|  |
| --- |
| Ähnliches Foto |
|  |
| **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 (1 Seite)

xxx

Inhaltsverzeichnis

[Management Summary 1](#_Toc29735236)

[1 Einleitung (1 Seite, Ziel erwähnen) 3](#_Toc29735237)

[2 Ausgangslage (1 Seite) 4](#_Toc29735238)

[3 Hauptteil (10 Seiten) 5](#_Toc29735239)

[3.1 Hauptteil Unterkapitel 5](#_Toc29735240)

[4 Fazit und Schlussfolgerung (1- 2 Seiten) 6](#_Toc29735241)

[Abkürzungsverzeichnis 7](#_Toc29735242)

[Literaturverzeichnis 7](#_Toc29735243)

[Abbildungsverzeichnis 7](#_Toc29735244)

[Tabellenverzeichnis 7](#_Toc29735245)

[Anhänge 7](#_Toc29735246)

# Einleitung (1 Seite, Ziel erwähnen)

xxx

# Ausgangslage (1 Seite)

Xxx

## Gantt-Diagram

xxx

## Beschreibung Ausgangslage

# Hauptteil (10 Seiten)

xxx

## Projektskizze Use Case

Xxx

## Projektskizze Mockups

Xxx

## Vorangehensweise (user storys, kanban, branch-Darstellung)

xxx

## 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.



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.



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 den Gegner ausgeführt, wird die Position der Attacke entsprechend gesetzt und den Status auf «ATTACKING» gesetzt. Der Gegner wertet die Attacke anhand der der Koordinaten aus, verändert anschliessend den Status der Attacke auf «SUCCESSFUL», wenn ein Panzer getroffen wurde oder «UNSUCCESSFUL», falls danebengetroffen 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 auch über eine Client/Server Verbindung geschickt werden kann.



Wird das Spiel im Einzelspieler Modus gestartet, wird ein virtueller Gegenspieler erstellt, wobei währen 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 triff 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 und returniert diese. Anschliessend wird geprüft, ob diese zufällige Attacke einen eigenen Panzer trifft oder nicht. Dementsprechend wird das Ergebnis auf dem GUI angezeigt.



Sollte das Spiel im Mehrspieler Modus gestartet werden, wird ein realer Gegenspieler 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.

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 nun den ersten Zug machen darf. Um dies zu garantieren wurde eine Klasse 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 |
| 1. Warten bis alle eigenen Panzer platziert wurden |
| 1. Ist Mehrspielermodus aktiv?   Nein 🡪 Starte Spiel mit eigenem Zug  Ja 🡪 Verbindung zum Gegner prüfen |
| 1. Kann eine Verbindung zum Gegner hergestellt werden.   Nein 🡪 Warte auf Verbindung  Ja 🡪 Starte Spiel mit eigenem Zug |
| 1. Warten, bis der Gegner sich verbindet. Sobald Verbindung akzeptiert wurde, wird Spiel mit gegnerischem Zug gestartet |
| 1. Eigener Zug (user input) |
| 1. Wurde Spiel nach eigenem Zug gewonnen?   Nein 🡪 Gegner ist am Zug  Ja 🡪 Spiel ist zu Ende |
| 1. Gegnerischer Zug |
| 1. Wurde Spiel nach gegnerischem Zug verloren?   Nein 🡪 setze das Spiel mit eigenem Zug fort.  Ja 🡪 Spiel ist zu Ende |
| 1. Ende des Spiels, verlassen der Dauerschleife |

## Realisierung JavaFX-Darstellung (Stage, Scene, x-Pane..)

xxx

## Unit Tests

xxx

## Code Diverses ( Java, JavaFX)

xxx

# Fazit und Schlussfolgerung (1- 2 Seiten)

Ende des ersten Semesters in unserem Nachdiplomstudium HF Applikationsentwicklung absolvieren wir eine Praxisarbeit, mit der wir das erste Modul abschliessen. Die erlernten Kenntnisse aus dem Unterricht konnten wir nun nutzen, um ein erstes Objektorientiertes Programm selbst zu entwickeln.

In einem ersten Schritt haben wir die Projektskizze erarbeitet, bei der wir auch User Stories einbinden mussten. 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.

Eines der grössten Schwierigkeiten bei der Umsetzung der Arbeit war es, eine passende Programmstruktur zu finden. Im vergangenen Semester hatten wir diverse Unterrichtsthemen wie Git, Unit Test, Client/Server, Parallele Programmierung, I/O und Grafische Benutzeroberfläche. Jedoch hatten wir nie mit einem Projekt zu tun, bei dem alle Themen kombiniert werden. Dies erhöht zum einen die Komplexität, jedoch stellt sich auch die Frage nach der Gliederung des Programms. Welche Aktionen sollen in JavaFX gelöst werden welche besser nicht? Welche Attribute einer Klasse sollten statisch sein, und welche nicht? Wie soll eine Grafische Benutzeroberfläche gegliedert werden? Da wir keinerlei Erfahrung haben, waren dies schwierige Fragen, die wir uns stellen mussten.

Wir denken jedoch, gerade weil wir auf solche Fragen gestossen sind, brachte uns dies persönlich weiter und weckte das Verständnis für die erlernte Theorie. Uns ist bewusst, dass wir die Programmstrukturierung noch weiter optimieren können, jedoch konnten wir dies aufgrund des Umfangs des Projekts nicht mehr realisieren.

Abkürzungsverzeichnis

|  |  |
| --- | --- |
| TW | Tank Wars |
| UML | xxxxx |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Literaturverzeichnis

(Kann automatisch eingefügt werden mit Mendeley – Bitte alle Literatur wenn möglich im pdf-format an Hutti senden)

Abbildungsverzeichnis

xxx

Tabellenverzeichnis

xxx

Anhänge

xxx