



WEBE, WEB ENGINEERING

Semesterarbeit 2021

vorgelegt von

Martin Bartolomé und Thomas Curiger

Stand 6. Dezember 2021

Prüfer: Peter Tellenbach

Inhaltsverzeichnis

0.1	Projekttagebuch	4
1	Einleitung	5
1.1	Beschreibung des Spiels	5
1.2	Beschreibung des Kommunikationsprotokolls	6
2	Anforderungskatalog	7
2.1	Funktionale Anforderungen	7
2.2	Qualitative Anforderungen	10
3	Entwurf	13
3.1	GUI	13
3.2	Protokoll Client/Server	14
3.2.1	Definition Enumeratoren	15
3.2.2	Spiel starten	16
3.2.3	Spielfeld aufbereiten	17
4	Umsetzung	18
4.1	GIT-Repository	18
4.2	Netzwerkprotokoll und zweiter Entwurf des Servers	18
4.3	Zweiter Entwurf GUI	21
5	TestProtokoll	22
6	Bedienungsanleitung	23

0.1 Projekttagbuch

Datum	Name	Beschreibung
20.08.2021	Martin Bartolomé	Erstellen des Dokuments und erfassen der Spielbeschreibung und UseCases
13.09.2021	Martin Bartolomé und Thomas Curiger	Sitzung; Protokoll Da Martin Bartolomé beim Kickoff abwesend war, wird der Verlauf
13.09.2021	Thomas Curiger	Erweitern der Use Cases und hinzufügen des Protokolls
20.09.2021	Martin Bartolomé	Einbinden des Projekttagbuchs
10.10.2021	Martin Bartolomé	Erstellen Prototyp GUI
11.10.2021	Martin Bartolomé	Hinzufügen des Protokolls Client/Server sowie Überarbeitung der UseCases
11.10.2021	Martin Bartolomé und Thomas Curiger	Krisensitzung I; Protokoll Bei der Analyse des Meilensteins fiel beiden auf, dass die Kriterien für den Meilenstein zu diesem Zeitpunkt noch nicht erfüllt waren. Thomas Curiger befand sich zusätzlich noch im Ausland, was die Zusammenarbeit erschwerte. Über Remote-Sitzung einigten sich beide auf eine Anfrage für eine Fristverlängerung um 3 Tage, welche von Peter Tellenbach genehmigt wurde.
13.10.2021	Martin Bartolomé und Thomas Curiger	Krisensitzung II; Protokoll Zwei Tage später wurde die Lage nochmals besprochen und die beim Meilenstein erforderlichen Punkte konnten während den zwei Tagen entwickelt werden.
1.-7.11.2021	Martin Bartolomé und Thomas Curiger	Erfolgreiche Weiterentwicklung des Protokolls. An der Sitzung vom 7. November wurden jedoch einige Fehler erkannt. So ist während der kollaborativen Entwicklung ein Chaos in der Server-Client-Architektur entstanden.
22.11.2021	Martin Bartolomé und Thomas Curiger	Krisensitzung III; Protokoll Die beiden Entwickler einigten sich auf eine Restrukturierung des Codes, um ihn einfacher wartbar zu machen. So wurde die gesamte Architektur in viele neue Klassen und Methoden unterteilt, anstatt alles in einem Objekt abzuspeichern. Die Änderungen an der Architektur wurden erfasst und in der Dokumentation notiert.

1 Einleitung

1.1 Beschreibung des Spiels

Bei dem zu programmierenden Spiel handelt es sich um ein „Tower Defense“ Spiel. Hierfür werden verschiedene Karten gezeichnet. Diese Karten enthalten Wege, welche gegnerische Einheiten entlang laufen. Diese Einheiten werden von einem Server kontrolliert. Der Spieler hat hierbei eine bestimmte Währung zur Verfügung um Einheiten oder Türme am Rand des Weges aufzustellen. Diese Einheiten haben eine gewisse Angriffskraft, um die gegnerischen Einheiten davon abzuhalten, das andere Ende des Weges zu erreichen. Erreicht eine gegnerische Einheit die andere Seite des Weges, so wird den Spielern eine ein Leben abgezogen. Haben die alle Spieler keine Leben mehr, wird das Spiel beendet. Wird ein Gegner besiegt, so erhält der Spieler, welcher den Gegner besiegt hat einen bestimmten Betrag der Währung und alle anderen Spieler einen gewissen prozentualen Wert an Währung.

Das Spiel ist für maximal 4 Spieler gedacht, wobei jeder Spieler einen bestimmten Bereich des Weges abdeckt. Gibt es weniger Spieler, so wächst der Sektor der Spieler, die anwesend sind. Das Spiel ist vorbei, sobald die Spieler die drei bestehenden Level absolviert haben oder alle Leben aufgebraucht sind. Verlässt ein Spieler das Spiel während es läuft, so ist es nicht mehr möglich erneut beizutreten. Der Bereich des Spielers wird gesperrt und die anderen Spieler erhalten mehr Einheiten der Währung, wenn eine gegnerische Einheit vernichtet wird. Eine grobe Skizze ist in Abbildung 1.1 zu sehen. Hier starten die Gegner in der unteren Linken Ecke und verfolgen den Weg bis an das Ende. Der Spielbereich ist in 4 Sektoren aufgeteilt.

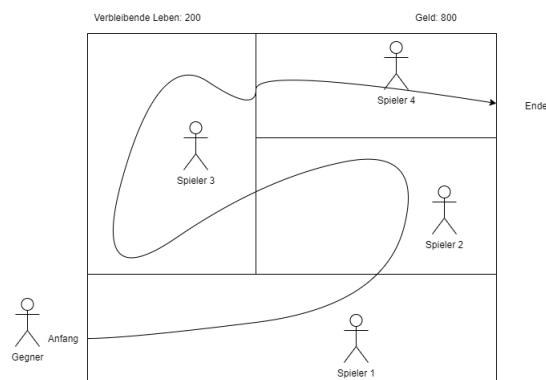


Abbildung 1.1: Skizze des Spielbereichs

1.2 Beschreibung des Kommunikationsprotokolls

Die Kommunikation zwischen Client und Server basiert auf dem Protokoll WebSocket. WebSocket befindet sich wie Http auf dem siebten Layer des OSI-Modells und verwendet TCP auf dem vierten Layer zum Senden und Empfangen.

Im Gegensatz zu Http unterstützt WebSocket das gleichzeitige Senden und Empfangen von Paketen. Http sendet jeweils einzelne Request, weswegen dieses Protokoll für ein Real-Time Multiplayer-Spiel nicht in Frage kommt.

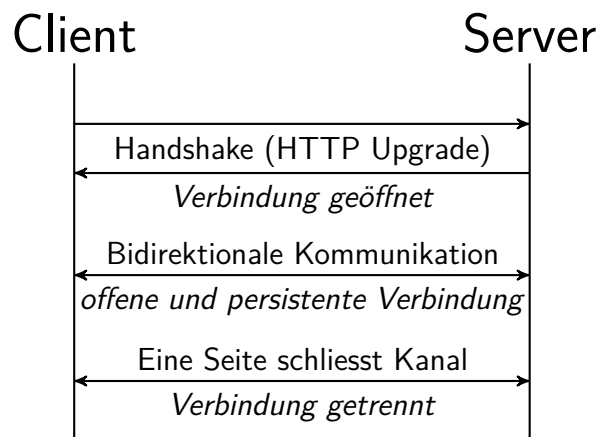


Abbildung 1.2: WebSocket Protokoll

2 Anforderungskatalog

In diesem Kapitel werden die funktionalen sowie die qualitativen Anforderungen beschrieben.

2.1 Funktionale Anforderungen

Die anschliessend aufgelisteten Use Cases beschreiben die funktionalen Anforderungen und sind auf jeden Fall umzusetzen.

Nr.	F1
Name	Start des Spiels
Ziel	Das Spiel wird gestartet.
Vorbedingung	Browser ist geöffnet.
Nachbedingung	Spiel wird angezeigt.
Akteure	Spieler
Trigger	Eingabe der korrekten URL.

Tabelle 2.1: F1 - Start des Spiels

Nr.	F2
Name	Start einer Session
Ziel	Session wurde mit Karte und Namen gestartet. Der Spieler hat einen Namen ausgewählt und der Wartebildschirm wird angezeigt
Vorbedingung	Das Spiel wurde gestartet und es wurden Information wie der Sessionname, Level und Spielernamen eingegeben.
Nachbedingung	Session gestartet und es wird auf Spieler gewartet
Akteure	Spieler
Trigger	Form mit Button zum Start der Session

Tabelle 2.2: F2 - Start eines Servers

Nr.	F3
Name	Beitreten einer Session
Ziel	Spieler ist einem anderen Spiel beigetreten
Vorbedingung	Spiel gestartet und ein Spielernamen wurde eingegeben
Nachbedingung	Spieler ist einem anderen Spiel beigetreten
Akteure	Spieler
Trigger	Selektierte Session und Button zum Beitreten

Tabelle 2.3: F3 - Beitreten eines Servers

Nr.	F4
Name	Platzieren von Einheiten und Türmen
Ziel	Einheit oder Turm wurde im Spiel platziert
Vorbedingung	Spiel gestartet und Runde hat begonnen
Nachbedingung	Einheit oder Turm wurde platziert
Akteure	Spieler
Trigger	Einheit oder Turm ausgewählt und im Spielbereich platziert

Tabelle 2.4: F4 - Platzieren von Einheiten und Türmen

Nr.	F5
Name	Verdienen von Währung
Ziel	Währung wird dem Spieler gutgeschrieben
Vorbedingung	Spieler hat eine Einheit platziert und Gegner wurde vernichtet
Nachbedingung	Währung wurde gutgeschrieben
Akteure	Einheit des Spielers
Trigger	Angriff einer Einheit oder Turms

Tabelle 2.5: F5 - Verdienen von Währung

Nr.	F6
Name	Spenden von Währung an anderen Spieler
Ziel	anderer Spieler hat Währung erhalten
Vorbedingung	Spieler möchte Währung spenden
Nachbedingung	Währung wurde gespendet
Akteure	Spieler
Trigger	Button im Spiel

Tabelle 2.6: F6 - Spenden von Währung an anderen Spieler

Nr.	F7
Name	Sehen der anderen Spieler
Ziel	Spieler sieht andere Spieler
Vorbedingung	Session wurde gestartet
Nachbedingung	Spieler sieht andere Spieler
Akteure	Spieler
Trigger	Session gestartet und andere Spieler sind beigetreten

Tabelle 2.7: F7 - Sehen der anderen Spieler

Nr.	F8
Name	Chat mit Spieler
Ziel	Spieler können sich neben dem Spiel unterhalten
Vorbedingung	Session gestartet
Nachbedingung	Nachricht gesendet
Akteure	Spieler
Trigger	Input und Button zum Senden

Tabelle 2.8: F8 - Chat mit Spieler

Nr.	F9
Name	Internationalisierung
Ziel	Das Spiel muss eine Sprachauswahl mit mehreren Fremdsprachen unterstützen.
Vorbedingung	Spiel wird angezeigt
Nachbedingung	Spiel wird in der ausgewählten Sprache angezeigt.
Akteure	Spieler
Trigger	Sprache wird vom Spieler neu gesetzt oder vom Browser angegeben.

Tabelle 2.9: F9 - Internationalisierung

Nr.	F10
Name	Responsiveness
Ziel	Das Spiel sollte an die Bildschirmgröße des Gerätes angepasst.
Vorbedingung	Browser ist geöffnet.
Nachbedingung	Der Spielbereich wird in der richtigen Größe angezeigt.
Akteure	Spieler
Trigger	Eingabe der korrekten URL.

Tabelle 2.10: F10 - Responsiveness

Nr.	F11
Name	Validierung
Ziel	Alle Spielzüge müssen vom Server erfasst und auf Regelkonformität geprüft werden.
Vorbedingung	Das Spiel wurde gestartet.
Nachbedingung	Spielaktion wird ausgeführt.
Akteure	Spieler
Trigger	Spielaktion wird eingegeben.

Tabelle 2.11: F11 - Validierung

2.2 Qualitative Anforderungen

Die folgenden qualitativen Anforderungen sind während der Entwicklung zu optimieren.

Nr.	Q1
Name	Wartbarkeit
Ziel	Jede Komponente der Anwendung muss unabhängig von den anderen austauschbar, wartbar und erweiterbar sein.
Definition	Die Komponenten der Anwendung müssen modular aufgebaut sein. Jede Komponente ist nur für eine Aufgabe zuständig. Eine Änderung an einer Komponente hat keine Auswirkung auf die Funktionsweise der anderen Komponenten, sofern die Schnittstelle nicht angepasst wurde. Neue Komponenten können die bestehenden Schnittstellen nutzen. Damit ist es möglich, einzelne Komponenten anzupassen, auszutauschen oder zu erweitern.
Messverfahren	Bei einer Änderung muss nur an einer Komponente etwas geändert werden ohne dass es Abhängigkeiten zu anderen Komponenten gibt.
Konsequenz	Werden Abhängigkeiten festgestellt, müssen diese bei der Anpassung, beim Austausch oder bei der Erweiterung entfernt oder wenn nicht möglich dokumentiert werden.

Tabelle 2.12: Q1 - Wartbarkeit

Nr.	Q2
Name	Latenz
Ziel	Die Latenz soll während des Spiels möglichst tief sein.
Definition	Die Latenz wird durch die verschiedene Faktoren beeinflusst. So erhöhen eine langsame Internetverbindung, laufende Firewalls sowie die Distanz zwischen Client und Server die Latenz und führen im schlimmsten Fall zu Framedrops.
Messverfahren	Die Latenz kann mit verschiedenen Browsertools getestet werden.
Konsequenz	Ist die Latenz zu hoch, muss eine grundlegende Fehlersuche durchgeführt werden.

Tabelle 2.13: Q2 - Latenz

Nr.	Q3
Name	Framerate
Ziel	Die Framerate sollte während dem Spiel konstant hoch sein.
Definition	Die Framerate des Spiels hängt von der Latenz und der Grafikoptimierung des Spiels ab. Die optimale Anzahl für Web-Anwendungen liegt bei 60 Frames pro Sekunde (fps).
Messverfahren	Die Framerate kann mit verschiedenen Browsertools getestet werden.
Konsequenz	Ist die Framerate zu tief oder werden gar Framedrops erkannt, muss eine grundlegende Fehlersuche durchgeführt werden.

Tabelle 2.14: Q3 - Framerate

Nr.	Q4
Name	Farboptimierung
Ziel	Die Farbauswahl sollte optimiert werden.
Definition	Um die Usability und Accessibility auch für Nutzer mit einer Sehschwäche zu verbessern, soll die Farbauswahl auf klaren Kontrasten basieren.
Messverfahren	Die Farbauswahl kann mit einem WCAG-Color-Checker getestet werden. Die Farben und Kontraste der Applikation sollten das WCAG AA-Level erfüllen.
Konsequenz	Wird das WCAG AA-Level nicht erfüllt, müssen die Farben der Game Sprites und der Texte angepasst werden.

Tabelle 2.15: Q4 - Farboptimierung

Nr.	Q5
Name	Verfügbarkeit
Ziel	Das Spiel sollte immer via Internet verfügbar sein..
Definition	Die Serverarchitektur muss so aufgebaut sein, dass es zu keinen Serviceunterbrüchen kommen kann.
Messverfahren	Die Verfügbarkeit kann mit Stresstests überprüft werden. Während des Testens sollten viele Clients gestartet werden, welche zur exakt gleichen Zeit auf das Spiel zugreifen.
Konsequenz	Kann die Serverarchitektur der Testbelastung nicht standhalten, muss diese überprüft und allenfalls überarbeitet werden.

Tabelle 2.16: Q5 - Verfügbarkeit

3 Entwurf

In diesem Kapitel werden die ersten Entwürfe des Spiels erstellt. Dabei geht es um Entwürfe der Benutzeroberfläche, des Client-/Server-Protokolls und allfällige weitere Entwürfe im Bereich der Architektur.

3.1 GUI

In diesem Abschnitt wird ein erster, grober Entwurf für einen grösseren Bildschirm (PC) im horizontalen Format entwickelt. Der Entwurf wird vermutlich zu einem späteren Zeitpunkt nochmals überarbeitet.

Auf der Startseite wird eine Liste aller aktuell laufenden Spiele angezeigt. Unter diesen Listen gibt es einen Button, mit dem es möglich ist einem Spiel beizutreten. Ebenfalls gibt es die Möglichkeit ein neues Spiel zu starten, wobei hier der Name des Spiels sowie das Level vorher ausgewählt werden müssen. Sobald ein Spiel gestartet wurde, erscheint ein Wartebildschirm, wo der Spieler auf Mitspieler warten kann. Der Spieler kann hier bis auf 3 weitere Spieler warten oder vorher durch einen Klick auf Start das Spiel starten.

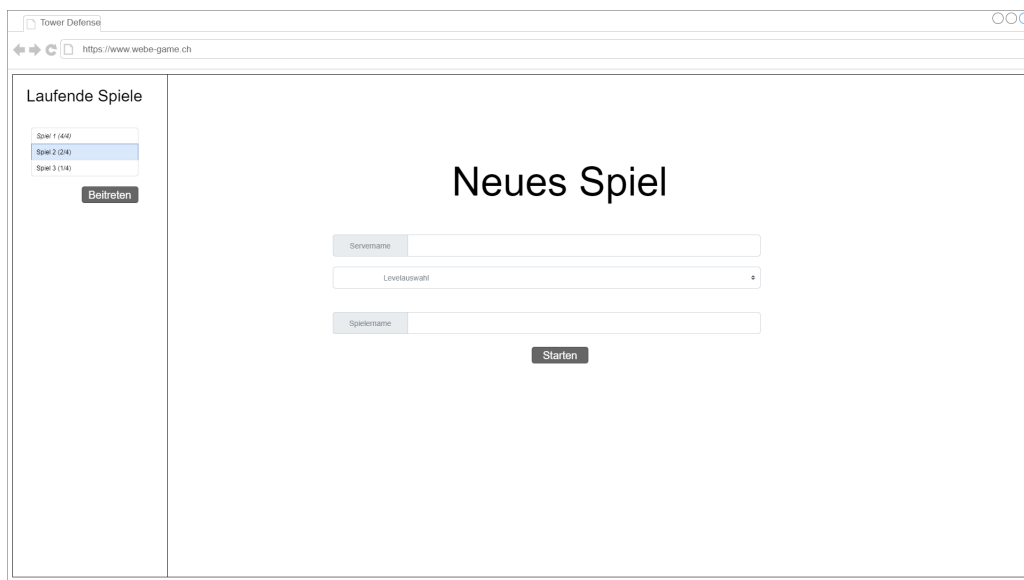


Abbildung 3.1: Startseite

Ist das Spiel gestartet wird das Fenster in 3 Bereiche aufgeteilt. Den grössten Bereich benötigt hier das Spielfeld, denn es ist wichtig, dass der Spieler einen Überblick über das

ganze Spielfeld erhält. In der rechten oberen Ecke sind alle zur Verfügung stehenden Türme aufgelistet. Diese können einfach per Drag & Drop oder per ziehen mit dem Finger ins Spielfeld platziert werden. In der unteren rechten Ecke werden alle Nachrichten angezeigt, sowie gibt es eine TextBox, mit dem der Benutzer selbst Nachrichten senden kann.

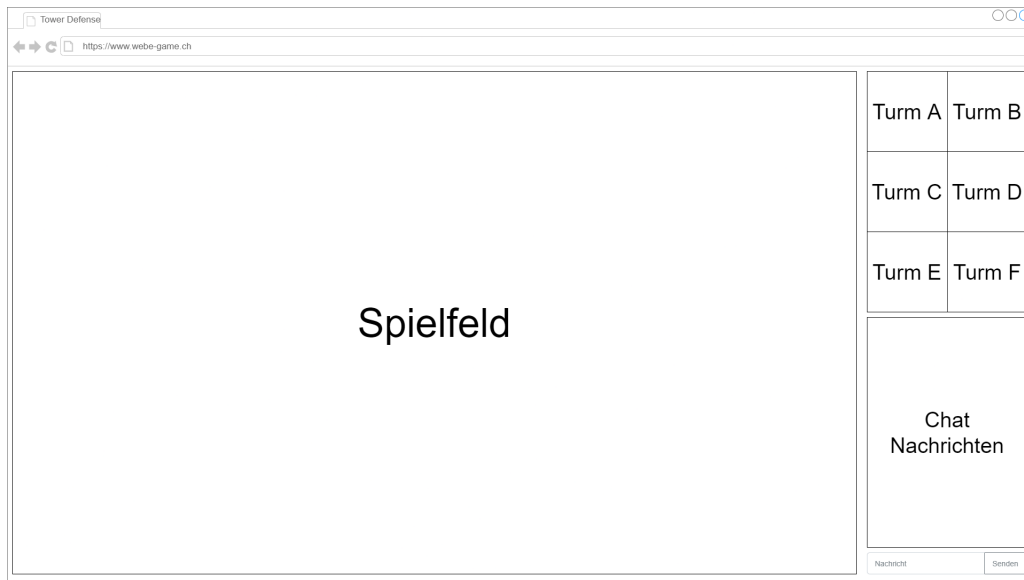


Abbildung 3.2: Spielbereich

3.2 Protokoll Client/Server

In den folgenden Kapiteln werden die Datenstrukturen der Objekte für den Datenaustausch zwischen Client und Server spezifiziert. Geplant ist, diese Daten als JSON-Objekte über das WebSocket-Protokoll zwischen Client und Server zu verschicken. Das WebSocket-Protokoll eignet sich insbesondere für die Echtzeit-Kommunikation zwischen Client und Server und ermöglicht eine bidirektionale Verbindung. Wir gehen deshalb davon aus, dass dies für ein interaktives Spiel und einen Chat, wo niedrige Latenzzeiten erwünscht sind, eine sinnvolle Wahl ist.

Zur Serverfunktionalität gehören folgende Anforderungen:

Aufstarten / Create Der Server muss erkennen, wenn genug Spieler für ein Spielstart bereit sind und das Spiel starten.

Verlauf / Run Der Spielverlauf muss verwaltet und gespeichert werden. Die Eingaben müssen genau überprüft werden.

Schliessen / Close Ist das Spiel vorbei muss ein Countdown beginnen, bis die Verbindung mit den Spielern getrennt wird. Verlassen die Spieler das Spiel vorher, muss der Server dies erkennen können.

Zur Clientfunktionalität gehören folgende Anforderungen:

Eingaben erkennen Der Client muss die Eingaben des Nutzers erkennen und an den Server senden.

Statusabgleichung Der Client muss überprüfen, dass der Status mit demjenigen des Servers übereinstimmt.

Chatfunktion Gemäss Anforderungen muss die Chatfunktion im Client implementiert sein.

3.2.1 Definition Enumeratoren

ObjectType - Objekt Typ		
Enumerator	Wert	Beschreibung
Undefined	0	Undefiniert
StartGame	1	Spiel starten
GameBoardSetup	2	Spielfeld aufbereiten
Playerplacing	3	Spieleraktion
EnemyState	4	Status eines Gegners
Request	5	Anfrage
Response	6	Antwort
WaitStateInvoke	7	Wartezustand aufrufen
ChatLogEntry	8	Chat Log Eintrag
ReadGameLogBook	9	Lese Logbuch eines SPIeles
GameLogBook	10	Logbuch eines Spieles
GameLevelScore	11	Punktestand aktuelles Level
ServerNotification	12	Server Broadcast Notifikation

TextKey - Texte		
Enumerator	Wert	Beschreibung
Undefined	0	?
NewGame	1	Neues Spiel
WaitForPlayers	2	Warten auf weitere Mitspieler
ConnectionLost	3	Verbindung zum Mitspieler verloren
GameAborted	4	Spiel vom Mitspieler abgebrochen
Win	5	Spiel Gewonnen
Lost	6	Spiel Verloren

Type - Art der Anforderung		
Enumerator	Wert	Beschreibung
Undefined	0	Undefiniert
Information	1	Information
Warning	2	Warnung
Exception	3	Ausnahme

3.2.3 Spielfeld aufbereiten

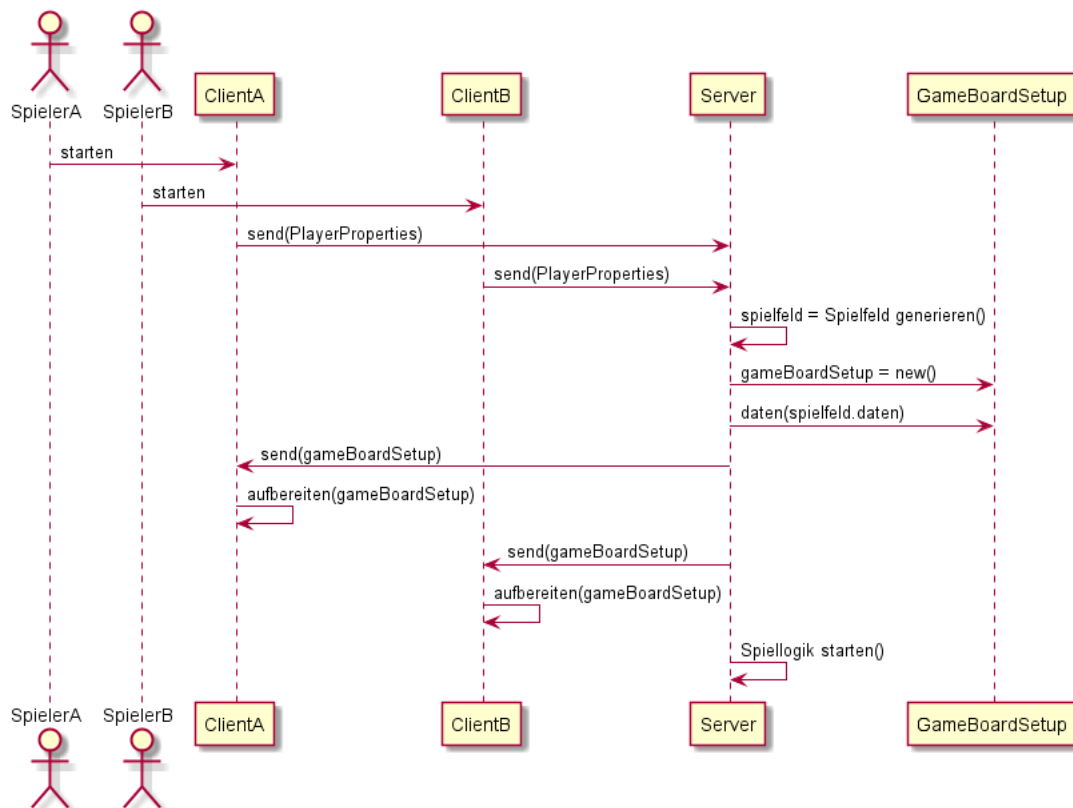


Abbildung 3.4: Spielfeld aufbereiten

GameBoardSetup			
Name	Typ	Beschreibung	Wert
ObjectType	Enum	Objekt Typ	GameBoardSetup
Player	Array of PlayerProperties		-
GameLevel	Enum	Spiellevel	GameLevel

4 Umsetzung

Im folgenden Kapitel wird beschrieben, wie das Projekt umgesetzt wurde.

4.1 GIT-Repository

Der Quellcode befindet sich eingeecheckt unter folgenden Links des GitLab der FFHS:
<https://git.ffhs.ch/martin.bartolome/webe-project.git>

4.2 Netzwerkprotokoll und zweiter Entwurf des Servers

Das Serverprotokoll wurde etwa wie geplant umgesetzt. Die Messagetypen werden so übergeben:

Messagetypen		
Enumerator	Wert	String
REGISTER	0	'register'
CHAT	1	'chat'
GAMEUPDATE	2	'gameupdate'
GAMESTART	3	'gamestart'
GAMESTOP	4	'gamestop'

Bei einer Spielaktion wird vom Client eine Message mit der createMessage-Funktion versendet. Dieser verarbeitet die Message wie hier dargestellt:

```
socket.onmessage = function (event) {
  try {
    this.message = new Message.Message();
    this.message.fromStream(event.data);
    switch (this.message.messageType) {
      case Message.MessageType.CHAT:
        let chatMessage = new ChatMessage();
        chatMessage.fromStream(event.data);
        broadcast(chatMessage);
        break;
      case Message.MessageType.REGISTER:
        let regmsg = new RegisterMessage();
        regmsg.fromStream(event.data);
        gameStatus.registeredPlayers.set(socket.id, regmsg.playerID);
        break;
      case Message.MessageType.GAMEUPDATE:
        let updatemessage = new GameUpdateMessage();
        updatemessage.fromStream(event.data);
        if(updatemessage.updateType == UpdateType.Tower)
        {
          broadcast(updatemessage);
        }
        if(updatemessage.updateType == UpdateType.Wave)
        {
          gameStatus.canSpawn = true;
        }
        break;
      default:
        console.log("[MESSAGE.WARNING] Client doesn't expect this message: " + data);
        break;
    }
  }
}
```

Abbildung 4.1: Messageverarbeitung

Nach der Verarbeitung werden verschiedene Operationen durchgeführt und an die Clients gebroadcastet.

```
function broadcast(data) {
  server.clients.forEach(client => {
    if (client.readyState === WebSocket.OPEN) {
      client.send(data.toStream());
    }
  });
}
```

Abbildung 4.2: Serverbroadcast an Clients

Die Clients nehmen anschliessend die Message auf und verarbeiten diese wiederum.

```

websocketGame.socket.onmessage = function (event) {
  try {
    this.message = new window.Message();
    this.message.fromStream(event.data);
    let data = JSON.parse(event.data);
    switch (data.messageType) {
      case messageType.CHAT:
        chatLogEntry(data);
        break;
      case messageType.GAMESTART:
        let gamestartmessage = new window.GameStartMessage();
        gamestartmessage.fromStream(event.data);
        level = [];
        level = gamestartmessage.Level.level;
        gameLoop();
        websocketGame.running = true;
        break;
      case messageType.GAMEUPDATE:
        let gameupdatemessage = new window.GameUpdateMessage();
        gameupdatemessage.fromStream(event.data);
        if(gameupdatemessage.updateType == UpdateType.Tower)
        {
          towers = [];
          for (var y = 0; y < gameupdatemessage.UpdateObject.length; y++) {
            towers.push( ...
              gameupdatemessage.UpdateObject[y].upgrade));
          }
        }
        if(gameupdatemessage.updateType == UpdateType.Level) ...
        }
        if(gameupdatemessage.updateType == UpdateType.Player) ...
        }
        if(gameupdatemessage.updateType == UpdateType.Wave)
        {
          wave.push( ...
            gameupdatemessage.UpdateObject.genre));
        }
        break;
      case messageType.SHOT:
        renderHit(data);
        break;
      case messageType.GAMESTOP:
        websocketGame.running = false;
        reset();
        removeCanvas();
        break;
      default:
        console.log(
          "[MESSAGE.WARNING] Client doesn't expect this message: " + data
        );
        break;
    }
  }
}

```

Abbildung 4.3: Client Messageverarbeitung

Handelt es sich um ein Gameupdate, wird nun eine Gameupdatesfunktion auf Clientseite aufgerufen, welche die jeweilige Karte aktualisiert. Das Kartenobjekt diente ursprünglich

auch der Spielverwaltung und sah zu Beginn so aus:

```
var map = {
  cols: 12,
  rows: 12,
  tsize: 64,
  tiles: [
    [119, 119, 119, 60, 119, 119, 119, 119, 119, 119, 119, 119],
    [119, 119, 119, 60, 119, 119, 119, 119, 119, 119, 119, 119],
    [119, 119, 60, 60, 119, 119, 119, 119, 119, 119, 119, 119],
    [119, 119, 60, 119, 119, 119, 119, 119, 119, 119, 119, 119],
    [119, 119, 60, 119, 119, 119, 119, 119, 119, 119, 119, 119],
    [119, 119, 60, 119, 119, 119, 119, 119, 119, 119, 119, 119],
    [119, 119, 60, 60, 60, 119, 119, 119, 119, 119, 119, 119],
    [119, 119, 119, 119, 60, 119, 119, 119, 119, 119, 119, 119],
    [119, 119, 119, 119, 60, 60, 119, 119, 119, 119, 119, 119],
    [119, 119, 119, 119, 119, 60, 119, 119, 119, 119, 119, 119],
    [119, 119, 119, 119, 119, 60, 119, 119, 119, 119, 119, 119],
    [119, 119, 119, 119, 119, 60, 119, 119, 119, 119, 119, 119]
  ],
  getTile: function (col, row) {
    return this.tiles[row][col];
  },
  // 245-248
  enemiesMoving: [],
  towersAlive: []
};
```

Abbildung 4.4: Spielverarbeitung mittels Kartenobjekt

Da die

4.3 Zweiter Entwurf GUI

Das GUI wurde aufgrund der Vereinfachung des Spielmodus angepasst:


 images/screenshot_gui_2.PNG

Abbildung 4.5: GUI Stand Anfangs November

5 TestProtokoll

6 Bedienungsanleitung

Abbildungsverzeichnis

1.1	Skizze des Spielbereichs	5
1.2	WebSocket Protokoll	6
3.1	Startseite	13
3.2	Spielbereich	14
3.3	Spieler Anmeldung	16
3.4	Spielfeld aufbereiten	17
4.1	Messageverarbeitung	19
4.2	Serverbroadcast an Clients	19
4.3	Client Messageverarbeitung	20
4.4	Spielverarbeitung mittels Kartenobjekt	21
4.5	GUI Stand Anfangs November	21

Tabellenverzeichnis

2.1	F1 - Start des Spiels	7
2.2	F2 - Start eines Servers	7
2.3	F3 - Beitreten eines Servers	8
2.4	F4 - Platzieren von Einheiten und Türmen	8
2.5	F5 - Verdienen von Währung	8
2.6	F6 - Spenden von Währung an anderen Spieler	8
2.7	F7 - Sehen der anderen Spieler	9
2.8	F8 - Chat mit Spieler	9
2.9	F9 - Internationalisierung	9
2.10	F10 - Responsiveness	9
2.11	F11 - Validierung	10
2.12	Q1 - Wartbarkeit	10
2.13	Q2 - Latenz	11
2.14	Q3 - Framerate	11
2.15	Q4 - Farboptimierung	11
2.16	Q5 - Verfügbarkeit	12