Schulprojekt Winter 2018

Fachinformatiker für Anwendungsentwicklung

Dokumentation zur schulischen Projektarbeit

Entwicklung einer Android Applikation

Entwicklung des Spiels „Dino Jousting“ als Prototyp für zukünftige Projekte

Abgabedatum: 21.12.2018

Schüler:

Gar Georg <Georg.Gar@cerner.com>

Fedorak Wilhelm <Willi.Fedorak@cerner.com>

Ausbildungsbetrieb:

Cerner Health Services Deutschland GmbH

Hadersberg 1

84427 St. Wolfgang

1. Inhaltsverzeichnis

1. Inhaltsverzeichnis 1
2. Projektauftrag 2
   2.1. Projektumfeld 2
   2.2. Projektziel 2
   2.3. Projektabgrenzung 2
3. Projektplanung 3
   3.1. Beschreibung des Ist-Zustandes 3
   3.2. Beschreibung des Soll-Konzepts 3
   3.3. Vorgehensmodel 4
   3.4. Zeitplanung 4
4. Projektdurchführung 5
   4.1. Entwurfsphase 5
      4.1.1. Spielkonzept 5
      4.1.2. User Interface 6
      4.1.3. Datenbankmodell 6
      4.1.4. Client-Server-Architektur 6
   4.2. Implementierung 6
      4.2.1. Datenbank 6
      4.2.2. Serverseite 6
      4.2.3. Clientseite 10
   4.3. Qualitätssicherung 13
5. Projektabschluss 14
   5.1. Soll-Ist-Vergleich 14
   5.2. Fazit 14
   5.3. Ausblick 15
6. Anhang 16
7. Glossar 23
8. Quellen 23

2. Projektauftrag

2.1. Projektumfeld

Im Rahmen eines Schulprojektes müssen die Auszubildenden der Berufsschule Freising in Zweiergruppen eine App bzw. einen Prototyp programmieren. Dabei dürfen sie selbst entscheiden, ob dies ein Spiel ist, oder nicht.

Für dieses Projekt haben die Schüler Georg Gar und Wilhelm Fedorak sich entschieden, den Prototypen für ein Spiel in Android Studio zu entwickeln. Dieser soll als Basis für zukünftige Entwicklung dienen.

Außerdem soll dadurch weiteres Wissen über die Entwicklung von Android Apps und deren Client-Server-Architektur erlangt werden. Das Projekt wird von Herrn Pfleiderer begleitet und bewertet.

In der Tabelle 1 können genaue Details zu den genutzten Technologien und deren Versionen eingesehen werden.

2.2. Projektziel

Das Ziel dieses Projektes ist es, ein funktionsfähiges Android-Spiel zu erstellen, welches von zwei Spielern an verschiedenen Orten gegeneinander gespielt werden kann. Außerdem soll die Möglichkeit geboten werden, das Spiel auch alleine gegen einen programmierten Gegner zu spielen. Dabei soll vor allem Erfahrung über das Programmieren im Bereich Android gesammelt werden. Das Spiel selbst dient dabei als Basis für zukünftige Ideen und hat daher den Fokus nicht direkt auf das Spielerlebnis. Die App soll einen Rahmen liefern, auf dessen Grundlage mit wenig Aufwand weitere Spielinhalte hinzugefügt werden können. Das Projekt soll alle Abläufe einer gebrauchsüblichen App enthalten. Dazu gehören User-Management mit einer Datenbank, ein autoritärer Game-Server, der den Hauptspielablauf berechnet und client-seitige Darstellung des Spielgeschehens.

2.3. Projektabgrenzung

Der Zeitrahmen für das Projekt läuft vom 14. September 2018 bis zum 21. Dezember 2018.

Am Ende dieses Rahmens soll das Projekt und alle seine Ziele abgeschlossen und dokumentiert sein. Zur besseren Verfolgung des Projektzustandes sollen Milestones mit einem bedingten Abschlussdatum angelegt werden. Diese sind möglichst einzuhalten, um einen erfolgreichen Abschluss des Projektes zu gewährleisten. In Tabelle 2 können alle Ziele und ihr geplantes Abschlussdatum eingesehen werden.

Dieses Projekt ist vorrangig ein technischer Prototyp und soll die Funktionsweisen einer Android-App aufweisen. Dabei wird nicht zu sehr auf das Game-Design geachtet, weshalb Spaß-Faktoren wie etwa Herausforderung und Variation des Spiel-Erlebnisses nicht im Vordergrund stehen. Diese Faktoren sollen in Zukunft jedoch ohne großen Aufwand implementierbar sein, weshalb ein Rahmen für leichte Erweiterbarkeit geschaffen werden soll.

3. Projektplanung

3.1. Beschreibung des Ist-Zustandes

Dieses Projekt ist der erste Prototyp zum Thema Android-Entwicklung. Es gibt daher keine Basis, auf der es aufbauen kann. Außerdem ist noch keine Erfahrung zu diesem Thema vorhanden. Es ist bereits Erfahrung in den Bereichen Datenbanken, Programmieren in Java und JavaScript vorhanden.

3.2. Beschreibung des Soll-Konzepts

Das Spiel soll einen personalisierten Login und eine Registrierungsmöglichkeit für neue Benutzer bereitstellen. Dabei muss sichergestellt sein, dass ein Benutzer sich nicht mehrfach anmelden kann. Es muss die Möglichkeit bereitgestellt werden, alleine gegen den Computer oder gegen einen anderen Spieler zu spielen. Dabei sollen alle erforderlichen Daten wie Benutzernamen, Passwörter und Sieger-Statistiken in einer Datenbank abgespeichert werden. Auf diese soll über ein DAO an nur einer Stelle im Code zugegriffen werden.

Es besteht die Möglichkeit, das Spiel so zu implementieren, dass beide Clients direkt miteinander über Peer-to-Peer kommunizieren und für das Abspeichern von Daten direkten Zugriff auf eine Datenbank haben. Eine weiter Möglichkeit ist es, jeden Client nur mit einem autoritären Server zu verbinden, welcher das Spielgeschehen kontrolliert und alle Datenbankaufrufe durchführt.

Erstere Möglichkeit bietet eine einfache Implementierung und benötigt keine Server-Architektur. Sie ist jedoch anfällig für Manipulation der Spieldaten. Außerdem ist es unumgänglich, dass hierbei die Datenbankzugriffe auf jedem Gerät vorhanden sind, wodurch ein Sicherheitsrisiko für die Datenbank besteht.

Die zweite Möglichkeit bietet mehr Sicherheit der Spieldaten wie auch der Datenbank, da die Clients über diese Daten keine Autorität haben. Sie erhalten lediglich das berechnete Spielgeschehen vom Server und schicken ihre Inputs zu diesem. Dafür muss jedoch ein eigener Web-Server implementiert werden, was mit zusätzlichem Aufwand verbunden ist und dadurch zu Zeitmangel führen könnte.

Da die erste Möglichkeit zu viele Sicherheitsrisiken birgt, wird für das Projekt ein autoritärer Server implementiert. Sowohl das Benutzermanagement als auch das Spielgeschehen werden auf diesem ablaufen.

Aufgrund der vorhandenen Erfahrung wurde entschieden, einen auf Node.js basierenden Web-Server mit einer MySQL-Datenbank zu verwenden.

3.3. Vorgehensmodel

Für dieses Projekt ist das inkrementelle Vorgehensmodell vorgeschrieben. Dies hat die Strategie Systeme nach einem gewissen Ablaufplan iterativ zu integrieren. Dabei sollen agil Entscheidungen getroffen werden können. So können Erfahrungen, die während dem Entwicklungsprozess gemacht werden, direkt angewandt werden und Probleme flexibel gelöst werden. Die einzelnen Schritte des Ablaufs bestehen aus Implementieren, Testen, Abwägen der Ergebnisse und Planen der nächsten Schritte. Diese werden wiederholt, bis das gewünschte Ergebnis erreicht ist.

Dieses Modell bietet den Vorteil, dass Änderungen schnell getestet werden können und dadurch schnelle Entscheidungen bei der Entwicklung getroffen werden können. So ist es möglich, Technologien anzuwenden und, je nach Entscheidung zu nutzen oder zu verwerfen. Dadurch wird ein großer Überblick über die Android-Entwicklung geschaffen.

3.4. Zeitplanung

Für das Projekt stehen 70 Stunden zur Verfügung. Diese wurden auf die verschiedenen Phasen der Projektentwicklung aufgeteilt. Die Aufteilung dieser Zeit kann Tabelle 3 entnommen werden.

Tabelle 3

Projektphase	**Dauer**
Entwurfsphase	8 h
Implementierungsphase	52 h
Erstellen der Dokumentation	10 h
Gesamt	**70 h**

In der Entwurfsphase wird das Spielkonzept ausgearbeitet. Es werden die Haupt-Mechaniken diskutiert und die notwendigen und möglichen Technologien abgewogen.

Darauf folgt die Implementierungsphase, in der nach dem inkrementellen Modell versucht wird, den im Projektziel beschriebenen Zustand zu erreichen.

Letztendlich wird zum Projekt die Dokumentation angelegt.

4. Projektdurchführung

4.1. Entwurfsphase

4.1.1. Spielkonzept

Das Kernprinzip des Spieles ist an das mittelalterliche Tjosten angelehnt. Zwei Ritter stehen sich dabei auf einer Zwei-Dimensionalen Ebene gegenüber. Während sie sich aufeinander zubewegen müssen sie versuchen, ihre Lanze so anzuheben, dass sie den Gegner trifft. Sie beschleunigen während dem Vorgang auf die maximale Geschwindigkeit des gewählten Reittieres. Dieser Ablauf stellt eine Runde dar. Es wird so lang gespielt, bis die Lebenspunkte eines Spielers auf null sinken.

Der gesamte Ablauf wird in 3 Phasen unterteilt:

1. Start:

   Beide Spieler stehen auf der x-Achse mit einem gewissen Abstand voneinander entfernt, dieser wird zufällig aus einem Spektrum gewählt. Ein Countdown läuft ab, nach dem die nächste Phase startet.

2. Bewegung:
   Beide Spieler starten zeitgleich zu beschleunigen. Dabei hängt Maximalgeschwindigkeit und Beschleunigung vom gewählten Reittier ab. In dieser Phase können die Spieler ihre Lanze durch drücken des Bildschirms anheben. Wird dies nicht gemacht, fällt die Lanze langsam. Es soll versucht werden, den Gegner zu treffen.

3. Treffpunkt:
   Die Spieler treffen sich. Abhängig von der Position der Lanze wird das Rundenergebnis berechnet. Dabei ist auch die Höhe der Reittiere der Spieler zu beachten. Es gibt drei Trefferzonen.
   1. Kein Treffer – Die Lanze verfehlt und der Gegner verliert keine Lebenspunkte
   2. Körpertreffer – Die Lanze trifft den Körper und zieht einen kleinen Teil der Lebenspunkte des Gegners ab
   3. Kopftreffer – Die Lanze trifft den Kopf und zieht einen großen Teil der Lebenspunkte des Gegners ab

Wenn die Lebenspunkte eines Spielers auf null sinken, endet das Spiel und der Sieger bekommt einen Sieg eingetragen. Im Falle eines unentschieden wird beiden Spielern ein Sieg eingetragen.

Außerhalb des Spiels kann man sein Reittier wählen. Außerdem kann das Spiel leicht um die Möglichkeit erweitert werden, weitere Lanzen zur Auswahl zur Verfügung zu stellen.

4.1.2. User Interface

Das Spiel enthält während der Kernphase eine minimap, auf der die Position beider Spieler angezeigt wird. Außerdem gibt es zwei Lebenspunkteanzeigen, welche den aktuellen Zustand beider Spieler anzeigen.

Bei der Auswahl eines Reittieres wird eine Vorschau sowie die Eigenschaften des Tieres angezeigt.

4.1.3. Datenbankmodell

Da alle relevanten Daten in der Datenbank abgespeichert werden sollen, wurde ein ERD entworfen. Dabei wurde darauf geachtet, dass Lanzen und Reittiere leicht erweiterbar sind. In Diagramm 1 kann das Datenbankmodell eingesehen werden. Die Passwörter von Benutzern werden als Hash abgespeichert, um die Sicherheit der Passwörter zu gewährleisten.

4.1.4. Client-Server-Architektur

Für das Projekt ist serverseitige Berechnungen und clientseitige Darstellung des Spieles geplant. Deshalb wurde ein Modell für die Kommunikation zwischen Server und Client erstellt, welches einen möglichen, jedoch nicht endgültigen Prozessablauf diesbezüglich darstellt. Dieses kann in Diagramm 2 eingesehen werden. Dabei ist zu beachten, dass Inputs vom Client und Updates vom Server asynchron ablaufen, wobei sie bis zur Durchführung abgespeichert werden. Dies ermöglicht ein flüssiges Spielerlebnis ohne Pausen.

4.2. Implementierung

Das Projekt wird in Client, Server und Datenbank aufgeteilt. Dabei werden auch verschiedene Sprachen und Technologien verwendet, auf die nun eingegangen wird.

4.2.1. Datenbank

Auf Basis des in Kapitel 4.1.3 Datenbankmodell beschriebenen ERD wurde eine MySQL-Datenbank erstellt und die nötigen Tabellen mit dem Command Line Tool angelegt.

Erwähnenswert ist hierbei die Spalte „token“ in der Tabelle „users“. Dieser ist der Authentifizierungstoken des Nutzers, dessen Funktion in Kapitel 4.2.2 Serverseite erläutert wird.

Somit entspricht die Datenbank den in der Entwurfsphase definierten Anforderungen.

4.2.2. Serverseite

Der Server wurde in Visual Studio Code entwickelt. Er basiert auf einem Node.js Server, der mit Typescript implementiert wurde, welches zu JavaScript kompiliert wird. Zusätzlich wurde npm verwendet, um benötigte Pakete einzubinden. Diese können in Tabelle 4 eingesehen werden. Erwähnenswert sind dabei die Pakete „socket-io“ für die dauerhafte Kommunikation zwischen

Server und Clients, „mysql“ für die Verbindung zur Datenbank und „express“ für das Routing von Anfragen an den Webserver. Ein Admin-Dashboard wurde für einfache Analysezwecke angelegt.

Der Einstiegspunkt ist die Datei „app.ts“.

4.2.2.1. Nutzer-Management

Zum Anmelden und Registrieren von Benutzern werden die Routen „/login“ und „/register“ für POST-Requests mithilfe von „express“ erstellt. Dazu wird ein neuer Router („GameRouter“) erstellt und der Route „/“ zugewiesen. Diese Routen sprechen das DAO („DatabaseConnection“) an. Ein Beispiel dieses Prozesses kann in Snippet 1 eingesehen werden. Beim Anmelden wird ein neuer Authentifizierungstoken in Form eines SHA256 Hashes erstellt. Dieser kann zur Anmeldung am Socket benutzt werden.

Snippet 1

```
app.use("/", GameRouter.createGameRouter());

GameRouter.post("/register", async (req: Request, resp: Response) => {
        if (req.body.user && req.body.password) {

            try {
                const result = await
                            DatabaseConnection.getDatabaseConnection()
                            .registerUser(req.body.user, req.body.password);
                resp.send(result);
            } catch (error) {
                resp.status(500).send();
            }

        } else {
            resp.status(400).send();
        }
});
```

4.2.2.2. Datenbankverbindung

Die Verbindung zur Datenbank wurde über die statische Klasse „DatabaseConnection“ gelöst. Diese stellt mit Hilfe eines Connection Pools aus dem Paket „mysql“ für jede Anfrage eine eigene Verbindung her, die danach sofort wieder geschlossen wird. Dadurch wird die Last auf die Datenbank verringert, die Performance verbessert und sichergestellt, dass alle Verbindungen geschlossen werden. Die abgefragten Daten werden direkt als das Objekt instanziiert, das angefragt wurde. Alle Datenbankabfragen werden über diese Klasse getätigt und sind asynchron, um den Hauptablauf des Programmes nicht zu blockieren.

4.2.2.3. Socket-Server

Zur dauerhaften bidirektionalen Kommunikation mit Clients wird ein Socket („GameSocket“) des Pakets „socket-io“ verwendet. Dieser überprüft den Token des Nutzers und ermöglicht ihm die Kommunikation zum Server, wenn der Token gültig ist. Dafür definiert der Socket Events, welche vom Client aufgerufen werden können. Bei diesen erfolgt ein Datenaustausch über JSON-Strings. Ein Beispiel eines solchen Events kann in Snippet 2 eingesehen werden. Eine Auflistung der Events befindet sich in Tabelle 5.

Snippet 2

```
socket.on("start_singleplayer", this.startSinglePlayer.bind(this, socket));

private async startSinglePlayer(client: Socket.Socket)  {
        const player = User.getUserBySocketId(client.id);
        if (player === null)  {
            client.disconnect();
            return;
        }
        if (Game.isUserInGame(player)) {
            return;
        }
        try {
            await DatabaseConnection.getDatabaseConnection()
                  .createGame(player, null, "singleplayer");
        } catch (error) {
            client.disconnect();
            return;
        }
}
```

4.2.2.4. User

Session relevante Daten werden in Objekten „User“ Klasse abgespeichert. Diese wird außerdem genutzt, um alle aktiven Nutzer zu verwalten. Wenn ein Spiel startet, erstellt sie ein Objekt der Klasse „Player“.

4.2.2.5. Player

Die „Player“ Klasse enthält alle Daten eines Nutzers, die für ein Spiel relevant sind. Dazu gehören auch jeweils ein Objekt der Klassen „Lance“ und „Mount“, welche wiederum Ausrüstungsdaten enthalten. Sie hört das „game_input“-Event ab, wenn der Spieler eine Eingabe tätigt. Dadurch wird das Anheben der Lanze gesteuert. Die Klasse enthält eine Methode zur Aktualisierung ihrer Daten, die jeden Tick von der „Game“-Klasse aufgerufen wird. Außerdem sendet sie über das „game_update“-Event die Daten des neuen Spielstandes an den Client.

4.2.2.6. NPC

Die „NPC“-Klasse ist für die Steuerung des Computer Gegners im Singleplayer-Modus zuständig. Da sie von der „Player“-Klasse erbt kann sie einfach als Ersatz für einen zweiten Spieler dem Spiel übergeben werden. Dafür werden einige Methoden der „Player“-Klasse überschrieben oder überladen, damit dieser keine Events aussendet und seine Lanze zufällig bewegt.

4.2.2.7. Game

Wenn durch den Socket ein Spiel gestartet wird, wird ein neues Objekt der Klasse „Game“ erstellt und dieses in die Datenbank eingetragen. Sie enthält beide Spieler und eine zufällig gewählte Distanz, die diese voneinander entfernt sind. Sobald beide Spieler bereit sind startet ein Countdown, nach dem das Spiel startet. Im Game-Loop wird dann das Update der Spieler aufgerufen und ihnen anschließend der neue Spielstand gesendet. Statt einer sonst üblichen „while“-Schleife wurde hier die „setTimeout“-Funktion verwendet, um den Programmablauf nicht zu blockieren. Die Frequenz wurde dabei auf 30 Ticks pro Sekunde festgelegt. Dieser Ablauf wird unterbrochen sobald die Distanz zwischen den Spielern auf null sinkt. Dann werden mit Hilfe der Lanzenwinkel und Höhe der Reittiere die Trefferzonen berechnet und die Lebenspunkte der Spieler entsprechend aktualisiert. Anschließend wird geprüft ob die Lebenspunkte eines oder beider Spieler auf null gesunken sind. Ist das der Fall wird das Spiel beendet und werden Gewinner und Verlieren benachrichtigt. Kommt es zu einem Unentschieden gewinnen beide Spieler.

Um einen flüssigen Ablauf des Spiels zu gewährleisten, muss sichergestellt werden, dass Updates unabhängig von ihrer eigenen Prozessdauer durchgeführt werden, sodass, zum Beispiel im Falle einer Verzögerung dennoch die richtigen Werte errechnet werden. Dies wird durch den Code in Snippet 3 erreicht. Besonders zu beachten ist die Variable „timeDelta“, die die Zeit seit dem letzten Update berechnet, sodass dann die korrekten Berechnungen durchgeführt werden können.

Snippet 3

```
private updateGame() {
        const newTime = Date.now();
        const timeDelta = (newTime - this.timeOfLastUpdate) / 1000;
        this.timeOfLastUpdate = newTime;
        if (this.player1.getPosition() >= this.player2.getPosition()) {
            this.endRound();
            return;
        }
```

```
        this.player1.update(timeDelta);
        this.player2.update(timeDelta);

        this.player1.sendPartialGameUpdate(this.player2, this.gameWidth);
        this.player2.sendPartialGameUpdate(this.player1, this.gameWidth);

        setTimeout(() => {
            this.updateGame();
        }, Math.round(1000 / Constants.TICKRATE));
    }
```

4.2.3. Clientseite

Der Client ist dafür zuständig, die vom Server berechneten und geschickten Daten weiter zu verarbeiten und auf dem Display darzustellen. Für diese Aufgabe wurde Android Studio und der damit verbundene Java-Code genutzt. Hier wird die Gradle-Bibliothek: „io.socket:socket.io-client" der Version 1.0.0 verwendet. Dies ist nötig, um die bi-direktionale Kommunikation mit dem Server zu ermöglichen. Der Einstiegspunkt der Applikation ist die Klasse „LoginActivity".

4.2.3.1. Anmelden und Registrieren

In der „LoginAktivity" (Bild 1) kann der Nutzer sich mit seinen Benutzerdaten anmelden, oder den Registriervorgang starten. Beim Anmelden werden die Daten an ein Objekt der Klasse „ServerCommunicator" gesendet, welcher diese prüft. Sind sie korrekt wird der Authentifizierungstoken zurückgegeben und eine Socket-Verbindung zum Server aufgebaut. Andernfalls wird eine Fehlermeldung ausgegeben. Der Token wird außerdem im App-Internen Speicher gelagert, bis sich der Spieler aktiv ausloggt. Dadurch kann beim erneuten Start der Anwendung eine automatische Anmeldung erfolgen, sofern der Token noch gültig ist.

Beim Registrieren wird der Nutzer auf die „RegisterActivity" (Bild 2) weitergeleitet. Beim Registrieren muss darauf geachtet werden, einen unbenutzten Namen und ein Passwort mit einer Länge von 4 oder mehr Zeichen zu verwenden. Das Passwort muss zweimal identisch eingegeben werden. Bei erfolgreicher Registrierung wird der Nutzer über den Erfolg benachrichtigt und zurück auf die „LoginActivity" geleitet. Andernfalls wird eine Fehlermeldung ausgegeben.

4.2.3.2. Die Kommunikation mit dem Server

Der „ServerCommunicator" ist eine Hilfsklasse, über die dem Server POST-Requests gesendet werden können. Dabei wandelt er die Daten in das vom Server benötigte JSON-Format um. Bekommt er eine Antwort, wird diese an ein Objekt einer Klasse, die das Interface „ICommunicationResult" implementiert zurückgegeben, um jegliche Antwort annehmen zu können.

Die Klasse „ConnectionSocket“ kann mit einem gültigen Authentifizierungstoken initialisiert werden und damit dann eine dauerhafte Verbindung zum Server aufbauen. Anschließend können mit diesem Objekt alle nötigen Anfragen an den Server getätigt werden und auf Events von diesem reagiert werden. Auch beim Socket erfolgt die Kommunikation mit dem Server im JSON-Format.

4.2.3.3. Das Hauptmenü

Nach erfolgreichem Anmelden gelangt man in die „MenuActivity“ (Bild 3). Von dort aus kann entschieden werden, die Ausrüstung zu wechseln, ein Singleplayer- oder Multiplayer-Spiel zu starten, den Ton an oder aus zu schalten oder sich abzumelden. Außerdem wird dort die Anzahl der gewonnenen Spiele angezeigt.

4.2.3.4. Der SoundManager

Die statische „SoundManager“ Klasse ist für das Abspielen und Loopen der Musik und Effekte zuständig. Sie wurde als statische Klasse implementiert, da der Ton über alle Abschnitte der Applikation hinweg durchgehend laufen muss. So ist außerdem sichergestellt, dass der gesamte Ton über eine Stelle ab- oder angeschaltet werden kann.

4.2.3.5. Das Ausrüstungsmenü

In der „EquipmentActivity“ (Bild 4) kann der Spieler sein Reittier ändern. Dabei wird eine Liste aller Reittiere inklusive ihrer Eigenschaften vom Server geladen. Für das ausgewählte Reittier wird eine Vorschau angezeigt. Dafür werden aus der „MountStats“ Klasse, welche davor befüllt wurde, die Eigenschaften sowie das Bild des Reittiers geladen, welches man anschließend in der Vorschau darstellen kann. Wählt der Spieler eines aus und bestätigt die Auswahl, wird diese Information an den Server weitergeleitet, welcher entsprechende Datenbankeinträge vornimmt.

4.2.3.6. Ein Multiplayer-Spiel suchen

Beim Starten eines Multiplayer-Spiels beim Server ein Event aufgerufen, welches eine Suche nach einem Gegner startet. Wenn noch kein anderer Spieler auf ein Spiel wartet, wird der „SearchHandler“ gestartet. Dieser gibt dem Spieler die Information, wie lange er bereits auf ein Spiel wartet in Form eines Zählers. Außerdem wird ihm die Möglichkeit geboten, die Suche abzubrechen. Gibt es bereits einen suchenden Spieler, so werden die betroffenen Clients benachrichtigt und ein Spiel gestartet.

4.2.3.7. Darstellung von Inhalten

Um die Applikation auf unterschiedlichen Bildschirmgrößen korrekt darzustellen, wurde die „PixelConverter“-Klasse erstellt. Diese stellt statische Methoden zur Verfügung, welche Pixelgrößen, die übergeben werden, an die Bildschirmgröße anpassen. Dabei müssen die übergebenen Parameter in einem Verhältnis von 1920x1080 Pixel angegeben werden. Außerdem

stellt die Klasse eine Methode zur Umwandlung von Y-Werten zur Verfügung, welche das Darstellen von Inhalten sehr erleichtert. Dies wird durch das Umkehren der Y-Achse erreicht, so dass ihr null-Wert am unteren Bildschirmrand liegt, und nicht wie standardmäßig am oberen.

4.2.3.8. Das Spiel

Das Singleplayer- sowie das Multiplayer-Spiel sind für den Client identisch und findet in der „GameActivity“ (Bild 5) statt. Diese startet die GV und lädt die Reittiere und Lanzen aus einer XML-Datei. Dabei werden deren Bilder bereits in den Arbeitsspeicher geladen, um die Performance im Spiel selbst zu verbessern. Im Falle eines Singleplayer-Spiels übernimmt der Server lediglich die Kontrolle des Gegners. Die Informationen über das Spiel werden mit dem Server dauerhaft über den Socket ausgetauscht. Zum Zeichnen der Inhalte wird die übliche Methode von Update- und Draw-Loops auf einem SV genutzt.

Für das Laden der Bilder wurde die Klasse BM erstellt, welche sicherstellt, dass jedes Bild nur einmal in den Speicher geladen wird, somit die Performance verbessert und das Nutzen der vielen Bilder im Spiel erleichtert. Hierbei ist der Code-Abschnitt aus Snippet 4 besonders erwähnenswert. Dort werden zunächst nur die Informationen des Bildes geladen und die benötigte Skalierung aus diesen berechnet, um das Bild direkt in der benötigten Größe zu laden. Dies verhindert unnötiges Laden zu großer Bilder und verbessert damit die Performance erheblich. Die Funktion *„calculateInSampleSize”* ist dabei der Android Dokumentation[1] entnommen. Des Weiteren fügt der BM die Bitmaps des Spielers und seines Reittieres zusammen, um die Anzahl der Zeichenvorgänge während des Spieles zu reduzieren.

Snippet 4

```
BitmapFactory.Options bitmapOptions = new BitmapFactory.Options();
bitmapOptions.inJustDecodeBounds = true;

BitmapFactory.decodeResource(context.getResources(), resource,
bitmapOptions);
bitmapOptions.inSampleSize =
GameView.calculateInSampleSize(bitmapOptions,
PixelConverter.convertWidth(width,context),
PixelConverter.convertHeight(height,context));
bitmapOptions.inJustDecodeBounds = false;
Bitmap prepareBitmapHolder =
BitmapFactory.decodeResource(context.getResources(), resource,
bitmapOptions);
```

[1] https://developer.android.com/topic/performance/graphics/load-bitmap#java

Während der durchgängigen Updates durch den Server werden die einzelnen Objekte der Spieler aktualisiert. Diese werden bei jedem Update der GV geladen und auf die SV gezeichnet. Eine Besonderheit ist, dass das Hintergrundbild von der Mitte ab Identisch ist. Außerdem wird das Bild auf der Spielerseite zweimal nebeneinander gezeichnet, um es ohne „Sprünge“ nach „Links ziehen“ zu können. Dabei werden – sobald eines der Bilder den linken Bildschirmrand verlässt – beide wieder in die Startposition zurückgesetzt. Auf der Gegnerseite ist dies nicht möglich, da auf einer SV keine Z-Achse existiert und somit das neueste Bild immer den eigenen Hintergrund überschreibt. Deshalb musste der Ablauf so implementiert werden, dass die Spielerseite von der Gegnerseite mit einem Ausschnitt des Hintergrunds überschrieben wird. Dabei wird bei jedem Zeichenvorgang der richtige Ausschnitt berechnet, zugeschnitten und auf die Leinwand gezeichnet. Aus gleichen Gründen muss bei jedem Update die gesamte Leinwand neu gezeichnet werden. Die Hintergrundbewegung, sowie die Position der Spieler auf der „Minimap“ werden dabei aus der Geschwindigkeit und Position der Spieler berechnet.

Sobald die Clients die Information des Zusammenstoßes der Spieler erhalten, starten sie eine Animation, bei der jeweils der Gegner mit seinem Hintergrund aus dem rechten Bildschirmrand gezogen wird. Anschließend wird der Gegner mit seiner Geschwindigkeit auf den Spieler zubewegt und das vom Server berechnete Ergebnis dargestellt. Es wird ein Button eingeblendet, über den die Bereitschaft für die nächste Runde mitgeteilt werden kann. Ist das Spiel zu Ende, weil einer der Spieler keine Lebenspunkte mehr hat, führt der Knopf stattdessen zurück in das Hauptmenü (Bild 6).

Zum Zeichnen der einzelnen User-Interface Elemente wie den Lebensbalken oder der Texte wurde die Hilfsklasse „PaintManager“ erstellt, um das Einstellen von Zeicheneigenschaften zu erleichtern. Des Weiteren stellt sie sicher, dass diese Eigenschaften nur einmal erstellt werden müssen.

4.3. Qualitätssicherung

Um sicher zu stellen, dass das Projekt einen guten Verlauf nimmt, wurde entschieden, alle Dateien in GitHub zu sichern. Damit ist eine Versionierung garantiert und selbst in schlimmsten Fällen ein Schaden meist reversibel. Updates wurden immer mit einem Kommentar getätigt. Dadurch ist es besser nachvollziehbar, welche Änderung wann getätigt wurde und was sie bewirken sollte.

Außerdem wurde zum Testen des Servers und des User-Managements ein kleines Dashboard angelegt, über das der aktuelle Status des Servers beobachtet werden kann. Auf dem Server wurden außerdem aktuelle Code-Analysen durchgeführt, und Anmerkungen behoben. Auf Clientseite wurden ebenfalls Analysen durchgeführt, jedoch wurden nicht alle Anmerkungen behoben, da das Beheben dieser in vielen Fällen sinnfrei ist.

5. Projektabschluss

5.1. Soll-Ist-Vergleich

Für das Projekt sind Anforderungen gegeben, welche bereits in Kapitel 3.2 Beschreibung des Soll-Konzepts gegeben sind. Im Folgenden werden die erreichten Ziele gelistet und anschließend deren Umsetzung bewertet.

Beim starten des Spieles werden Nutzerdaten über eine Login-Ansicht abgefragt, wenn noch keine Login-Daten abgespeichert sind. Es wird außerdem ein Button angeboten, der zu einer Registrations-Ansicht führt. Dort kann dann ein neuer Benutzer angelegt werden. Dafür sind ein nicht benutzter Name und ein mindestens vierstelliges Passwort nötig. Beim Anmelden wird dem Nutzer ein eindeutiger Token zugewiesen, den dieser für die Authentifizierung beim Server benutzt. Außerdem wird dieser abgespeichert, sodass kein erneuter Login notwendig ist. Beim Ausloggen wird der Token gelöscht. Dann ist eine erneute Anmeldung erforderlich.

Über diesen Token wird außerdem sichergestellt, dass jeder Nutzer nur einmal angemeldet sein kann. Loggt sich ein anderes Gerät mit gleichen Daten ein, wird der Token des ersten Nutzers ungültig und der Nutzer damit abgemeldet.

Nach der Authentifizierung gelangt man in ein Hauptmenü, in dem der Nutzer ein Einzel- und Mehrspieler Spiel starten kann. Außerdem kann er dort seine Siege einsehen und sich ausloggen.

Für jede dieser Optionen erfolgt ein Datenaustausch mit einem Server. Dabei ist eine dauerhafte „Socket“-Verbindung für bidirektionale Kommunikation zuständig. Die Spieledaten werden auf dem Server verarbeitet, und auf dem Client dargestellt.

Für die Datenspeicherung wird eine MySQL-Datenbank genutzt, deren Steuerung über ein DAO auf dem Server erfolgt.

Somit sind alle an das Projekt gestellten Anforderungen erfüllt.

5.2. Fazit

Die Entwicklung des Webservers erfolgte auf die für Spiele übliche Weise und stellte höchstens kleinere Probleme dar. Diese konnten immer gelöst werden. Dadurch wurden viele wertvolle Erfahrungen in diesem Bereich gesammelt.

Die Darstellung der Spielinhalte im Zusammenhang mit der SV, an welche man durch Android Studio gebunden ist, stellte jedoch eine größere Herausforderung dar. Im Laufe des Projektes konnte immer wieder festgestellt werden, dass die gebotenen Möglichkeiten der SV den Anforderungen eines Spieles, welches mehrfach pro Sekunde aktualisiert werden muss nicht gerecht werden. Dies liegt daran, dass Android Studio den Fokus nicht auf Spiele-Applikationen setzt. Obwohl äußerst penibel Performance-Verbesserungen durchgeführt wurden, konnte zum Schluss nur ein zufrieden stellendes Ergebnis erreicht werden, kein perfektes.

Deshalb ist zu erwägen, die Darstellung einer Spiele-Applikation nicht über Android-Studio, sondern über ein anderes Framework mit einer angepassten Spiele-Engine zu implementieren. Im Rahmen dieses Projektes war es jedoch Aufgabe, Erfahrung diesbezüglich zu sammeln. Daher kann selbst diese Tatsache als voller Erfolg gewertet werden.

5.3. Ausblick

Da dieses Projekt als Basis für künftige Spiele genutzt werden soll und der Server eine generische Schnittstelle für Spiele zwischen Clients darstellt, kann er nahezu ohne Änderung für weitere Projekte genutzt werden. Die Darstellung auf der Clientseite sollte jedoch mit einem anderen Framework implementiert werden. An dieser Stelle werden daher Anpassungen notwendig sein, jedoch kann die allgemeine Architektur der Applikation, wie sie angefertigt wurde verwendet werden. Als nächstes Projekt wird dabei angestrebt, den Client in Unity[2] zu implementieren, da diese Engine speziell für Spiele entwickelt wurde und das einfache Erstellen einer mobilen App ermöglicht und alle erwähnten Probleme bezüglich der Darstellung von Bildern beseitigt.

[2] https://unity3d.com/

6. Anhang

Tabelle 1

Technologie	**Version**
Android Compiler (Client, IDE: Android Studio)	API 28: Android 9.0 (Pie)
Min. benötigte SDK-Version	API 25: Android 7.1.1 (Nougat)
Node.js (Server, IDE: Visual Studio Code)	9.4.0
Typescript (Server Programmiersprache)	3.2.1 (ES2018)
HAXM (Hardware-Beschleunigung)	7.3.2

Tabelle 2

FÄLLIGKEITSDATUM	WAS	FERTIG
14.09.2018	Anlegen des Projektes	Fertig
14.09.2018	Lauffähige App am Handy starten	Fertig
14.09.2018	Einbinden des Projektes in einem GitHub Branch	Fertig
Gängiger Prozess, etwa bis 07.12.2018	Starten des Entwicklungsprozesses, ausarbeiten des GDD	Fertig
05.10.2018	Darstellen von Elementen in der App	Fertig
05.10.2018	Server Aufsetzen, Branch dafür anlegen	Fertig
19.10.2018	User-Management über Server regeln	Fertig
26.10.2018	Umschreiben der Logik: Client -> Server	Fertig
02.11.2018	Gegnerlogik erstellen	Fertig
09.11.2018	Darstellung des Spiels am Klienten	Fertig
16.11.2018	Kollisionen	Fertig
23.11.2018	Kernelemente des Spiels ausarbeiten	Fertig
30.11.2018	Rahmen: Optionen, Equipment ausarbeiten	Fertig
07.12.2018	Refactoring des Codes	Fertig
20.12.2018	Dokumentation anlegen	

Tabelle 4

Paket	**Version**
source-map-support	0.5.9
tslint	5.11.0
socket.io	2.2.0
mysql	2.16.0
express	4.16.4
basic-auth	2.0.1
@types/mysql	2.15.5
@types/express	4.16.0
@types/node	10.12.10
@types/basic-auth	1.1.2
@types/node	10.12.10
@types/socket.io	1.4.39
@types/source-map-support	0.4.1

Tabelle 5

Event	**Beschreibung**
connected	Client verbunden (Standardevent)
disconnected	Verbindung getrennt (Standardevent)
start_singleplayer	Startet ein Singleplayer Spiel
start_multiplayer	Startet ein Multiplayer Spiel
get_equipment	Hole Ausrüstung aus Datenbank
leave_game	Spieler verlässt das Spiel
get_wins	Hole Anzahl der gewonnenen Spiele aus Datenbank
player_ready	Spieler ist bereit für nächste Runde
game_input	Spieler hat Aktion ausgeführt
found_game	Spiel gefunden
game_update	Nächster Spielstand wird gesendet

Diagramm 1

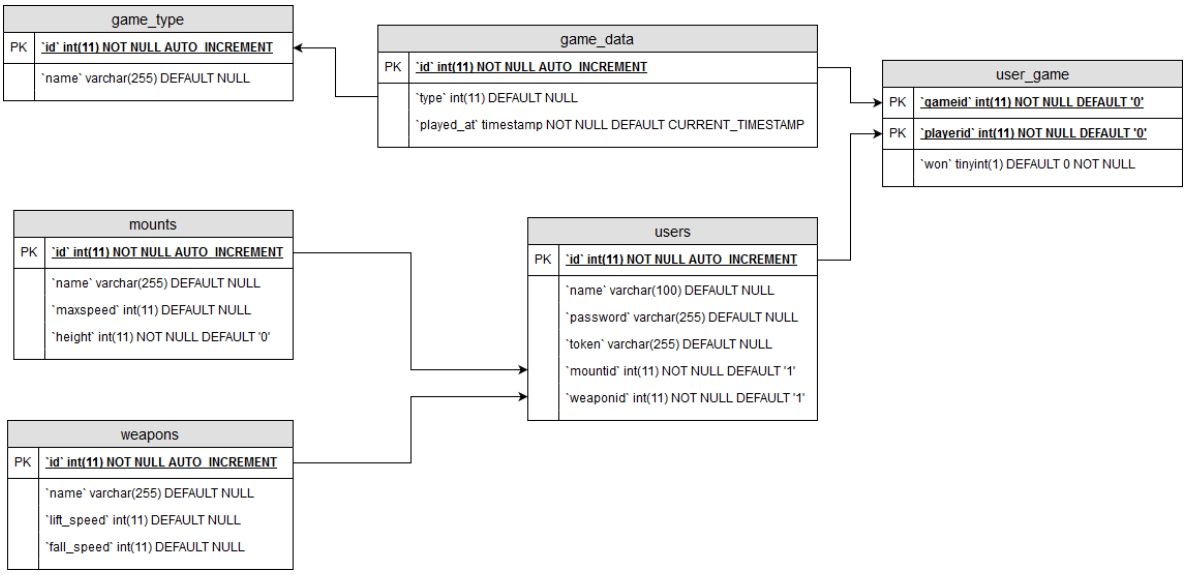

game_type
PK
`id` int(11) NOT NULL AUTO_INCREMENT
`name` varchar(255) DEFAULT NULL
game_data
PK
`id` int(11) NOT NULL AUTO_INCREMENT
`type` int(11) DEFAULT NULL
`played_at` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP
user_game
PK
`gameid` int(11) NOT NULL DEFAULT '0'
PK
`playerid` int(11) NOT NULL DEFAULT '0'
`won` tinyint(1) DEFAULT 0 NOT NULL
mounts
PK
`id` int(11) NOT NULL AUTO_INCREMENT
`name` varchar(255) DEFAULT NULL
`maxspeed` int(11) DEFAULT NULL
`height` int(11) NOT NULL DEFAULT '0'
users
PK
`id` int(11) NOT NULL AUTO_INCREMENT
`name` varchar(100) DEFAULT NULL
`password` varchar(255) DEFAULT NULL
`token` varchar(255) DEFAULT NULL
`mountid` int(11) NOT NULL DEFAULT '1'
`weaponid` int(11) NOT NULL DEFAULT '1'
weapons
PK
`id` int(11) NOT NULL AUTO_INCREMENT
`name` varchar(255) DEFAULT NULL
`lift_speed` int(11) DEFAULT NULL
`fall_speed` int(11) DEFAULT NULL

Diagramm 2

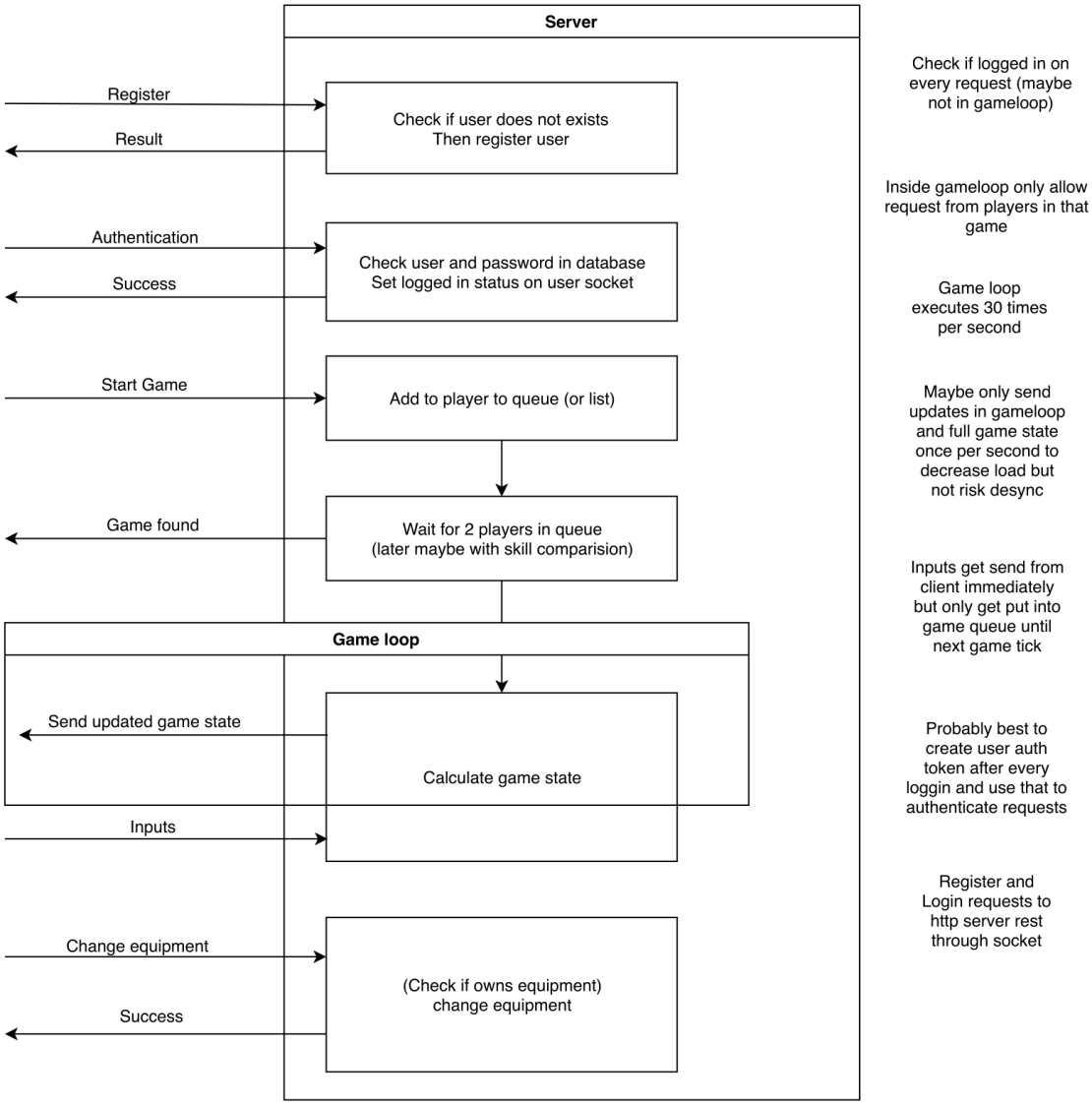

Server
Register
Result
Check if user does not exists
Then register user
Authentication
Success
Check user and password in database
Set logged in status on user socket
Start Game
Add to player to queue (or list)
Game found
Wait for 2 players in queue
(later maybe with skill comparision)
Game loop
Send updated game state
Calculate game state
Inputs
Change equipment
Success
(Check if owns equipment)
change equipment
Check if logged in on every request (maybe not in gameloop)
Inside gameloop only allow request from players in that game
Game loop executes 30 times per second
Maybe only send updates in gameloop and full game state once per second to decrease load but not risk desync
Inputs get send from client immediately but only get put into game queue until next game tick
Probably best to create user auth token after every loggin and use that to authenticate requests
Register and Login requests to http server rest through socket

Bild 1

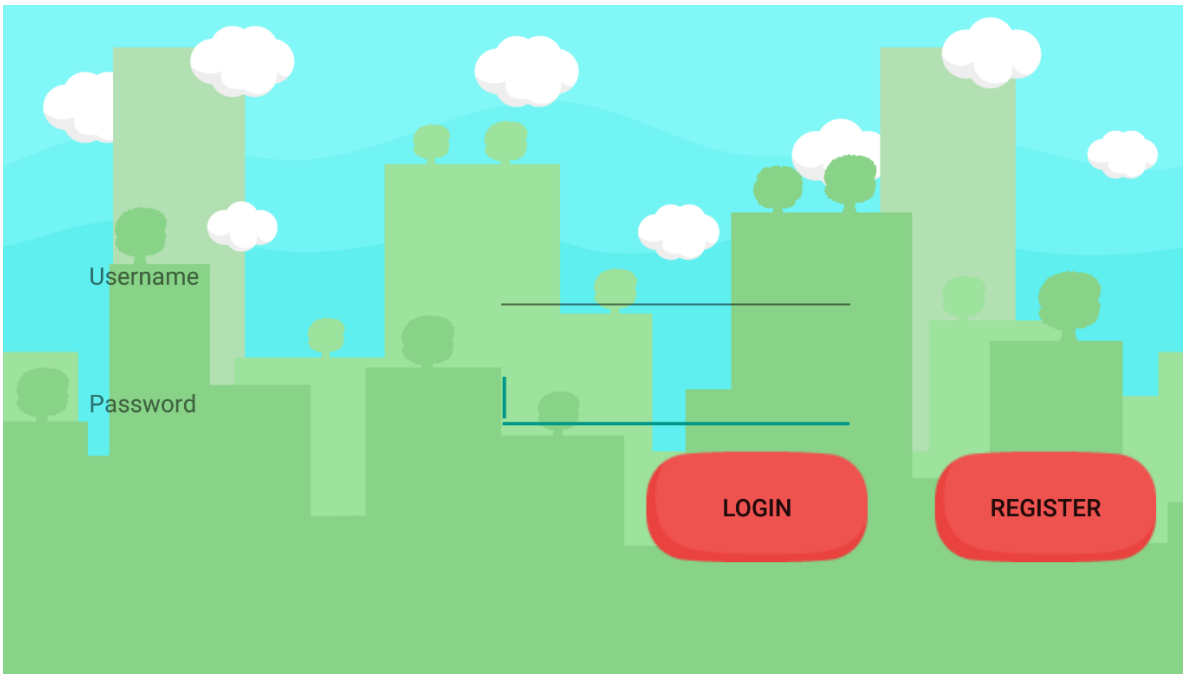

Username
Password
LOGIN
REGISTER

Bild 2

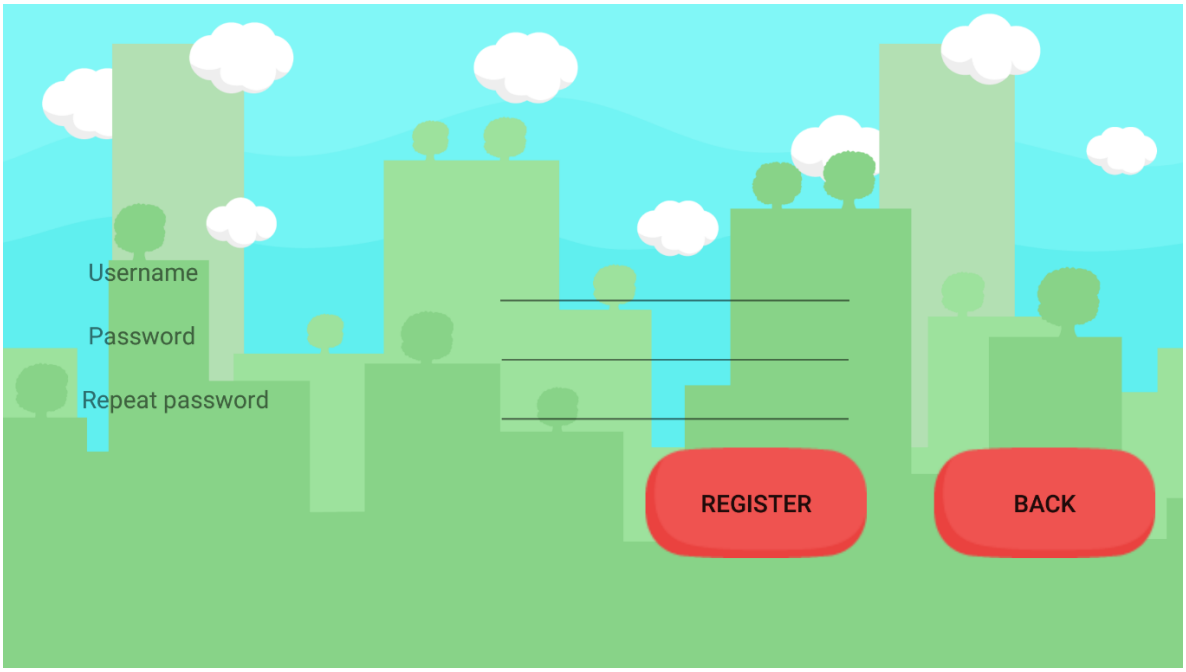

Username
Password
Repeat password
REGISTER
BACK

Bild 3

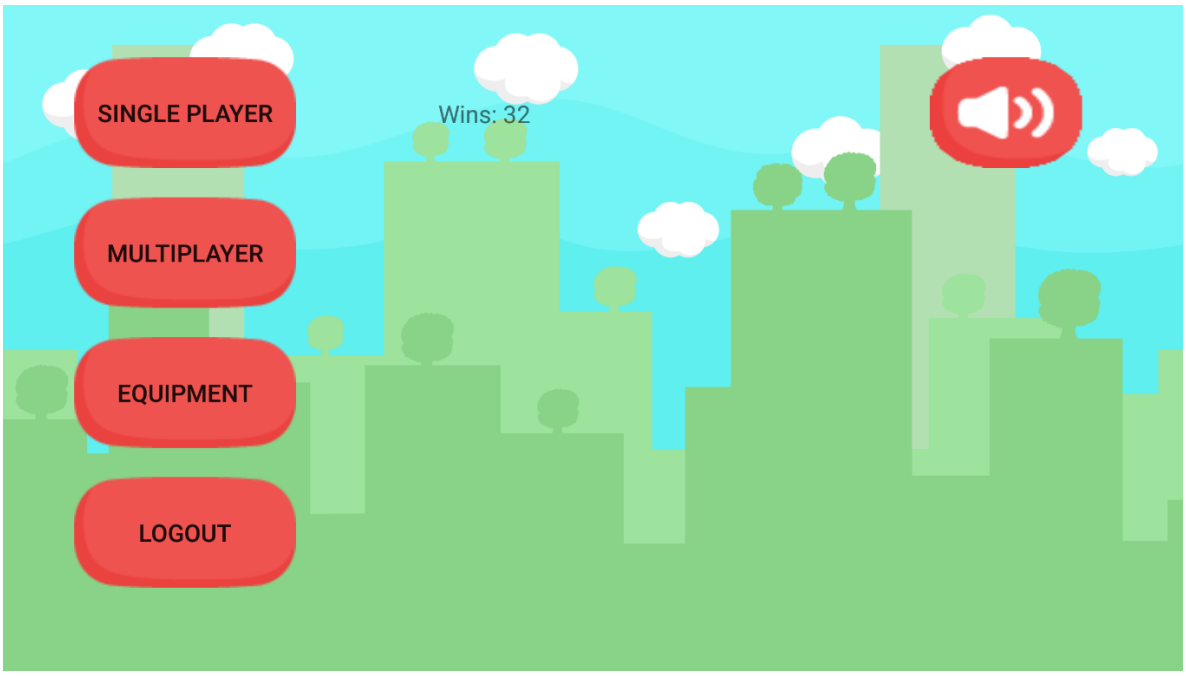

SINGLE PLAYER
Wins: 32
MULTIPLAYER
EQUIPMENT
LOGOUT

Bild 4

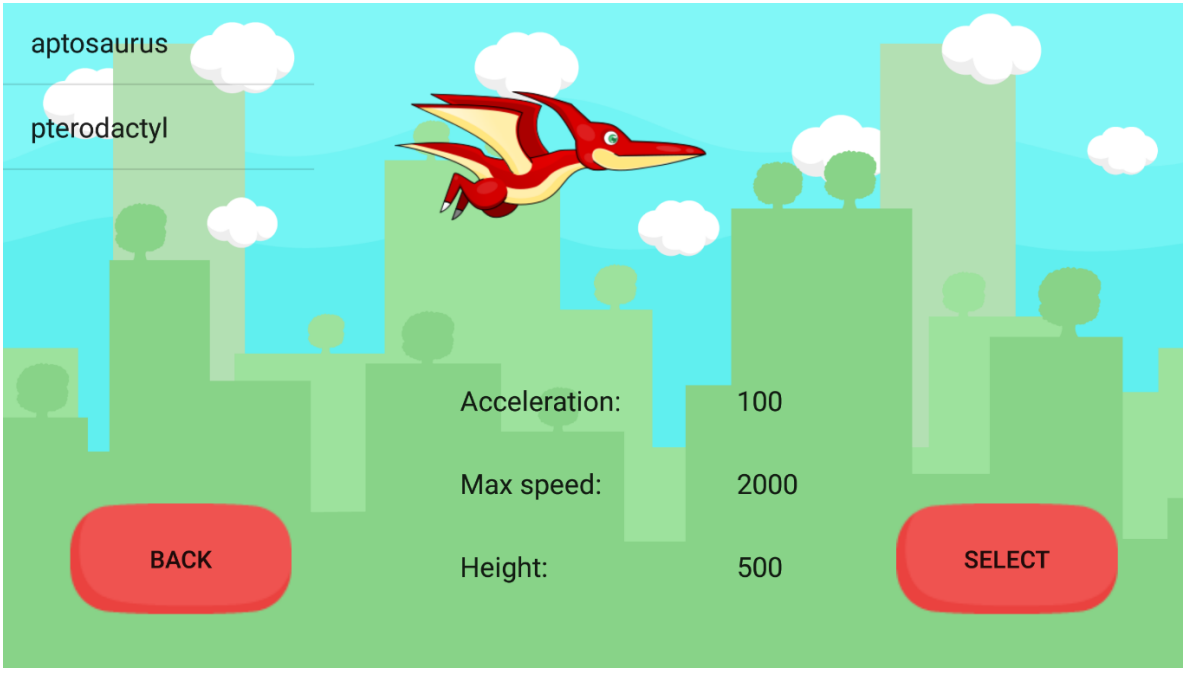

aptosaurus
pterodactyl
Acceleration: 100
Max speed: 2000
Height: 500
BACK
SELECT

Bild 5

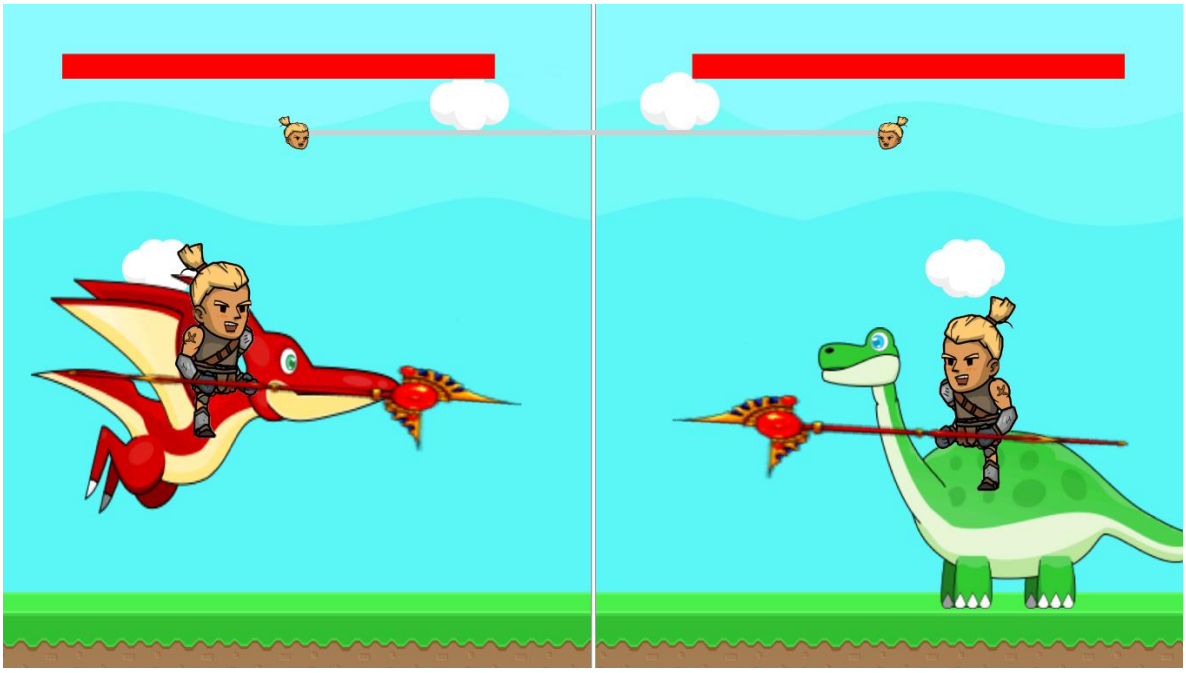
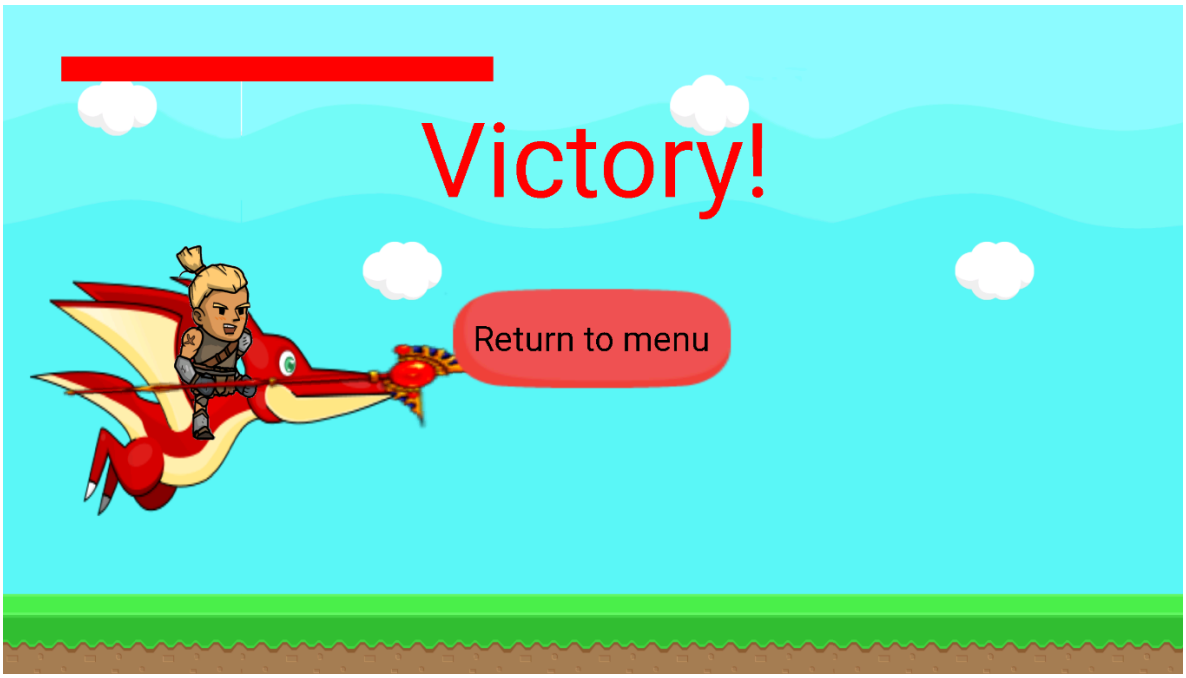


Bild 6



7. Glossar

Begriff	Bedeutung
DAO	Data-Access-Object
minimap	Kleine Karte des Spielfeldes
ERD	Entity-Relationship-Diagramm
SHA-256	Secure Hash Algorithm 256
npm	Node-Package-Manager
Tick	Ein Update des Servers
BM	BitmapManager
GV	GameView
SV	SurfaceView

8. Quellen

Game Assets:

- https://www.gamedevmarket.net/asset/platform-game-assets-7990/
- https://www.gamedevmarket.net/asset/fantasy-2d-character-free/
- https://www.gamedevmarket.net/asset/fantasy-equipment-pack-196/
- https://opengameart.org/content/2d-dinosaur-set

Flow Chart Maker:

- https://www.draw.io/

Einstieg in Android Spiele:

- https://www.simplifiedcoding.net/android-game-development-tutorial-1/

Dokumentationen:

- https://socket.io/docs/
- https://developer.android.com/docs/
- https://nodejs.org/api/