



Höhere Fachschule
Südostschweiz

Meine Schule. Meine Zukunft.

Semesterarbeit

Modul Grundlagen Software-Entwicklung

Dokumentation "Tank Wars"

Rade Ilic, Phillip Tuor, Stefan Hutter

16.1102A-2019
NDS HF Applikationsentwicklung
Abgabedatum: 31.01.2020

Fachkorrektor:

Ueli Kunz
ibW Höhere Fachhochschule Südostschweiz
Gürtelstrasse 48, 7000 Chur

Management Summary

Die vorliegende Arbeit wurde in einer Gruppe bestehend aus drei Personen erarbeitet. Dabei beträgt der ungefähre Arbeitsumfang pro Person 20 Stunden und die zu erstellende Applikation soll in Java geschrieben werden. Anhand dieser Informationen wurde nach einer Projektidee gesucht, wobei der Entschluss auf eine Applikation genannt "Tank Wars" fiel. Kurzgefasst ist "Tank Wars" ein Spiel bei dem Panzer gesetzt und angegriffen werden können, gegen einen virtuellen wie auch realen Gegner.

Zu Projektbeginn wurde eine Projektskizze erstellt, welche den Umfang der Arbeit widerspiegelt. Als Grundbasis zur Erstellung der Applikation dient die Entwicklungsumgebung (IDE) IntelliJ. Zuerst wurde die Klassen Topologie anhand von UML-Diagrammen konzipiert. Anschliessend wurden die Klassen implementiert und die Applikation Schritt für Schritt erweitert. Die Benutzeroberflächen (GUI) wurden ebenfalls im Vorfeld mithilfe von Mockups definiert. Für die GUI-Erstellung wurde JavaFX verwendet. Die Applikation wurde schlussendlich mittels drei GUI's realisiert. Das erste GUI für die generellen Spieleinstellungen, das Zweite für den Mehrspieler-Modus und das Dritte für den Spielverlauf.

Das Ergebnis ist eine funktionsfähige Applikation, welche den Anforderungen der Projektskizze entspricht. Zu Beginn erscheint eine Benutzeroberfläche, bei der der Spielmodus, die Spielfeldgrösse und Anzahl Panzer gewählt werden. Nach Bestätigung der Inputs kann das Spiel gestartet werden. Zuerst müssen die Anzahl Panzer im eigenen Spielfeld gesetzt werden. Ist der Modus auf Singleplayer geschaltet, so setzt der virtuelle Gegner automatisch die gleiche Anzahl Panzer auf das gegnerische Spielfeld. Danach kann der Spieler den ersten Angriff durchführen, indem in eine Zelle des gegnerischen Felds gedrückt wird. Anschliessend startet der virtuelle Gegner automatisch einen Gegenangriff. Wer zuerst alle Panzer getroffen hat, gewinnt das Spiel. Beim Mehrspielermodus erscheint eine weitere Benutzeroberfläche, in der die IP-Adresse des realen Gegners eingegeben werden kann. Nach der Etablierung der Verbindung kann nun abwechselungsweise Angriffe gestartet werden bis jemand das Spiel gewinnt. Zusammengefasst wurden alle User Storys mit der Priorisierung „MUST“ und „SHOULD“ umgesetzt. Optionale Features, welche mit „COULD“ priorisiert wurden, sind aus zeitlichen Gründen nur teilweise implementiert worden.

Dementsprechend könnten in den nächsten Schritten die Applikation um die optionalen, nicht abgearbeiteten Features erweitert werden. Darunter versteht sich eine Spielerdaten-Aufzeichnung mit Ranking und die Auswahl der Panzergrösse (z.B. 1 – 3 Felder) als erweiterte Spieleinstellung.

Inhaltsverzeichnis

Management Summary	1
1 Einleitung	3
2 Ausgangslage	4
2.1 Aufgabenstellung	4
2.2 Projektidee	4
2.3 Vorgehensweise.....	5
2.4 Gantt-Diagramm.....	5
3 Hauptteil.....	6
3.1 Use Case	6
3.2 Mockups und Diagramme	8
3.3 Priorisiertes Backlog.....	9
3.4 Versionsverwaltung.....	11
3.5 Code Darstellung UML	12
3.6 Realisierung JavaFX-Darstellung.....	16
3.7 Unit Tests.....	17
3.8 Code Diverses.....	17
4 Fazit und Schlussfolgerungen	18
Abkürzungsverzeichnis.....	20
Literaturverzeichnis	20
Abbildungsverzeichnis.....	20
Tabellenverzeichnis.....	20

1 Einleitung

Immer mehr steigt die Nachfrage nach kostengünstigen Apps/Spielen mit möglichst hohem Spassfaktor, welche vor allem in kurzen Wartezeiten spielbar sind, jedoch auch generell über einen längeren Zeitraum Spielspass versprechen. Die Dimensionen der Spielentwicklung sind sehr vielfältig und unterscheiden sich erheblich im Spielkonzept und Spieldesign, je nachdem welche Zielgruppe angesprochen werden möchte. Dabei bieten sich zahlreiche Entwicklungsebenen an zur Entwicklung einer Applikation. Die in dieser Arbeit verwendete Entwicklungsumgebung (IDE) IntelliJ ist eine der mächtigsten IDE's und basiert auf der Hochsprache Java. Diese funktionsreiche IDE ermöglicht eine schnelle Entwicklung und hilft bei der Verbesserung der Code-Qualität. (Jet Brains, 2008)

Das Ziel der Arbeit ist, eine funktionsfähige Applikation zu entwickeln, welche die oben genannten Eigenschaften widerspiegelt. Des Weiteren sollen umfangreiche Eigenschaften zur Befriedigung des Spielers umgesetzt werden, wie z.B. variantenreiche Spieleinstellungen (Einzel-, Mehrspielermodus, Spielverlauf), ansprechendes Design (GUI) und eine angenehme Spielatmosphäre (Musik/Soundeffekte).

Das Zeitmanagement der Arbeit wird mithilfe eines Gantt-Diagramms realisiert. Zu Beginn wurde eine Projektskizze erstellt mit relevanten Inhalten zur Implementation. Dabei wird besonderen Wert auf den Use Case und die Mockups gelegt, welche die Basis der Applikation visualisieren. Ein Teil der Projektorganisation richtet sich nach dem Scrum Rahmenwerk, jedoch wird nicht komplett nach Scrum gearbeitet, da sich diese Form von agilem Projektmanagement nicht in allen Bereichen für Projekte mit sehr engem Zeitrahmen eignet. Allerdings wird gemäss Scrum Rahmenwerk ein Backlog erstellt mit funktionalen und nicht funktionalen, priorisierten User Storys. Zur Versionsverwaltung wird Git verwendet. Das Remote Repository wird auf Github eingerichtet. Die Regeln zur Nutzung des Repository sind im readme.txt-file niedergeschrieben und die User Storys werden in einem Scrum Board visualisiert. (Schwaber & Sutherland, 2017)

Parallel zur Erstellung der Projektskizze und Aufsetzung des Repository wird die Testphase eingeleitet. Dabei werden die Klassen gemäss UML-Konzeption programmiert und erste Testphasen durchgeführt. Ebenfalls werden Unit Tests mit JUnit implementiert, um die Codequalität jederzeit testen zu können. Nach der Testphase wird die Entwicklungsphase eingeleitet, in welcher die Applikation im Detail optimiert wird.

Im ersten Teil der Arbeit wird die Ausgangslage des Projekts beschrieben. Darauf aufbauend wird der Hauptteil bestehend aus Vorbereitung bis Umsetzung der Applikation dargestellt. Zum Schluss wird anhand der Schlussfolgerungen die Arbeit abgerundet.

2 Ausgangslage

Dieser Abschnitt beschreibt, wie die Aufgabenstellung und Ausgangssituation aussehen. Die Aufgabenstellung zeigt die Grundvoraussetzungen für das Projekt. Um die Ausgangssituation zu verstehen, wird die Projektidee zusätzlich erklärt. Nach diesen Informationen wird die Vorgehensweise aufgezeigt. In der Vorgehensweise sind die Methoden zur Erreichung der User Stories erläutert.

Nach der definierten Ausgangslage kann das Projekt gemäss Zeitplan gestartet werden. Weitere Informationen zum Thema Projektstart und Planung sind in der Projektskizze dokumentiert.

2.1 Aufgabenstellung

Im Nachdiplomstudium HF Applikationsentwickler wird nach dem ersten Modul eine Semesterarbeit erwartet. Die Semesterarbeit wird in einer Gruppe erarbeitet. Der Umfang der Arbeit beträgt pro Student mindestens 20 Arbeitsstunden. Folgende technischen Rahmenbedingungen sind verbindlich für die Semesterarbeit:

- Entwicklung einer objektorientierten Java Applikation
- Weitere Technologien möglich (z.B. Datenbank)
- Einsatz von Git
- Testabdeckung mit Unit Tests

2.2 Projektidee

Mit der Aufgabenstellung als Grundlage wurden verschiedene Projektideen gesammelt. Diese Ideen wurden anschliessend analysiert und zur Auswahl bereitgestellt. Die Entscheidung fiel auf ein Spiel genannt "Tank Wars". Nach dem Entscheid wird die Vorgehensweise definiert.

2.3 Vorgehensweise

Um die Projektidee umzusetzen wird ein Teil von Scrum angewendet. Es wird ein Use Case zusammen mit User Stories erstellt. Die User Stories werden im Git Repository auf einem Kanban verteilt. In den User Stories sind die einzelnen Tasks genauer beschrieben. Im Git Repository im Readme-File wird definiert, wann ein Task beendet ist. In den Mockups ist die grafische Oberfläche, die im Use Case erwähnt wird, dargestellt.

Sobald die Ausgangslage definiert und das Projektteam mit allen Schritten einverstanden ist, kann das Projekt gemäss Zeitplan gestartet werden.

2.4 Gantt-Diagramm

Die Zeitplanung für das Projekt sieht wie folgt aus:

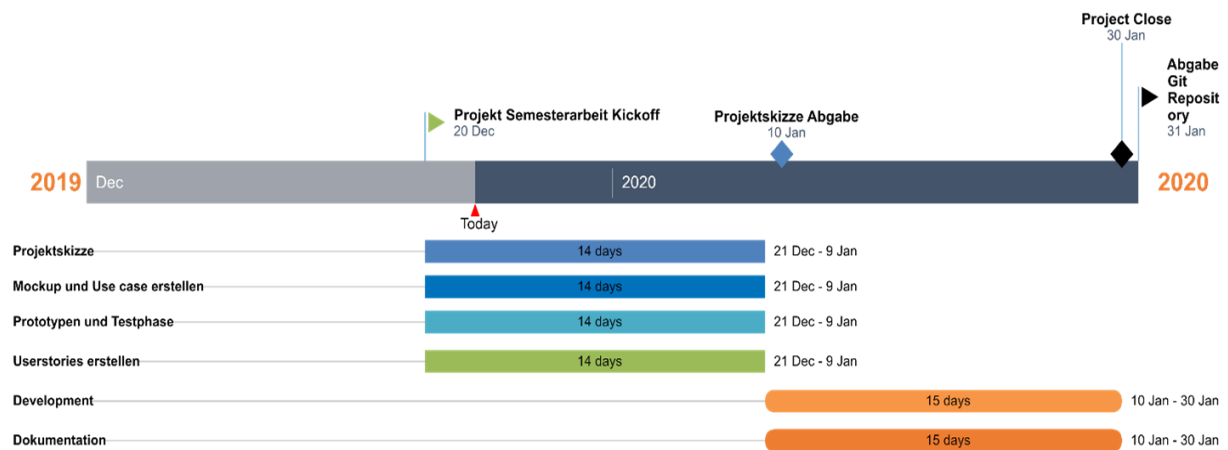


Abbildung 1: Gantt-Diagramm für die Zeitplanung des Projekts.

3 Hauptteil

3.1 Use Case

Der Use Case wird in [Abbildung 2](#) graphisch dargestellt. Dabei wird dieser in drei verschiedene Stufen unterteilt:

1. GUI mit zwei Benutzeroberflächen (Spieleinstellungen und Spielverlauf)
2. Die Applikation, welche den Code beinhaltet
3. Optionale Features, welche möglicherweise implementiert werden

Wird die Applikation gestartet, erscheint die Benutzeroberfläche für die Spieleinstellungen. Der Spieler hat die Möglichkeit die Menge der Panzer und die Spielfelddimension auszuwählen.

- Panzer: Die Grösse eines Panzers beträgt eine Zelle im Spielfeld
- Spielfeld: Die Anzahl der Zellen im Spielfeld wird erstellt, indem der eingegebene Wert mit 2 potenziert wird (z.B. bei Eingabe von 5 hat das Spielfeld 25 Zellen).

Sind die Einstellungen bestätigt, erscheint ein „Play“-Button. Nach Drücken dieses Buttons öffnet sich eine weitere Benutzeroberfläche für den Spielverlauf.

Auf dieser Benutzeroberfläche werden die Spielfelder des Spielers und des Gegners visualisiert. Zuerst muss der Spieler die gewählte Anzahl Panzer auf dem eigenen Spielfeld, auf der linken Seite, platzieren. Die verbleibende Anzahl der zu setzenden Panzer wird in einem Textfeld angezeigt. Sobald alle eigenen Panzer gesetzt sind, erscheint das gegnerische Feld auf der rechten Seite. Der virtuelle Computergegner setzt anschliessend dieselbe Anzahl an Panzern zufällig auf dem gegnerischen Spielfeld. Der Spieler kann jetzt nicht mehr in sein eigenes Feld klicken. Beide Felder haben die Farbe schwarz für die Rahmenlinien und weiss für den Zellinhalt. Die gesetzten Panzer auf dem Spielfeld des Spielers werden grün angezeigt, die des virtuellen Gegners werden nicht angezeigt.

Die Spielvorbereitungen sind abgeschlossen und somit kann das eigentliche Spiel beginnen. Dabei kann der Spieler eine Zelle des gegnerischen Spielfelds anklicken. Geht der Angriff daneben wird die Zelle schwarz eingefärbt, bei einem Treffer rot. Nach jedem ausgeführten Angriff startet der Gegner einen zufälligen Angriff auf das Feld des Spielers. Ebenfalls wird bei einem Fehlschuss die Zelle schwarz und bei einem Treffer rot gefärbt. Das Spiel wird solange weiter gespielt bis einer die komplette Anzahl Panzer des Gegners getroffen hat und somit das Spiel gewinnt. Die Anzahl der getroffenen und zu treffenden Panzer wird in einem Textfeld angezeigt. Das Spiel kann jederzeit durch Drücken des „Cancel“-Buttons beendet werden.

Optionale Features:

- Multiplayer Modus, um gegen eine reale Person spielen zu können. GUI Erweiterung mit Eingabe der IP-Adresse des Gegners.
- Festlegung der Panzergröße (z.B 2, 3... Zellen)
- Registrierung des Spielers und Spielerdaten Aufzeichnung mithilfe einer Datenbank.

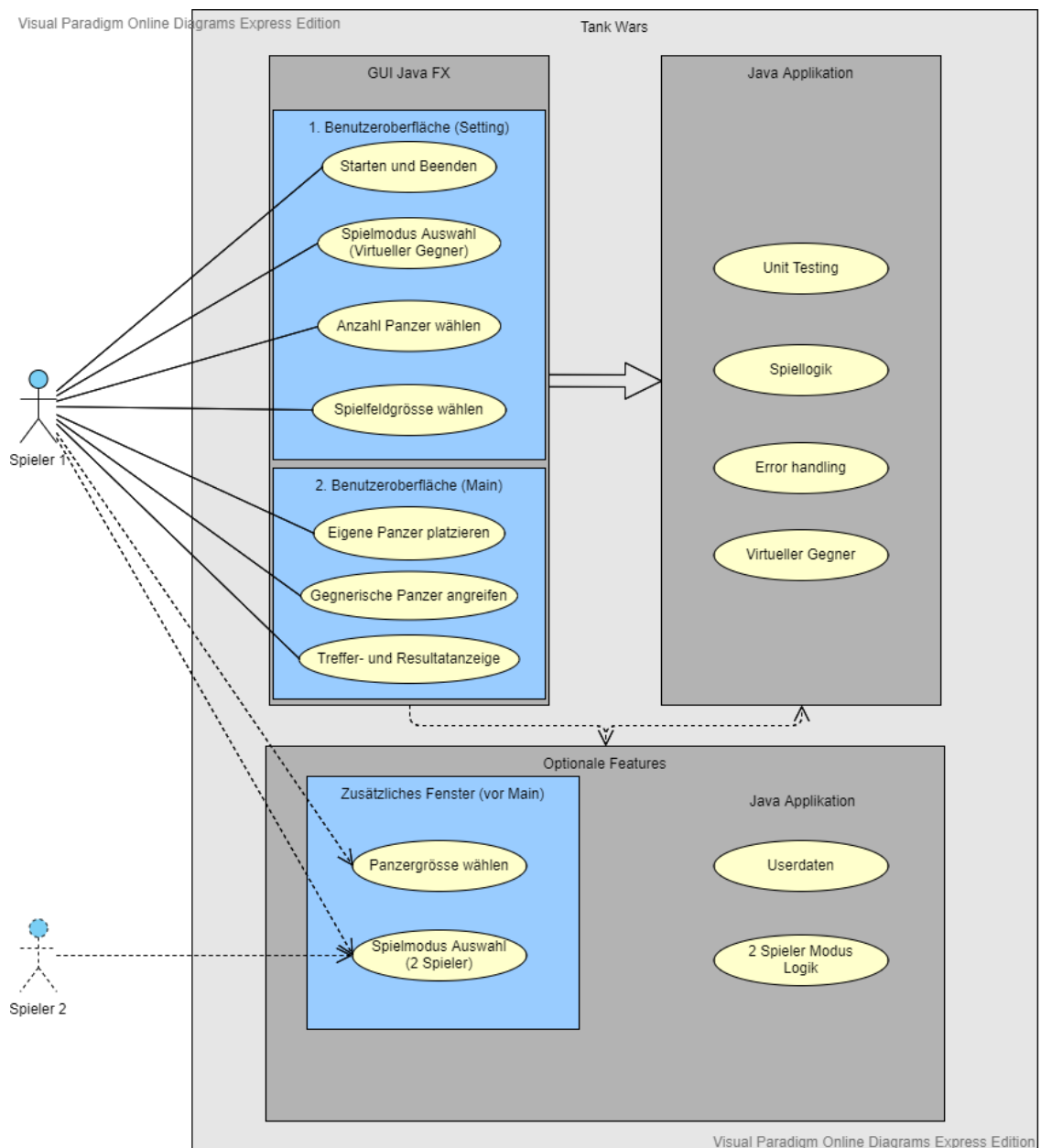


Abbildung 2: Use Case der Applikation Tank Wars

3.2 Mockups und Diagramme

Die Mockups dienen als Visualisierungsvorlage für die Erstellung der Benutzeroberflächen. Dabei werden drei Benutzeroberflächen realisiert:



Abbildung 3: Benutzeroberfläche Spieleinstellungen

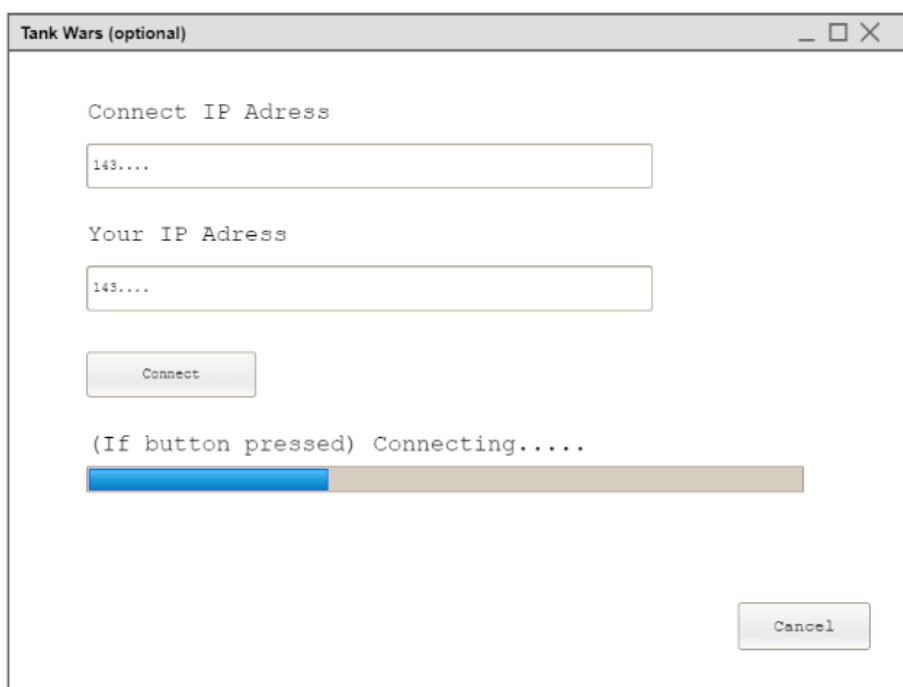


Abbildung 4: Benutzeroberfläche Mehrspieler-Modus (optional)

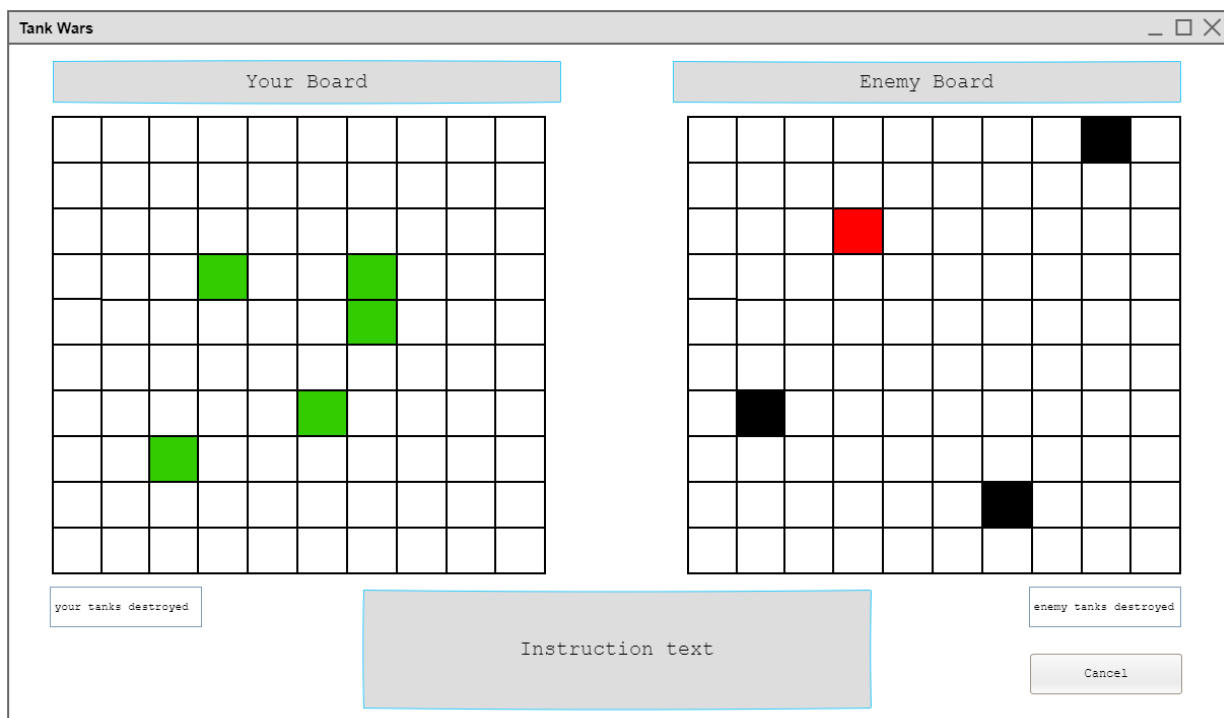


Abbildung 5: Benutzeroberfläche Spielverlauf

3.3 Priorisiertes Backlog

Das Backlog wird in funktionale und nicht funktionale User Stories unterteilt. Der Aufwand der User Stories wurde geschätzt und dementsprechend Story Points vergeben. Ein Story Point entspricht einer Arbeitsstunde.

Die Priorisierung der Einträge erfolgt nach dem MoSCoW Schema. Die Einträge werden folgendermassen klassifiziert:

- MUST (unbedingt für die Umsetzung erforderlich)
- SHOULD (Umsetzen, wenn alle Must have's erledigt sind)
- COULD (kann umgesetzt werden, wenn höherwertige Anforderungen nicht beeinträchtigt werden)
- WON'T (wird nicht umgesetzt, evtl. für spätere Umsetzungen interessant)

Tabelle 1: Funktionale User Stories

ID	Priorisierung	Beschreibung	Thema/Story	Story Points
1	MUST	Als User kann ich das Game direkt auf einem GUI spielen, wodurch die Bedienung erleichtert wird.	GUI Spielverlauf	15
2	MUST	Als User möchte ich das Game gegen einen Computergegner spielen, um alleine spielen zu können.	Applikation	6
3	MUST	Als User möchte ich die Anzahl Panzer und Feldgrösse selbst bestimmen, um das Game variantenreich zu spielen.	GUI Spieleinstellung	4
4	MUST	Als User möchte ich eine farbliche Darstellung für die ausgeführten Angriffe bzw. die getroffenen Panzer und Fehlschüsse auf dem jeweiligen Spielfeld, um den Spielverlauf besser nachvollziehen zu können.	GUI Spielverlauf	10
5	MUST	Als User möchte ich anhand einer Anzeige erkennen, wer während dem Spielverlauf wie viele Panzer bereits zerstört hat.	GUI Spielverlauf	2
6	COULD	Als User möchte ich die Möglichkeit die Panzergrösse zu wählen, um das Game variantenreicher zu spielen	Applikation	15
7	COULD	Als User möchte ich die Möglichkeit Mehrspieler Modus, um mit meinen Freunden spielen zu können.	Applikation	15

Tabelle 2: Nicht funktionale User Stories

ID	Priorisierung	Beschreibung	Thema/Story	Story Points
8	SHOULD	Als User möchte ich eine intuitiv gestaltete Benutzeroberfläche haben, damit das Game schnell verständlich ist.	GUI allgemein	10
9	COULD	Als User möchte ich während dem Spielen Musik hören, um eine passende Atmosphäre zu schaffen	Applikation	3

Die User Stories wurden zu Projektbeginn im GitHub Repository ins Kanban-Board platziert und Schritt für Schritt abgearbeitet. Das Kanban-Board ist unterteilt in die Spalten To Do, In Progress, Review und Done. Dadurch konnten wir, als kleines Team, unserem Projekt mehr Struktur geben und somit die Effektivität steigern.

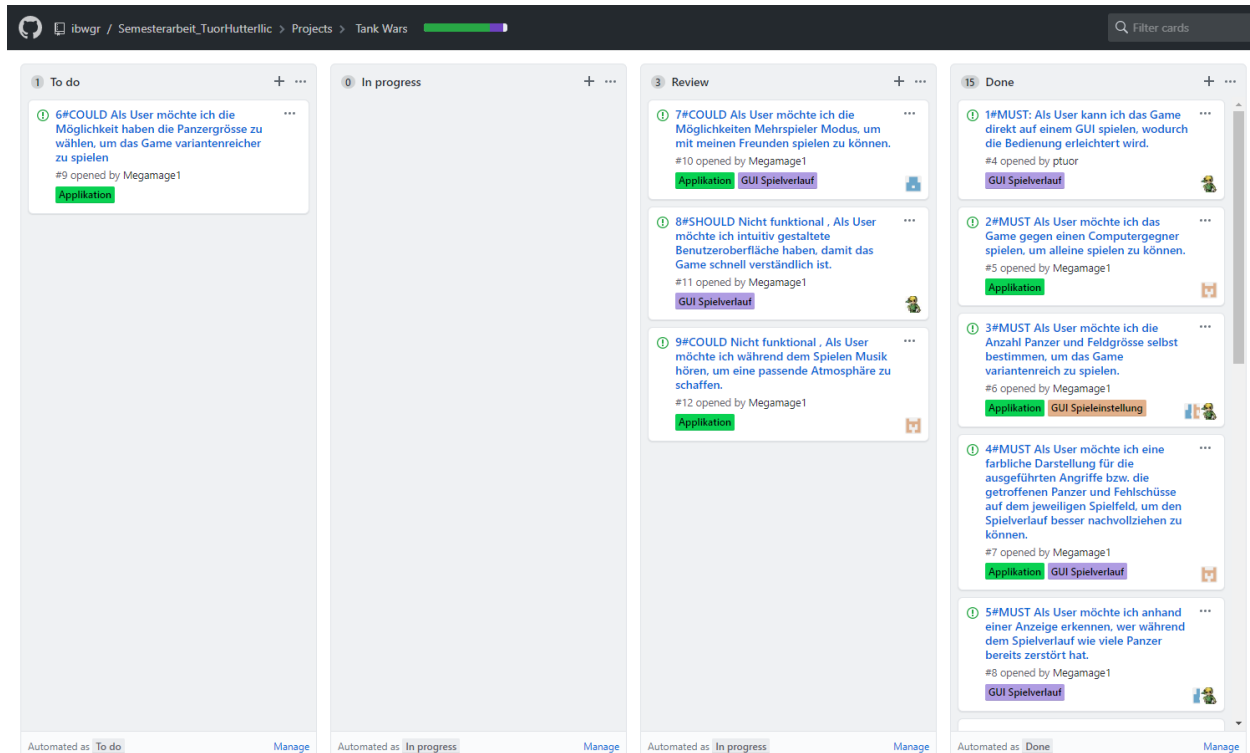


Abbildung 6: Kanban-Board auf GitHub Repository

3.4 Versionsverwaltung

GitHub wird als Repository für die Arbeit verwendet. Darin ist ein README File enthalten, welches die Regeln des Repository beschreibt. Gearbeitet wurde auf dem Development Branch, welcher jeweils die neusten Commits aufweist. In den Master Branch wurde erst ein Push durchgeführt, wenn eine User Story den Status DONE erreicht hat. DONE bedeutet, dass bei einer abgeschlossenen User Story ein weiteres Teammitglied die Funktion im Development Branch und anschliessend nach dem Merge in den Master Branch getestet hat.

3.5 Code Darstellung UML

Der Ansatz der Objektorientierten Programmierung besteht darin, Objekte aus der realen Welt im Programm abzubilden wobei Objekte mit gemeinsamen Eigenschaften zusammengefasst werden.

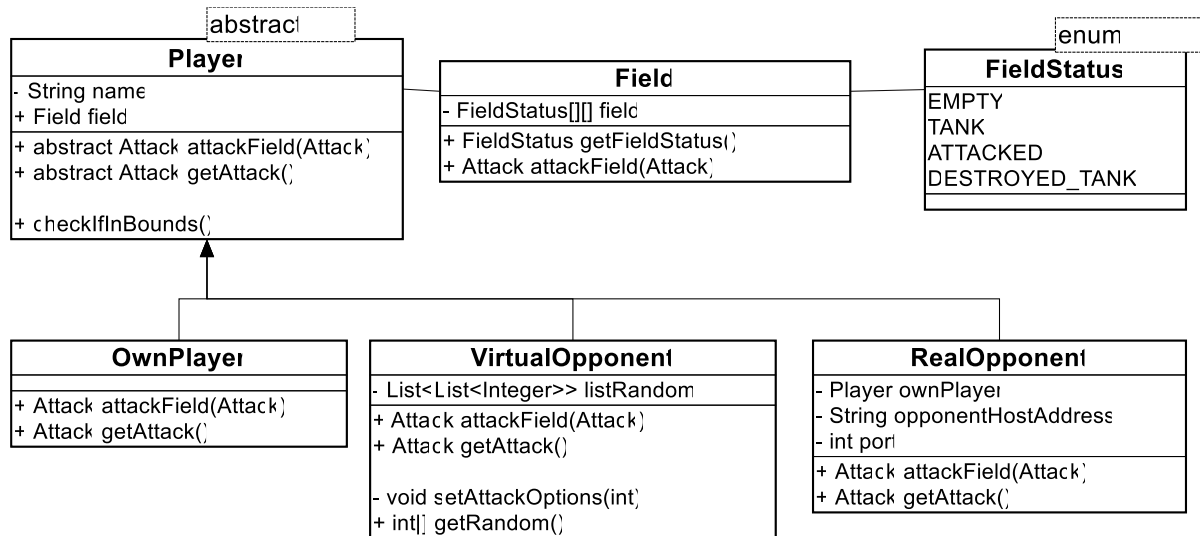


Abbildung 7: UML Modellierung Abstrakte Klasse Player

Dieser Ansatz wurde umgesetzt, indem eine abstrakte Klasse «Player» erstellt wurde, welche die gemeinsamen Eigenschaften aller Spieler zusammenfasst. Zudem gibt diese abstrakte Klasse zwei abstrakte Methoden «attackField» und «getAttack» vor, welche von den Subklassen implementiert werden müssen. Die Implementierung dieser Methoden in den Subklassen kann jedoch individuell umgesetzt werden.

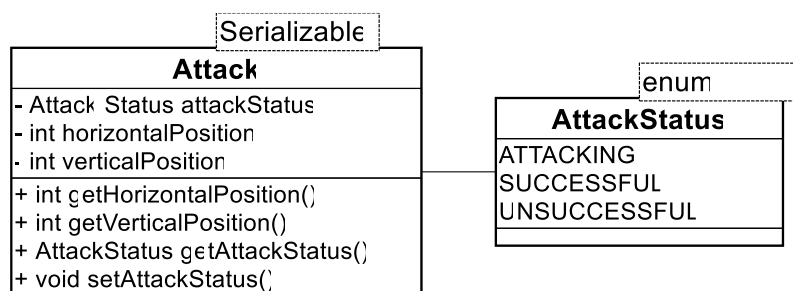


Abbildung 8: UML Modellierung – Attacke

Um den Gegner zu Attackieren oder um vom Gegner attackiert zu werden, benötigt es Objekte, welche die Informationen solcher Attacken speichern. Dafür wurde die Klasse «Attack» entworfen, welche die Koordinaten einer Attacke, sowie den Status der Attacke als Attribut implementiert. Wird eine Attacke auf das gegnerische Spielfeld ausgeführt, wird die Position der Attacke entsprechend gesetzt und den Status auf «ATTACKING» gesetzt. Der Gegner wertet

die Attacke anhand der Koordinaten aus, verändert anschliessend den Status der Attacke auf «SUCCESSFUL», wenn ein Panzer getroffen wurde oder «UNSUCCESSFUL», falls nicht getroffen wurde. Das veränderte Objekt wird anschliessend wieder zurückgegeben, wodurch der Status der Attacke ausgewertet werden kann. Die Klasse «Attack» implementiert das Interface «Serializable», wodurch dieses Objekt über eine Client/Server Verbindung geschickt werden kann.

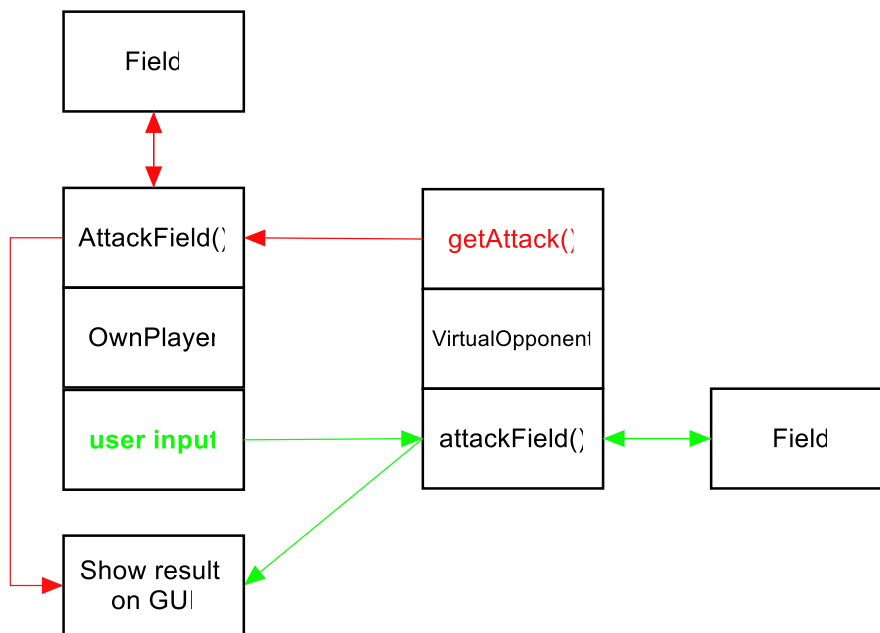


Abbildung 9: Spielverlauf Einzelspieler-Modus

Wird das Spiel im Einzelspieler Modus gestartet, wird ein virtueller Gegenspieler erstellt, wobei während der Erzeugung des Objekts die Panzer mit Hilfe der «getRandom» Methode zufällig auf dessen Spielfeld platziert werden. Wird der virtuelle Gegner mit der Methode «attackField» angegriffen, prüft der virtuelle Gegner ob dieser Angriff einen Panzer auf seinem Feld trifft oder nicht. Dementsprechend wird der Status der Attacke gesetzt und wieder zurückgegeben. Wird eine Attacke des Gegners mit der Methode «getAttack» angefordert, erzeugt der virtuelle Gegner mit Hilfe der «getRandom» Methode eine zufällige gewählte Attacke (X/Y-Position) und returniert diese. Es wird darauf geachtet, dass nicht zweimal auf das gleiche Feld attackiert werden kann. Anschliessend wird geprüft, ob diese zufällige Attacke einen eigenen Panzer trifft oder nicht. Dementsprechend wird das Ergebnis auf dem GUI angezeigt.

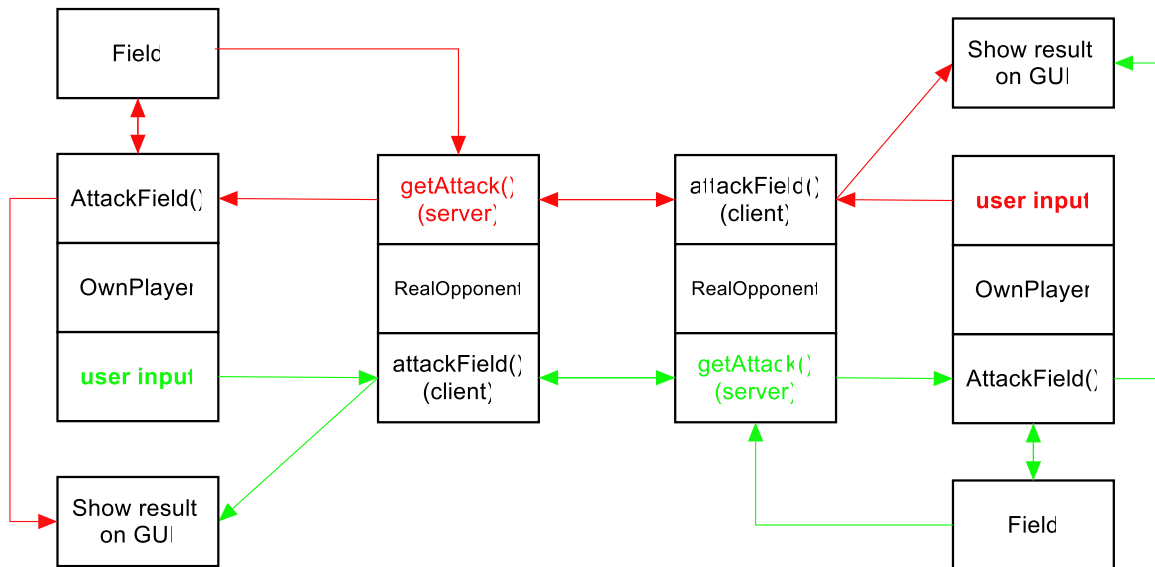


Abbildung 10: Spielverlauf Mehrspieler-Modus

Sollte das Spiel im Mehrspieler Modus gestartet werden, wird ein Objekt (Realer Gegner) erstellt. Wird der reale Gegner mit der Methode «attackField» angegriffen, leitet dieser die Attacke über eine Client Verbindung zum Gegenspieler (Server) weiter. Dieser wertet die Attacke aus, indem geprüft wird, ob die Attacke einen selbst platzierten Panzer trifft oder nicht. Dementsprechend wird der Status der Attacke gesetzt und wieder zum Client zurückgeschickt. Dieser muss das Ergebnis nur noch auf dem GUI anzeigen.

Wird eine Attacke des Gegners mit der Methode «getAttack» angefordert, wird eine Server Verbindung gestartet und auf die Attacke des Gegenspielers (Client) gewartet. Sobald die Attacke des Gegenspielers erhalten wurde, wird geprüft ob diese Attacke einen eigenen Panzer trifft oder nicht. Dementsprechend wird dies im GUI angezeigt, sowie den Status der Attacke angepasst und wieder zurückgeschickt.

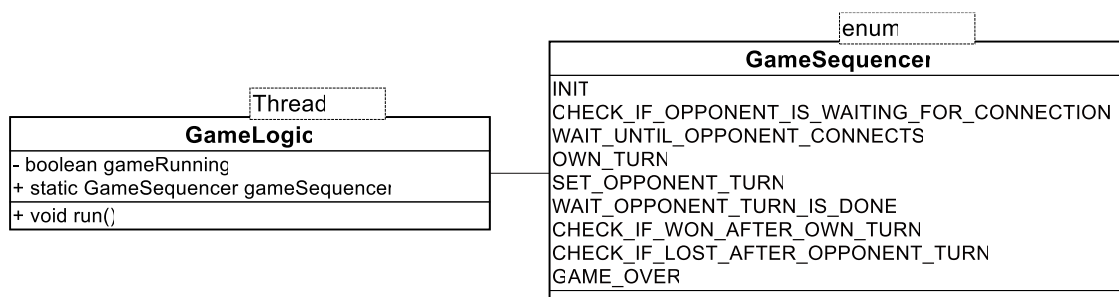
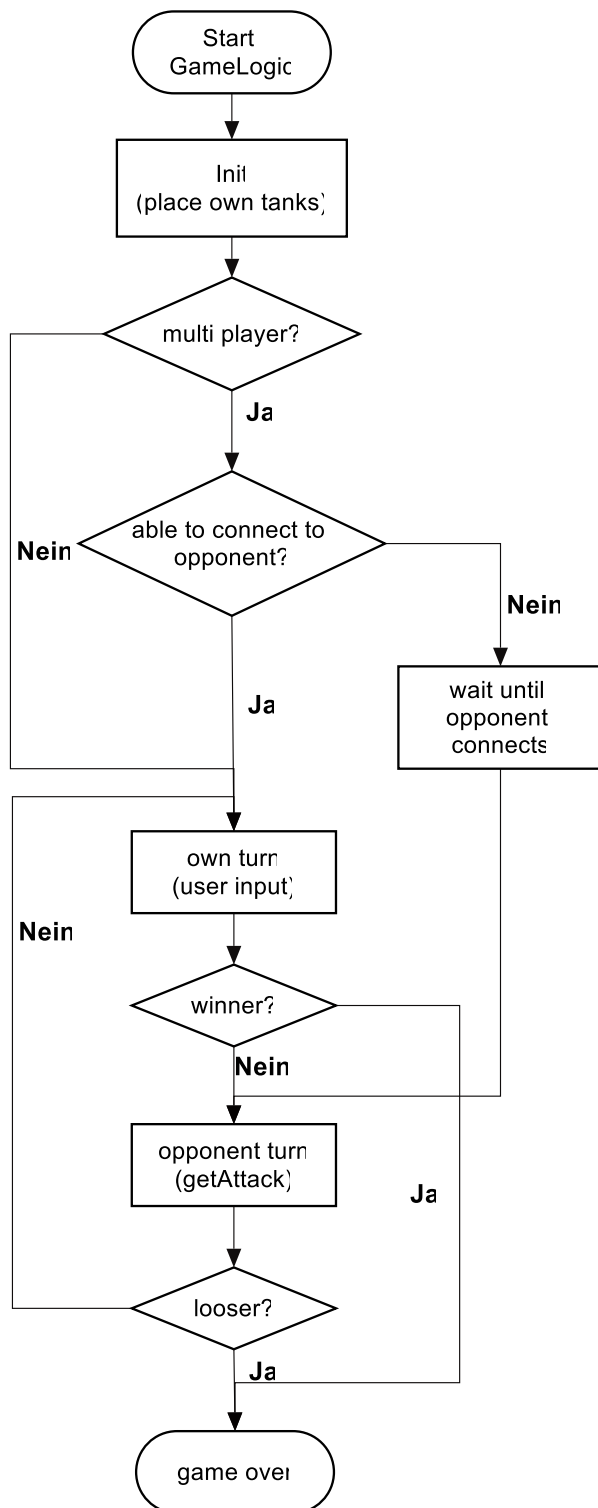


Abbildung 11: UML-Modellierung GameSequencer

Damit beide Spielmodi ordnungsgemäss funktionieren, ist es wichtig die Reihenfolge der Spielzüge sowie Aktionen während des Spiels einzuhalten. Zudem ist es wichtig für den Mehrspieler Modus, festzulegen wer den ersten Zug machen darf. Um dies zu garantieren

wurde eine Klasse «GameLogic» entwickelt, welche mit der Methode «run» als eigenständiger Thread gestartet werden kann. Diese «run» Methode läuft so lange in einer Dauerschleife, bis das Spiel entweder gewonnen, verloren oder abgebrochen wird. Die einzelnen Schritte, die eingehalten werden müssen, wurden zur besseren Lesbarkeit des Codes in einem Enumerationstyp «GameSequencer» definiert.



1. Start der «run» Methode
2. Warten bis alle eigenen Panzer platziert wurden
3. Ist Mehrspielermodus aktiv?
Nein → Starte Spiel mit eigenem Zug
Ja → Verbindung zum Gegner prüfen
4. Kann eine Verbindung zum Gegner hergestellt werden.
Nein → Warte auf Verbindung
Ja → Starte Spiel mit eigenem Zug
5. Warten, bis der Gegner sich verbindet. Sobald Verbindung akzeptiert wurde, wird Spiel mit gegnerischem Zug gestartet
6. Eigener Zug (user input)
7. Wurde Spiel nach eigenem Zug gewonnen?
Nein → Gegner ist am Zug
Ja → Spiel ist zu Ende
8. Gegnerischer Zug
9. Wurde Spiel nach gegnerischem Zug verloren?
Nein → setze das Spiel mit eigenem Zug fort.
Ja → Spiel ist zu Ende
10. Ende des Spiels, verlassen der Dauerschleife

3.6 Realisierung JavaFX-Darstellung

Die grafische Benutzeroberfläche (GUI) ist mit JavaFX umgesetzt. Dazu werden drei Klassen benötigt: StartScreen, MainWindow, IpWindow.

Die drei Klassen müssen von der Klasse Application erben, damit die benötigten Methoden benutzt werden können.

Der StartScreen wird mit launch() über die Main-Methode gestartet. Diese Methode kann nur einmal in einem Programm ausgeführt werden.

Um das MainWindow oder IpWindow aufzurufen, wird ein Objekt der Klasse MainWindow oder IpWindow erzeugt. Die beiden Klassen übergeben mit der Methode start() die Stage, in welcher eine neue Szene erstellt. In der Szene braucht es zwingend eine Root Node. Erst aus dem Root Node können Parent Nodes erzeugt werden. Ein Root Node besteht aus einem ausgewählten "Pane" (z.B. Gridpane). In diesem Root Nodes können die Parent Nodes (z.B. Buttons, Textfelder und Label) platziert werden.

Der Aufbau der grafischen Benutzeroberflächen ist in folgendem Bild dargestellt:

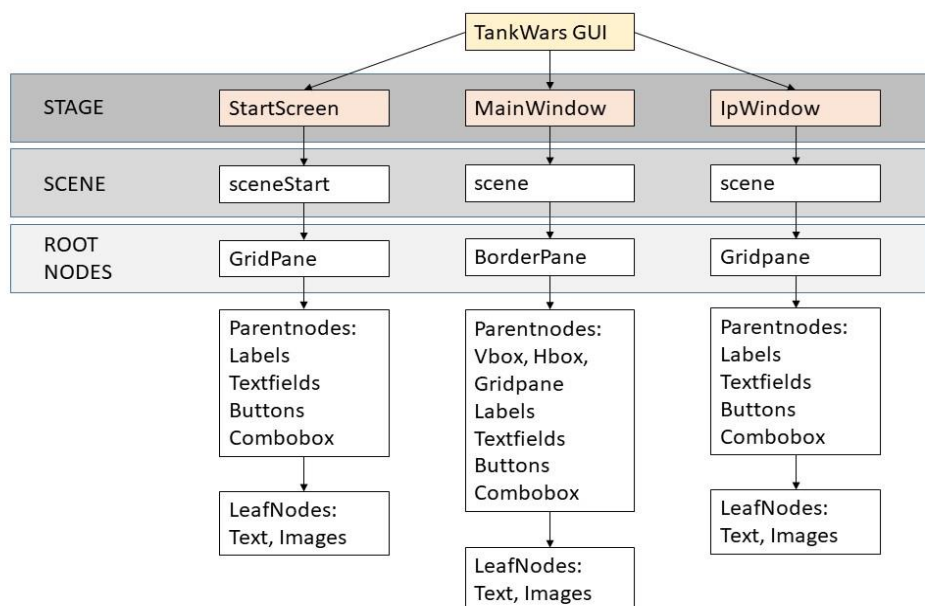
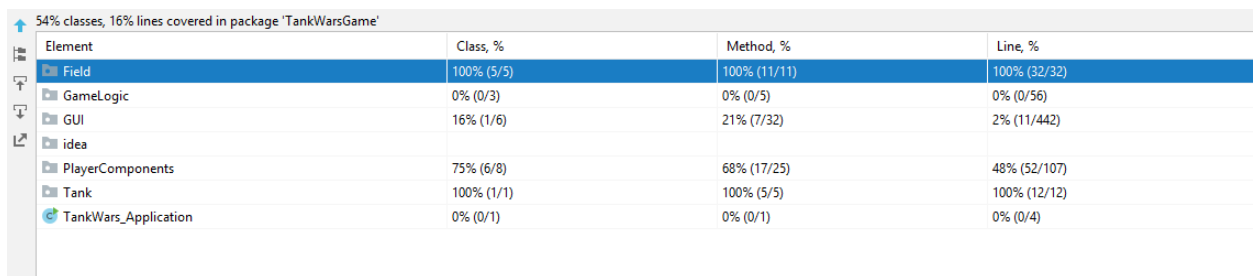


Abbildung 12: Aufbau der graphischen Benutzeroberflächen

3.7 Unit Tests

Ziel der Unit Tests war, zu überprüfen ob unsere entwickelten Methoden arbeiten wie beabsichtigt. Da wir zu Beginn nicht genau wussten, welche Methoden benötigt werden, war es kaum möglich die Testgetriebene Entwicklungsmethode anzuwenden. Aus diesem Grund wurden alle Unit Tests erst geschrieben, nachdem die Klassen/Methode erstellt wurden. Da das JUnit Framework direkt in Maven integriert war, konnten alle Unit Test auf einmal ausgeführt werden, wodurch diese öfter durchgeführt wurden. Dank Unit Tests konnten kleine Programmierfehler frühzeitig erkannt und behoben werden.



54% classes, 16% lines covered in package 'TankWarsGame'			
Element	Class, %	Method, %	Line, %
Field	100% (5/5)	100% (11/11)	100% (32/32)
GameLogic	0% (0/3)	0% (0/5)	0% (0/56)
GUI	16% (1/6)	21% (7/32)	2% (11/442)
idea			
PlayerComponents	75% (6/8)	68% (17/25)	48% (52/107)
Tank	100% (1/1)	100% (5/5)	100% (12/12)
TankWars_Application	0% (0/1)	0% (0/1)	0% (0/4)

Abbildung 13: Prozentuale Anzeige der UNIT-Test Abdeckung

Die Unit Tests sind mit einer Gesamtabdeckung von 54% der Klassen und Methoden eher gering ausgefallen, jedoch konnten alle JavaFX Komponenten nicht durch Unit Tests getestet werden.

3.8 Code Diverses

Eine weitere implementierte Klasse, welche nicht von Beginn an geplant war, da diese User Story mit "COULD" priorisiert war, ist der MusicPlayer. Darin wurden anhand zweier Methoden einerseits die Titelmusik, welche bei laufender Applikation in einer Endlosschleife abgespielt wird und andererseits diverse Soundeffekte erzeugt:

- Setzen der Panzer
- Angriff durch Spieler oder Gegner
- Sieg
- Niederlage

4 Fazit und Schlussfolgerungen

Zu Beginn wurde eine Projektskizze erarbeitet, bei der ein Teil das Erstellen von User Stories beinhaltet. Dies war eine schwierige Aufgabe, da man sich Gedanken über das Endprodukt aus einer völlig neuen Perspektive machen musste. Obwohl es schwierig war passende User Stories zu erarbeiten, hat es dennoch sehr geholfen, einen Überblick über die eigenen Vorstellungen, sowie die Vorstellungen des gesamten Teams zu erlangen.

Die grösste Schwierigkeit bei der Umsetzung der Arbeit war, eine passende Programmstruktur zu etablieren. Obwohl ein ausführlicher Use Case, Mockups und UML-Diagramm erarbeitet wurde, legte die Kombination der verschiedenen Bereiche der Objektorientierten Programmierung Stolpersteine in den Weg. Das Ziel war den Source Code von Beginn an so zu strukturieren, dass im weiteren Verlauf der Arbeit keine Schwierigkeiten respektive Neumodellierungen stattfinden sollten. Im Bereich JavaFx war die Implementierung zu Beginn übersichtlich und funktionierte einwandfrei. Mit der Zeit wurde der Code unübersichtlich und es fiel deshalb umso schwieriger den Ablauf nachzuvollziehen respektive neue Elemente einzubauen.

Trotz den Stolpersteinen ist das Ergebnis sehr positiv zu betrachten, denn die Integrationstests haben gezeigt, dass die Applikation wie vorgesehen funktioniert. Die Eingabe der variablen Parameter (Spieleinstellung) werden eingelesen und korrekt verarbeitet, gespielt werden kann im Single- und Multiplayer Modus, die Wechsel zwischen den Benutzeroberflächen funktionieren einwandfrei, der Spielverlauf verläuft ohne jegliche Fehlverhalten und schliesst eine fehlerhafte Bedienung des Benutzers aus, die Gestaltung der Benutzeroberflächen und die Musik/Sound Effekte führen zu einer ansprechenden Spielatmosphäre und zudem kann auf Wartezeiten während dem Spiel verzichtet werden aufgrund hoher Code Qualität.

Zusammengefasst wurde gemäss der Projektskizze gearbeitet und alle relevanten Anforderungen implementiert. Das Ziel einer lauffähigen und ansprechenden Applikation wurde somit erreicht. Optionale Features wie Spielerdaten Aufzeichnung oder Auswahl der Panzergrösse sind aus zeitlichen Gründen nicht Teil der Applikation, jedoch sind diese Features aufgrund einer gut durchdachten Zeitplanung mit "COULD" priorisiert worden und somit nicht zur Erreichung des Zieles notwendig sind.

Zukünftig, um die Applikation weiter zu entwickeln, können die oben genannten optionalen Features implementiert werden. Dadurch kann die Spielumgebung erweitert und somit der Spassfaktor erhöht werden. Im Bereich JavaFx wurde gegen das Ende der Code etwas unübersichtlich. Eine Lösung dazu wäre die Variante des Model View Controller, welcher die Übersicht des Source Codes deutlich verbessern würde. Im MVC wird das Modell (die Daten,

die dargestellt werden sollen), die Präsentation (View, Darstellung) und die Steuerung (Controller, verwaltet das Modell und die Darstellung) aufgeteilt.

Abkürzungsverzeichnis

GUI	Graphical User Interface
UML	Unified Modeling Language
IDE	Integrated Development Environment
MVC	Model View Controller

Literaturverzeichnis

Jet Brains. (2008). *Key IntelliJ IDEA Facts*.
Schwaber, K., & Sutherland, J. (2017). *Scrum Guide*.

Abbildungsverzeichnis

Abbildung 1: Gantt-Diagramm für die Zeitplanung des Projekts.....	5
Abbildung 2: Use Case der Applikation Tank Wars	7
Abbildung 3: Benutzeroberfläche Spieleinstellungen	8
Abbildung 4: Benutzeroberfläche Mehrspieler-Modus (optional).....	8
Abbildung 5: Benutzeroberfläche Spielverlauf	9
Abbildung 6: Kanban-Board auf GitHub Repository	11
Abbildung 7: UML Modellierung Abstrakte Klasse Player	12
Abbildung 8: UML Modellierung – Attacke	12
Abbildung 9: Spielverlauf Einzelspieler-Modus	13
Abbildung 10: Spielverlauf Mehrspieler-Modus.....	14
Abbildung 11: UML-Modellierung GameSequencer	14
Abbildung 12: Aufbau der graphischen Benutzeroberflächen	16
Abbildung 13: Prozentuale Anzeige der UNIT-Test Abdeckung.....	17

Tabellenverzeichnis

Tabelle 1: Funktionale User Stories.....	10
Tabelle 2: Nicht funktionale User Stories.....	10