigt oder die Datei ausgewählt. Zum Erstellen eines **FileChooser**s 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 Ordnern 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

Der Großteil der verwendeten Grafiken wurde auf <http://opengameart.org/> gefunden. Das Hintergrundbild stammt von <https://mobilegamegraphics.com/product/airplane-dogfight-assets-free-assets/> und 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 in Form einer .json Datei 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 zur Verwendung 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 standardmäßig auskommentierte 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

2.4.1 JUnit Tests

Das Package `de.uniwaerzburg.battletanks.test` enthält Unit-Tests für die Klassen aus dem Package `Entity` (Siehe 2.2.3). Hierfür wird JUnit4 eingesetzt. Um die Unit-Tests zu ermöglichen werden einzelne Funktionen von libGDX von den Klassen in `de.uniwaerzburg.battletanks.test.mock` imitiert. Hierfür ist es wichtig, dass zu Beginn der Test eine Instanz von `MockBattleTanks` erstellt wird. Dies wird beim ausführen der Test-Suite `AllTests` getan. Beim Ausführen einzelner Tests muss dies manuell geschehen. Die anderen Komponenten von Battletanks lassen sich aufgrund der komplexen Zusammenhänge von graphischer Oberfläche, OpenGL, Spielmechanik, libGDX und Benutzereingaben nicht mit Unit-Tests testen. Hierfür existiert ein Test-Protokoll.

2.4.2 Testprotokoll

Da die JUnit Tests viele Bereiche der GUI nicht abdecken können, soll das folgende Protokoll die restlichen Tests enthalten. In diesem Testprotokoll werden verschiedene mögliche Fehlerquellen angegeben und deren Überprüfung beschrieben.

MenuScreen

Fehlerquelle: Bei der Eingabe einer Spielzeit könnten Probleme entstehen, wenn der Nutzer zum Beispiel negative Zahlen oder Buchstaben eingibt.

Test: Im Test wurde die Zeit auf 'ABC' gesetzt, Spieler ausgewählt und das Spiel gestartet. Daraufhin öffnete sich ein Fenster mit dem Hinweis 'NO VALID TIME ENTERED!'. Mit dem Zeitwert '-60' startete das Spiel jedoch und der Betrag der Zahl wurde als Spielzeit gewählt.

Fehlerquelle: Beim Einlesen von Maps könnte häufiges Wechseln der Map die alte Map nicht überschreiben oder falsche Namen anzeigen.

Test: Im Test wurde eine Map geladen, deren Name anschließend neben dem 'LOAD MAP' Button erschien. Als erneut eine Map ausgewählt wurde, änderte sich auch der Schriftzug und beim Spielstart wurde letztere auch richtig geladen.

Fehlerquelle: Eine fehlerhafte Map könnte das Spiel zum Absturz bringen.

Test: Im Test wurde deshalb eine leere Datei 'map.tmx' erstellt und anschließend geladen. Nach Auswahl der Spieler und Starten des Spiels traten keinerlei Fehler auf und die Standard-Map wurde geladen.

Fehlerquelle: Bei der Auswahl der Tanks könnten die Buttons falsch zugewiesen worden sein oder Spielern nicht alle Tanks zur Verfügung stehen.

Test: Aus diesem Grund wurden im Test bei jedem Spieler alle Tanks rückwärts und vorwärts durchgeschaltet. Dies funktionierte ohne Probleme und bei jedem Spieler waren alle Tanks verfügbar.

Fehlerquelle: Ein weiteres Problem könnte das Tankwechseln nach dem betätigen des Lock-Buttons sein.

Test: Hierbei wurde für jeden Spieler ein Tank 'gelandet' und dann versucht, den Tank in beide Richtungen zu ändern. Das Ergebnis zeigte, dass die Lock-Buttons richtig implementiert wurden.

Fehlerquelle: Außerdem könnte es beim Spielstart passieren, dass ein Spieler den falschen Tank erhält.

Test: Dies wurde getestet indem jeder Spieler einen anderen Tank erhielt und alle gelockt wurden. Beim Spielstart zeigte sich, dass jeder Spieler auch den richtigen Tank steuerte.

Fehlerquelle: Des Weiteren musste getestet werden, was bei einem Spielstart mit zu wenig oder fehlerhaften Einstellungen passiert.

Test: Hierzu wurde versucht das Spiel ohne gelockten Tank zu starten. Das Ergebnis war ein Fenster mit dem Hinweis 'NO TANK CHOSEN!'. Versuchte man das Spiel z.B. mit '1F?' als Zeit zu starten, erschien hingegen der Hinweis 'NO VALID TIME ENTERED!'.

FileChooser

Fehlerquelle: Bei der Verwendung des FileChoosers könnte es passieren, dass der Nutzer beliebige Dateien einliest. Auch die Navigation durch Verzeichnisse könnte nicht ordentlich funktionieren.

Test: Bei dem Versuch, eine falsche Datei einzulesen zeigte sich, dass dies nicht möglich ist, da die Dateien von vornherein nach Dateiendung '.tmx' gefiltert wurden. Um die Ordnernavigation zu testen wurde hierbei versucht in diverse Verzeichnisse zu wechseln und diese auch wieder zu verlassen. Dies funktionierte ohne Probleme, da immer der richtige Ordner geöffnet wurde. Anschließend wurde das Auswählen und Bestätigen einer Datei getestet. Hierbei wurde auf den Desktop navigiert und anschließend eine Datei 'map.tmx' durch einen Klick ausgewählt. Beim Bestätigen mit dem 'LOAD' Button wurde der richtige Dateiname an das Menü übergeben und beim Spielstart auch geladen. Nun wurde noch die Funktion des 'CANCEL' Buttons getestet. Es wurde zunächst eine Map eingelesen und dann ein weiteres Mal der FileChooser geöffnet. Die Verzeichnisse wurden gewechselt und zum Schluss eine andere Map ausgewählt, jedoch dann auf 'CANCEL' gedrückt. Im Menü stand anschließend immer noch der alte Mapname und beim Spielstart wurde diese auch korrekt geladen.

GameScreen

Fehlerquelle: Im GameScreen musste sicher gestellt werden, dass die Tasten zur Steuerung der Tanks auch immer die richtige Spielfigur steuern und alle funktionierten.

Test: Im Test wurde hierzu ein Spiel mit vier Spielern erstellt und jeder Tank in alle Richtungen bewegt und geschossen. Es traten keinerlei Probleme auf und jeder Tank hatte die gleichen Funktionalitäten. In einem weiteren Test wurde überprüft, ob die Kugeln beim Schießen in eine beliebige Richtung immer am Ende des Panzerrohrs spawnen. Auch hierbei traten keine Fehler auf. Des Weiteren wurde getestet, ob sich sowohl Panzer als auch Kugeln mit konstanter Geschwindigkeit bewegen. Dieser Test erfolgte mit Augenmaß und durch den Vergleich paralleler Kugeln und Panzern.

Fehlerquelle: Auch die Kollisionen von Kugeln mit anderen Objekten könnten nicht immer das erwünschte Ergebnis geben.

Test: Im Test wurden hierzu einige Kugeln aus unterschiedlichsten Richtungen auf ein Hindernis geschossen. Alle Kugeln wurden unverzüglich nach dem Aufprall gelöscht. Bei Kollisionen mit anderen Spielern musste zusätzlich überprüft werden, ob diese auch entsprechenden Schaden nehmen. Dabei wurden von jedem Tank Kugeln auf andere Tanks abgefeuert. Der Schaden wurde immer korrekt vom Leben abgezogen und beim Sinken der Lebensanzeige auf 0, wurden zusätzlich die Kill- und Death-Counter korrekt erhöht.

Fehlerquelle: Sehr wichtig sind auch die Kollisionen der Tanks mit anderen Objekten und dass diese auch realistisch sowie fehlerfrei aufgelöst werden.

Test: Beim ersten Test wurde versucht mit den Tanks das Spielfeld zu verlassen. Die Ränder verhielten sich hierbei wie Wände, durch die der Tank nicht fahren konnte. In einem weiteren Test erfolgten dann Kollisionen mit Hindernissen, indem von allen Richtungen auf diese zugefahren wurde. Diese Kollisionen verliefen größtenteils sehr gut, sodass die Tanks nie die Hindernisse durchdringen konnten. Lediglich in wenigen Fällen, in denen der Tank diagonal die Ecken von Hindernissen streifte, verlief die Kollision etwas unnatürlich. In einem letzten Test wurden die komplexeren Kollisionen zwischen zwei sich bewegenden Tanks getestet. Dies geschah indem alle vier Spieler auf engem Raum aufeinander zufuhren. Auch hierbei kam es zu fast keinen Problemen. Nur in seltenen Fällen verursachten die vielen Kollisionen kleine, unnatürliche Sprünge der Tanks.

Fehlerquelle: Inkorrektter Ablauf der Spielzeit.

Test: Dies wurde getestet, indem die Spieldauer mit der aus dem Menü verglichen wurde und geprüft wurde ob sie sich im Sekundentakt verringert. Dies funktionierte ohne Probleme und nach Ablauf der Zeit wechselte das Spiel automatisch zum EndScreen.

EndScreen

Fehlerquelle: Im EndScreen könnten falsche oder unbeteiligte Spieler gezeigt werden oder die Sortierung nicht der Platzierung der Spieler entsprechen.

Test: Deswegen wurden mehrmals die Ergebnisse und Spieler eines Spiels notiert und mit den Werten im EndScreen verglichen. Auch die Sortierung nach der Anzahl an Kills wurde überprüft. Hierbei zeigten sich keinerlei Fehler. Zuletzt wurde die Funktionalität des 'NEW GAME' Buttons überprüft. Beim Betätigen wechselte das Spiel erfolgreich in den MenuScreen, in dem die Einstellungen für ein weiteres Spiel gemacht werden konnten.