

# Projekt Dokumentation

6. Mai 2023

# Inhaltsverzeichnis

<b>1 Kurzdarstellung des Projektes</b>	<b>5</b>
1.1 Einleitung . . . . .	5
1.2 Ausgangsszenario . . . . .	5
1.3 Zielsetzung . . . . .	5
<b>2 Datenbank</b>	<b>6</b>
2.1 Was wird genutztzt? . . . . .	6
2.2 User bezogene Tabellen . . . . .	6
2.3 Die Tabelle User . . . . .	7
2.3.1 Rights und Superrights . . . . .	7
2.4 Die Tabelle Rounds . . . . .	7
2.5 Die Tabelle Scores . . . . .	7
2.6 Release bezogene Tabellen . . . . .	7
2.6.1 Die Tabelle Game . . . . .	7
<b>3 API</b>	<b>9</b>
3.1 Was ist eine API? . . . . .	9
3.2 Was ist REST? . . . . .	9
3.2.1 Warum überhaupt eine REST-API verwenden? . . . . .	9
3.3 Grundfunktionen . . . . .	9
3.4 Klasse getRequest . . . . .	10
3.4.1 Methode: getUser . . . . .	10
3.4.2 Methode: getRounds . . . . .	11
3.4.3 Methode: getRank . . . . .	11
3.5 Klasse postRequest . . . . .	11
3.5.1 Methode: postScore . . . . .	11
3.5.2 Methode: postFullScore . . . . .	12
<b>4 Website</b>	<b>13</b>
4.1 Webdesign . . . . .	13
4.2 Planung . . . . .	13
4.3 Umsetzung-Design . . . . .	13
4.4 Umsetzung-Frontend . . . . .	15
4.4.1 Navigationsleiste und Footer . . . . .	15
4.4.2 Startseite . . . . .	15
4.4.3 Anmelde- und Registrierbildschirm . . . . .	15
4.4.4 Profilverwaltung . . . . .	15
4.4.5 Rangliste . . . . .	16
4.4.6 Chnagelog . . . . .	17
4.5 Umsetzung-Backend . . . . .	17
4.5.1 Klassen . . . . .	18
4.5.1.1 Allgemeines zu den Konstruktoren . . . . .	19
4.5.1.2 Klasse DBconnection . . . . .	19
4.5.1.3 Klasse: login . . . . .	19
4.5.1.4 Klasse: registration . . . . .	20
4.5.1.5 Klasse: ranking . . . . .	20
4.5.1.6 Klasse: profile . . . . .	21
4.5.1.7 Klasse: changeData . . . . .	23
<b>5 Planung Website</b>	<b>25</b>
5.1 Planänderungen des Projektes . . . . .	25

<b>6 Organisation Spiel-Entwicklung</b>	<b>26</b>
6.1 Game-Engine . . . . .	26
6.1.1 Unreal Engine . . . . .	26
6.1.2 Unity . . . . .	26
6.1.3 Godot . . . . .	26
6.1.4 Entscheidung Game-Engine . . . . .	26
6.2 Zielsystem . . . . .	26
<b>7 Client-Kommunikation</b>	<b>27</b>
7.1 Kommunikations-Software . . . . .	27
7.1.1 Riptide . . . . .	27
7.1.2 Netcode for GameObjects (NGO) . . . . .	27
7.1.3 Entscheidung Kommunikations-Software . . . . .	27
<b>8 Kommunikations-Typ</b>	<b>28</b>
8.1 Dedicated Game Server (DGS) . . . . .	28
8.2 Client Hosted (Listen Server) . . . . .	28
8.3 Relay Service . . . . .	28
8.4 Entscheidung Kommunikations-Typ . . . . .	29
<b>9 Implementation Client-Kommunikation</b>	<b>30</b>
9.1 Implementation Netcode for GameObject (NGO) . . . . .	30
9.2 Implementation Relay Service . . . . .	30
9.2.1 Klasse: RealyManager . . . . .	30
9.2.1.1 Funktion: setupRelay() . . . . .	30
9.2.1.2 Funktion: joinRelay() . . . . .	31
9.2.1.3 Programm Ablauf setupRelay() . . . . .	31
9.2.1.4 Programm Ablauf joinRelay() . . . . .	31
9.3 Implementation Listen Server . . . . .	32
9.3.1 Verbindungs-Fehler . . . . .	32
9.3.2 RPC-Fehler . . . . .	32
9.4 Implementation Remote Procedure Calls (RPC's) . . . . .	32
9.4.1 Server-RPC . . . . .	32
9.4.2 Client-RPC . . . . .	33
<b>10 Interne Kommunikation</b>	<b>36</b>
10.1 Implementation API . . . . .	36
10.1.1 Klasse apiManager . . . . .	36
10.1.1.1 Funktion: GetUserFromId() . . . . .	36
10.1.2 Struct User . . . . .	36
<b>11 Nutzeroberfläche</b>	<b>38</b>
11.1 Inventar UI . . . . .	38
11.2 Developer Tool . . . . .	38
11.2.1 LogDisplay . . . . .	38
<b>12 Gameplay</b>	<b>39</b>
12.1 Inventar System . . . . .	39
12.1.1 Umsetzung Inventar-System . . . . .	39
12.1.1.1 Klasse interactionManager . . . . .	39
12.2 Spielsteuerung . . . . .	40
12.2.1 Inventar Steuerung . . . . .	40
<b>13 KI</b>	<b>41</b>
13.1 Defienition KI . . . . .	41
13.2 Anforderungen . . . . .	41
13.3 Behaviours . . . . .	41
13.3.1 Defienition Behaviors . . . . .	41
13.3.2 Motion Behaviors . . . . .	41

13.3.3	Action Selection . . . . .	42
13.3.4	Steering . . . . .	42
13.3.5	Locomotion . . . . .	42
13.3.6	Vorteile . . . . .	42
13.3.7	Nachteile . . . . .	42
13.3.8	Lösung . . . . .	43
13.4	Context Steering . . . . .	43
13.5	Pfadfindung . . . . .	43
13.5.1	Anforderungen . . . . .	43
13.5.2	Alternatieven . . . . .	43
13.5.3	A* Algorithmuss . . . . .	43
13.5.3.1	Vorteile: . . . . .	44
13.5.3.2	Nachteile: . . . . .	44
13.5.3.3	Probleme Während der Entwicklung: . . . . .	44
13.5.4	Unity Navigation System . . . . .	44
13.5.4.1	Agent: . . . . .	44
13.5.4.2	Navmesh und Pfadfindung: . . . . .	44
13.5.4.3	Vorteile: . . . . .	44
13.5.4.4	Nachteile: . . . . .	45
<b>14</b>	<b>Mapdesign</b> . . . . .	<b>46</b>
14.1	Türen . . . . .	47
<b>15</b>	<b>Unity Assets</b> . . . . .	<b>49</b>

## 1 Kurzdarstellung des Projektes

### 1.1 Einleitung

Das Horst-Abschlussprojekt findet im Laufe der schulischen Ausbildung zum informationstechnischen Assistenten, an der BBS-ME, durchgeführt. Das Projekt beginnt am 07.01.2023 und endet am 26.03.2023. Das Projekt wird von Herr Kowalewski betreut. Für dieses Projekt sind 160 Schulstunden eingeplant.

### 1.2 Ausgangsszenario

Das Spiel ist an den Lehrfilm „Elektriker Horst“ angelehnt und bringt dem Spieler die elektrotechnischen Sicherheitsregeln näher. Dieses fungiert als Multiplayer-Spiel. In diesem gibt es den Manipulator „Grim“ und den Wächter „Horst“. Zu dem Projekt gehört auch eine zugehörige Website und eine Datenbank, damit Spiel und Website miteinander kommunizieren können. Damit das bewerkstelligt werden kann, wird eine API benötigt, die zwischen Datenbank & Spiel vermittelt. Das gesamte Team ist in zwei Gruppen aufgeteilt. Eine Gruppe besteht jeweils aus drei Personen. Die GameDev-Gruppe ist für die Spielentwicklung mittels Unity verantwortlich und die WebDev-Gruppe ist für die Datenbank, die API & die Website verantwortlich. Teamleiter sind bei der GameDev-Gruppe Roman Khundaze und bei der Web-Dev-Gruppe Luca Henschel. Das Team GameDev nutzt Scrum und das Team WebDev nutzt das Wasserfallprinzip als Projektmanagement-System.

### 1.3 Zielsetzung

Bestandteile des Produktes ist ein Spiel welches Spielweise- und Aussehens-Standards des GDD erfüllt, dazu gehören eine funktionale Spielsteuerung, KI (für NPC's) und ein Mehrspielermodus welcher den Spielern ermöglicht gegeneinander zu Spielen. Für den Mehrspielermodus wird ein Konto benötigt welches auf der Website erstellt wird. Die User-Information werden in die Datenbank übernommen und die API stellt diese für das Spiel bereit. Über die Website werden zudem Releases des Spiels durch einen Admin hochgeladen und auf dieser befindet sich ein User-Ranking, dass anhand der Win-Lose-Rate bestimmt wird.

## 2 Datenbank

### 2.1 Was wird genutzt?

Das Datenbankensystem das verwendet wird ist MariaDB und basiert auf SQL. MariaDB wird verwendet, da es weit verbreitet ist und öfter Updates erhält als MySQL. Es sind zwei ER-Diagramme für die User-Informationen und für das hochladen des Spiels und dessen Changelog vorhanden. Die umgesetzten ER-Diagramme befinden sich in zwei SQL-Dateien. Das erstellen der Tabellen innerhalb der Datenbank werden somit über die SQL-Dateien erleichtert.

### 2.2 User bezogene Tabellen

Diese Tabellen werden von der Website und dem Spiel, über die REST-API, angesprochen. So werden die User-Informationen im Spiel, sowie auf der Website dargestellt.

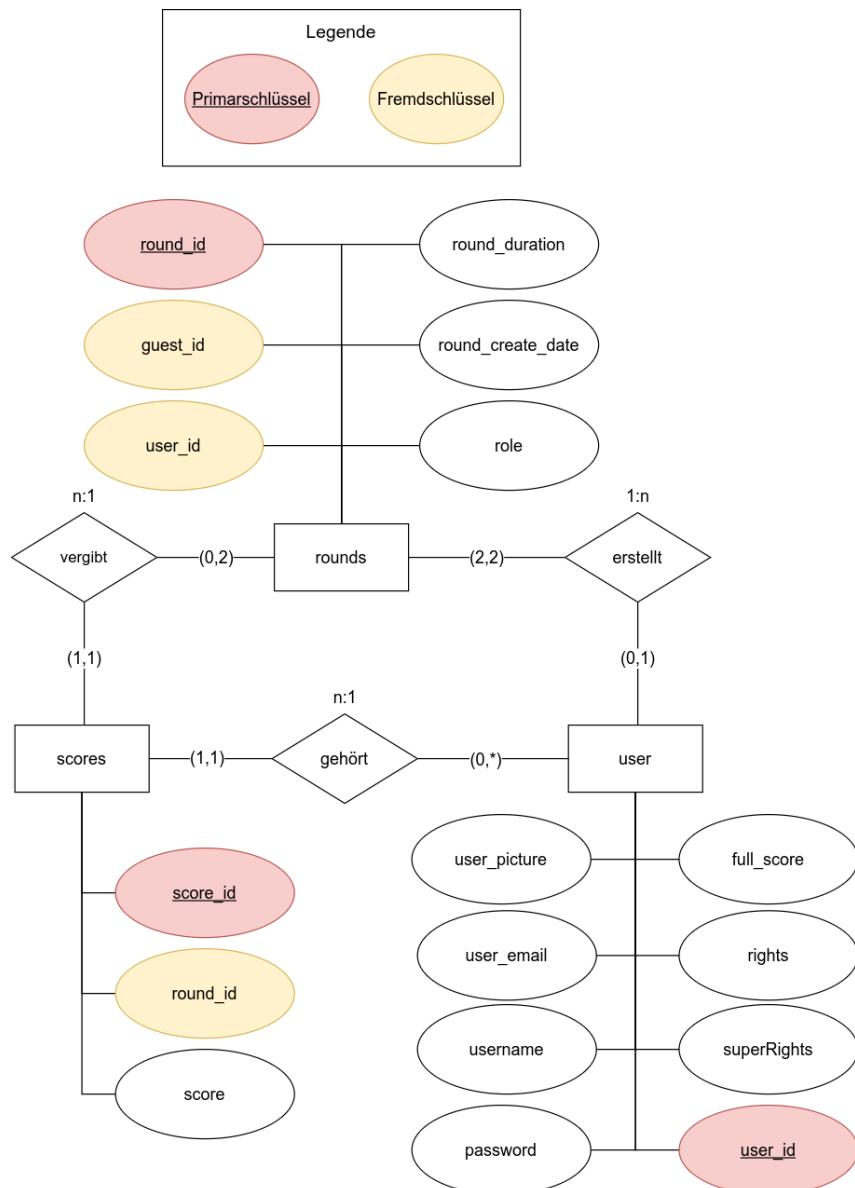


Abbildung 1: ER-Diagramm - User

## 2.3 Die Tabelle User

Der Attribut User-ID ist als Bigint deklariert und wird durch AI imkrementell mit jedem erstellen eines neuen User erhöht. Somit wird gewährleistet, dass jedem User eine einzigartige ID zugewiesen ist. Die Entscheidung ist auf Große Integer gefallen, damit es auch mit mehreren Benutzern keine Probleme mit der maximalen Spieleranzahl in der Datenbank gibt. Das Passwort für den Benutzer ist in einem Varchar deklariert und kann maximal 255 Zeichen enthalten. Somit kann jeder Benutzer seinen Account sichern. Die E-Mail Adresse wird ebenfalls in Varchar deklariert, kann aber nur 50 Zeichen lang sein. Das selbe gilt auch für den Username. Das Attribut full\_score, welcher die Win-/Lose Rate darstellt und in einem float gespeichert wird.

### 2.3.1 Rights und Superrights

Jeder User hat die Attribute rights und superrights, die jeweils mit einem boolean deklariert werden. Beide Attributbe sind standartmäßig auf false gesetzt. Rights steht hierbei für Admins. Diese können Spieler bei einem Regelverstoß banen. Superrights steht für die Owner, damit diese noch Kontrolle über die Admins behalten.

## 2.4 Die Tabelle Rounds

Der Primärschlüssel dieser Tabelle ist die Round\_id. Sie ist als Bigint deklariert und ist genau wie die User-ID mit AI versehen. Jede Runde hat somit eine einzigartige Round-ID. Es werden zwei User-IDs angegeben, welche aus der Tabelle User referenziert werden. Der eine User ist der Host, der andere ist der Guest, der dem gehosteten Spiel beitritt. Der Attribut role, welcher als Boolean deklariert wurde, zeigt an, ob der host den Horst, oder den Grim übernimmt. Der Guest bekommt somit automatisch die andere Rolle. Mit dem Attribut round\_create\_date wird das Spieldatum der Runde festgehalten. Round\_duration speichert die Laufzeit der Runde und ist als time Datentyp deklariert.

## 2.5 Die Tabelle Scores

Sobald eine Runde abgeschlossen wurde, werden den beiden Spielern neue Scores gesetzt. Genau wie Round-ID und User-ID ist auch der Attribut Score-ID ein Bigint, der sich auch immer automatisch auf den nächst größten Wert erhöht. Durch die Verknüpfung mit den Rounds ist die Round-ID mit Referenziert. Der Score setzt sich aus zwei Werten zusammen. Gewonnen oder Verloren. Somit ist dieser Attribut ein Boolean. Genau wie das Role Attribut referenziert sich dieser Boolean auf den Host. Der Guest bekommt somit seinen Score automatisch zugewiesen.

## 2.6 Release bezogene Tabellen

Diese Tabellen werden nur von der Website genutzt, um eine Spiel-Veröffentlichung, durch einen Admin, durchzuführen. Es wurde sich dazu entschieden die notwendigen Attribute innerhalb einer Tabelle anzulegen, da die Referenzierung und den Arbeitsaufwand im Backend, mittels PHP, zu viel Zeit in Anspruch nehmen würde.

### 2.6.1 Die Tabelle Game

Bei einem Release wird der Pfad zu den Dateien Bei einem Release wird der Pfad zu den Dateien innerhalb der gameFile und der markdownfile gespeichert. createDate speichert den aktuellen Zeitstempel des Datums.

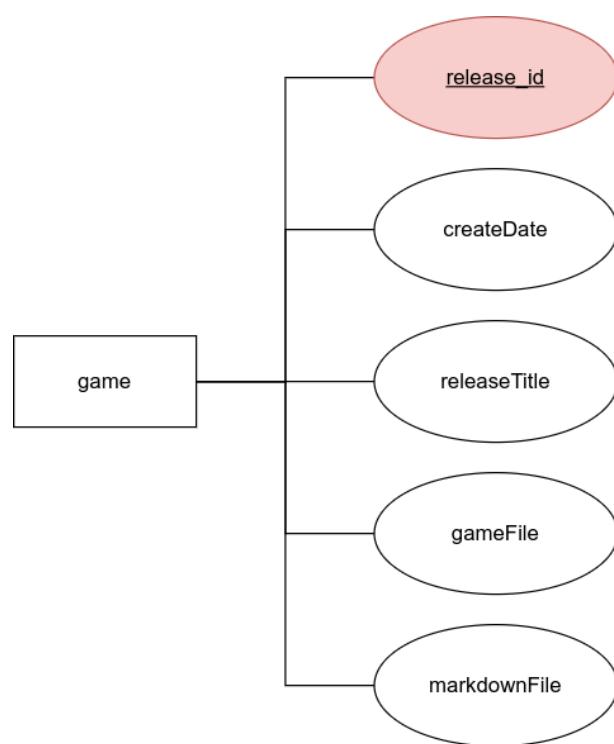


Abbildung 2: ER-Diagramm - Game

### 3 API

Die API ist nach dem REST-Standard entwickelt. Die Request-Methoden, die an die API gestellt werden, sind auf Wunsch des GameDev Teams auf POST und GET beschränkt.

#### 3.1 Was ist eine API?

API steht für „Application Programming Interface“. Eine API wird verwendet, um zwei oder mehr Applikationen miteinander zu verbinden. In diesem Fall Spiel und Datenbank. Sie wird als unabhängige Schnittstelle zwischen diesen verwendet.

#### 3.2 Was ist REST?

Der REST-Standard dient zu Standardisierung von APIs und dessen Struktur. Sie geben die Request-Methoden, wie POST, GET, UPDATE & DELETE, und das Rückgabeformat fest. Das Rückgabeformat ist dabei immer eine JSON. Zudem sind alle REST-APIs zustandslos.

##### 3.2.1 Warum überhaupt eine REST-API verwenden?

Durch den REST-Standard lässt sich die API leichter anpassen, da es einer festen Struktur folgt und ist somit leichter zu erweitern. Da das GameDev Team durch Bibliotheken leichter die zurückgegebene JSON verarbeiten können.

#### 3.3 Grundfunktionen

Die Nutzung der API ist nur von dem Spiel vorgesehen. Es werden über die Methoden der API Nutzer bezogene Daten angefragt und aktualisiert. Die Umsetzung erfolgt mit PHP, da es ein Kriterium einer REST-API ist, dass eine Request für sich alleine steht und nicht aufeinander aufbauen. PHP bringt diese Eigenschaft schon mit sich. Zudem ist die API Klassen basierend, da somit eine bessere Übersicht über die Struktur gewährleistet ist und die Erweiterung von Methoden kein Problem darstellt.

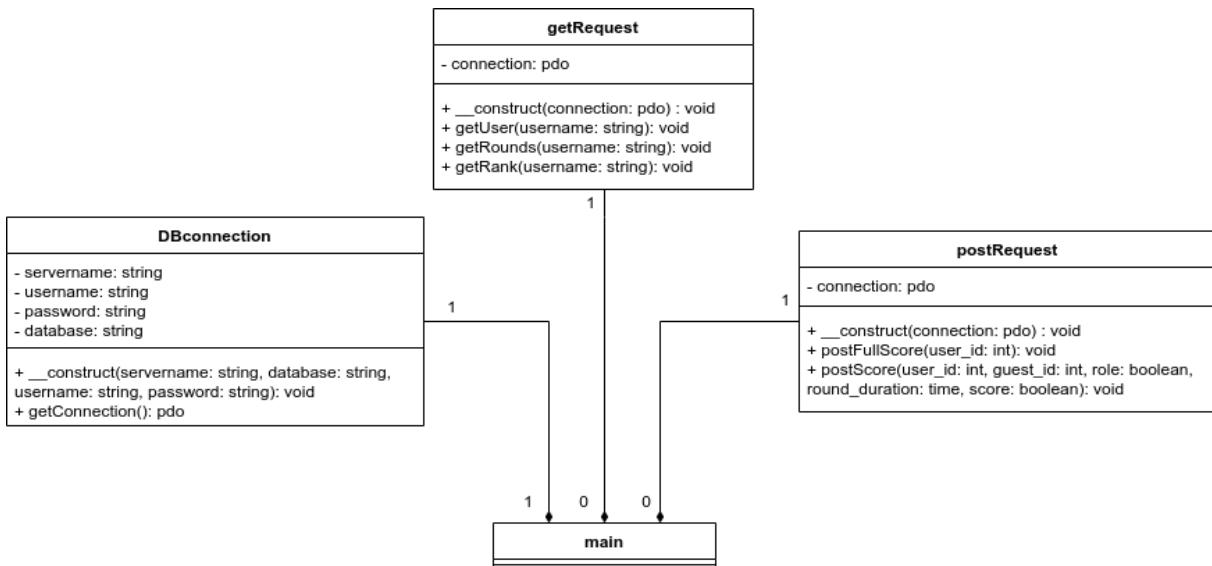


Abbildung 3: API Klassen-UML

Die Klasse „DBconnection“ dient zur Verbindung mit der Datenbank. Die „getRequest“ biete die Funktion User bezogene Daten abzufragen und mit der „postRequest“ werden die User bezogenen Runden hochgeladen.

```
1 header("Access-Control-Allow-Origin: *"); // verhindert mögliche Fehler durch den Browser
2 $method = $_SERVER["REQUEST_METHOD"]; // fragt die Request-Methode ab
3
4 if($method == "GET") {
    $getMethod = $_GET["method"];
5 }
```

```

6     if(!empty($getMethod)) {
7         header("Content-Type: application/json"); // schaltet auf die JSON-Ansicht
8         $getRequest = new getRequest($connection); // uebergibt PDO-SESSION an Klasse
9
10
11     $username = $_GET["username"];
12     if($getMethod == "getUser") {
13         $getRequest->getUser($username);
14     }
15     if($getMethod == "getRounds") {
16         $getRequest->getRounds($username);
17     }
18     if($getMethod == "getRank") {
19         $getRequest->getRank($username);
20     }
21 }
22 }
23
24 if($method == "POST") {
25     $postMethod = $_POST["method"];
26
27     if(!empty($postMethod)) {
28         $postRequest = new postRequest($connection); // uebergibt PDO-SESSION an Klasse
29
30         if($postMethod == "postScore") { // speichert die Runde und den zugehoerigen
31             Score
32             // ist inklusieve der Eintraege der Rounds
33             $user_id = $_POST["user_id"];
34             $guest_id = $_POST["guest_id"];
35             $role = $_POST["role"];
36             $round_duration = $_POST["round_duration"];
37             $score = $_POST["score"];
38
39             $postRequest->postScore($user_id, $guest_id, $role, $round_duration, $score);
40         }
41         if($postMethod == "postFullScore") { // fullScore bildet sich aus den letzten
42             Runden und zeigt den Durchschnitt an
43             $user_id = $_POST["user_id"];
44
45             $postRequest->postFullScore($user_id);
46     }
47 }
48 }
```

Bei einem Request-Versuch wird ermittelt welche Request-Methode verwendet wird, mithilfe von „\$\_SERVER[REQUEST\_METHOD]“. Anhand der Request-Methode sind andere Parameter zu übergeben. Jede Datenbank-Anfrage wird über eine Methode innerhalb der Klasse „getRequest“ oder „postRequest“ durchgeführt und jede dieser Requests werden innerhalb von der PDO-Funktion PREPARE gestellt, um vor einer SQL-Injektion zu schützen.

### 3.4 Klasse getRequest

Alle Methoden die sich innerhalb der Klasse „getRequest“ befinden, benötigen als Parameter den „username“ des Ziel-Users. Zudem ist jede Antwort an den Client, falls keine Fehler auftreten, eine JSON.

#### 3.4.1 Methode: getUser

```

1 public function getUser($username) { // gibt UserDaten wieder
2     try {
3         $statement = $this->connection->prepare("SELECT * FROM user WHERE username = :username");
4         $statement->execute(array("username" => $username));
5         $row = $statement->fetchAll(PDO::FETCH_ASSOC);
6
7         echo json_encode($row);
8     } catch (PDOException $e) {
9         echo "Fehler: " . $e->getMessage();
10    }
11 }
```

AnAnhand des übergebenden „username“ werden dem Client alle Attribute des Users übergeben.

### 3.4.2 Methode: getRounds

```
1 public function getRounds($username) { // uebergibt alle Informationen zu den Runden,
2     anhand des Username
3     try {
4         $statement = $this->connection->prepare("SELECT scores.score, user.username AS
5             owner, guest.username AS guest, rounds.role, rounds.round_create_date FROM rounds
6             INNER JOIN scores ON rounds.round_id = scores.round_id INNER JOIN user as user ON
7             rounds.user_id = user.user_id INNER JOIN user as guest ON rounds.guest_id = guest.
8             user_id WHERE user.username = :username ORDER BY rounds.round_create_date DESC LIMIT
9             20;");
10        $statement->execute(array("username" => $username));
11        $row = $statement->fetchAll(PDO::FETCH_ASSOC);
12
13        echo json_encode($row);
14    } catch (PDOException $e) {
15        echo "Fehler: " . $e->getMessage();
16    }
17}
```

Anhand des übergebenden „username“ werden dem Client die letzten 20 Runden zurückgegeben. Es werden innerhalb des SQL-Querys werden mittels „INNER JOIN“ der Ausgabe der Tabelle „Rounds“ der Username, innerhalb der Tabelle „user“, von der „guest\_id“ und von der „user\_id“ und der „score“, aus der Tabelle „Scores“, beigefügt. Es werden alle User einem Rang zugewiesen. Anhand des Username wird dann der Rang wiedergegeben.

### 3.4.3 Methode: getRank

```
1 public function getRank($username) { //gibt fuer den entsprechenden User den Rang wieder.
2     try {
3         $statement = $this->connection->prepare("SELECT position, user_id, username,
4             full_score FROM (SELECT user_id, username, full_score, @row_number:=@row_number+1 AS
5             position FROM user, (SELECT @row_number:=0) AS rn ORDER BY full_score DESC) AS
6             ranked_users WHERE username = :username"); // Ordnet dem User die Tabellen-Position
7             zu und selektiert dann nach Username
8         $statement->execute(array("username" => $username));
9         $row = $statement->fetchAll(PDO::FETCH_ASSOC);
10
11         echo json_encode($row);
12     } catch (PDOException $e) {
13         echo "Fehler: " . $e->getMessage();
14     }
15}
```

Der SQL-Query bewirkt, dass jedem User anhand des Attributes „full\_score“, innerhalb der Tabelle „User“, nach absteigender Reihenfolge ein Rang zugeordnet.

## 3.5 Klasse postRequest

Alle Methoden die sich innerhalb der Klasse „getRequest“ befinden, benötigen als Parameter den „username“ des Ziel-Users. Zudem ist jede Antwort an den Client, falls keine Fehler auftreten, eine JSON. Alle Methoden in dieser Klasse benötigen den Parameter „user\_id“, der über die „getUser“-Methode innerhalb der „getRequest“-Klasse ermittelt wird. Innerhalb der Klasse „postRequest“ ist es vorgesehen auf Spiel-Seite nur die Methode „postScore“ aufzurufen. Diese benötigt zusätzliche Parameter, wie „guest\_id“, „role“ und co.

### 3.5.1 Methode: postScore

```
1 public function postScore($user_id, $guest_id, $role, $round_duration, $score) { //speichert die Runde und den Zugehörigen Score für die Runde (der Score gilt für die "user_id")
2     try {
3         $statement = $this->connection->prepare("INSERT INTO rounds (user_id, guest_id,
4             role, round_create_date, round_duration) VALUES (:user_id, :guest_id, :role, NOW(), :
5             round_duration); INSERT INTO scores (round_id, score) VALUES (LAST_INSERT_ID(), :
6             score);");
7         // LAST_INSERT_ID() nimmt die letzte round_id, die in der jetzigen erstellten PDO
8         -SESSION erstellt wurde. Dies ist also nicht Verbindungs-Uebergreifend
```

```
5     $statement ->execute(array("user_id" => $user_id, "guest_id" => $guest_id, "role"
6         => $role, "round_duration" => $round_duration, "score" => $score));
7
8     $this ->postFullScore($user_id); // ruft die Funktion auf fuer den Score
9 } catch (PDOException $e) {
10     // Behandlung des Fehlers
11     echo "Fehler: " . $e->getMessage();
12 }
13 }
```

Bei der Methode „postScore“ wird User bezogen die Runde und dessen Score in die Datenbank eingetragen. Am Ende ruft diese Methode die Methode „postFullScore“ auf.

### 3.5.2 Methode: postFullScore

Vorgesehen ist, dass diese Methode durch „postScore“ nach jeder Runde aufgerufen wird. Somit ist nicht vorgesehen, dass direkt durch das Spiel diese Methode aufgerufen wird. Die Möglichkeit besteht jedoch. Die Methode „postFullScore“ wird das Attribut „fullScore“ User bezogen gesetzt, falls mindestens 15 Runden von diesem User gespielt wurden. Das Attribut setzt sich aus dem Durchschnitt der letzten 15 bis 20 Runden berechnet und ergibt eine Prozentzahl, ie die Win- Lose-Rate widerspiegelt.

## 4 Website

### 4.1 Webdesign

Vor dem Umsetzen ist ein Design geplant, damit in der Umsetzung keine Komplikationen auftreten. Somit ist von anfang an klar, wie das Design aussehen soll. Dieses Design ist in Stickpunkten notiert.

### 4.2 Planung

Für die Planung des Webfrontents, ist die Entscheidung auf das Online-Tool Figma gefallen. Es bietet das Designen von Webseiten und die Möglichkeit kollaboratives Arbeiten zwischen Mitarbeitern an. Figma ist Cloud-basierend und somit Betriebssystemunabhängig.

### 4.3 Umsetzung-Design

In der Gruppe Web-Dev ist ein Design ausgearbeitet. Auf fast jeder Seite ist eine Navigationsleiste und ein Footer eingebettet.

**Startseite:**

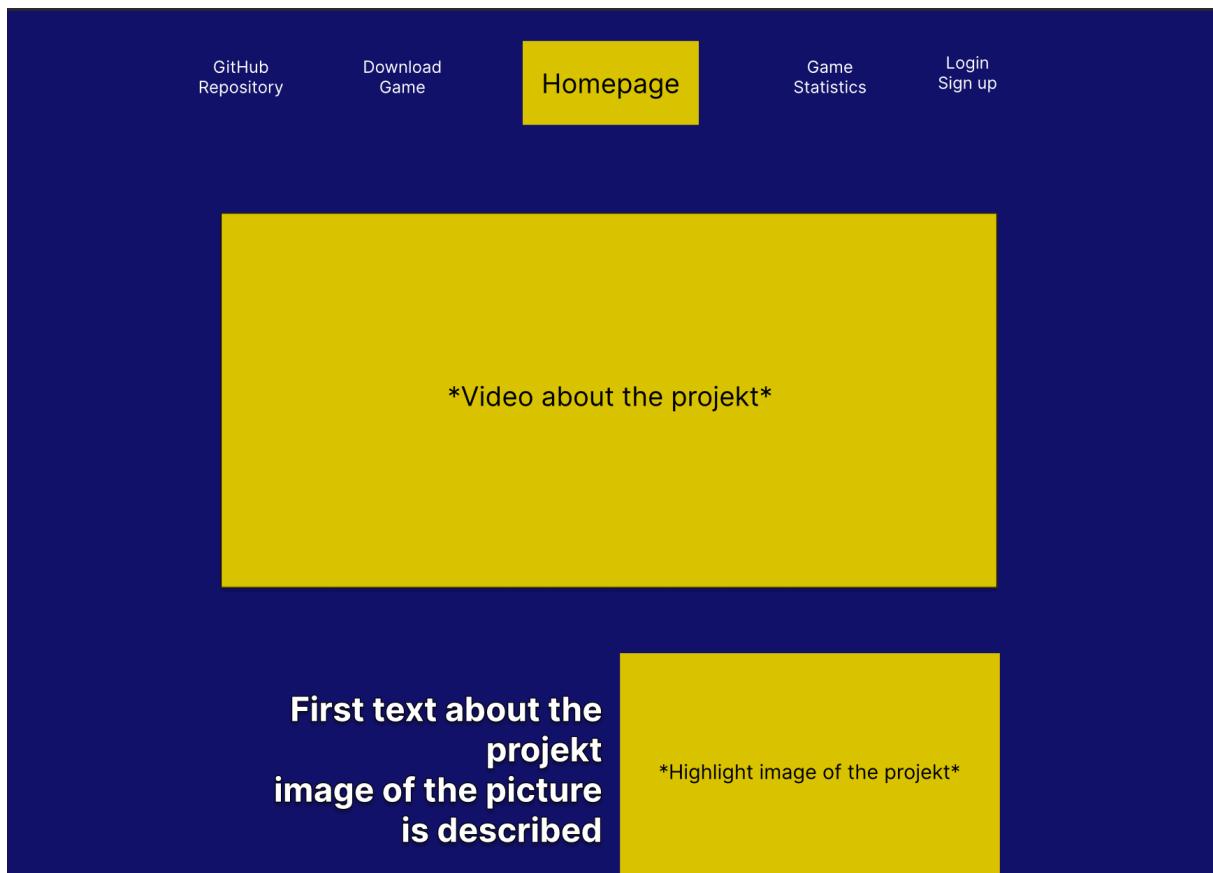


Abbildung 4: Figma Homepage

Die Startseite beinhaltet ein Video, über das Projekt. Darunter werden dann Bilder mit einem Text dazu angezeigt. Die Bilder beschreiben stellen einen estimmtten Bereich des Projektes dar. Das Bild wird dann vom Text dazu erklärt.

**Login:**

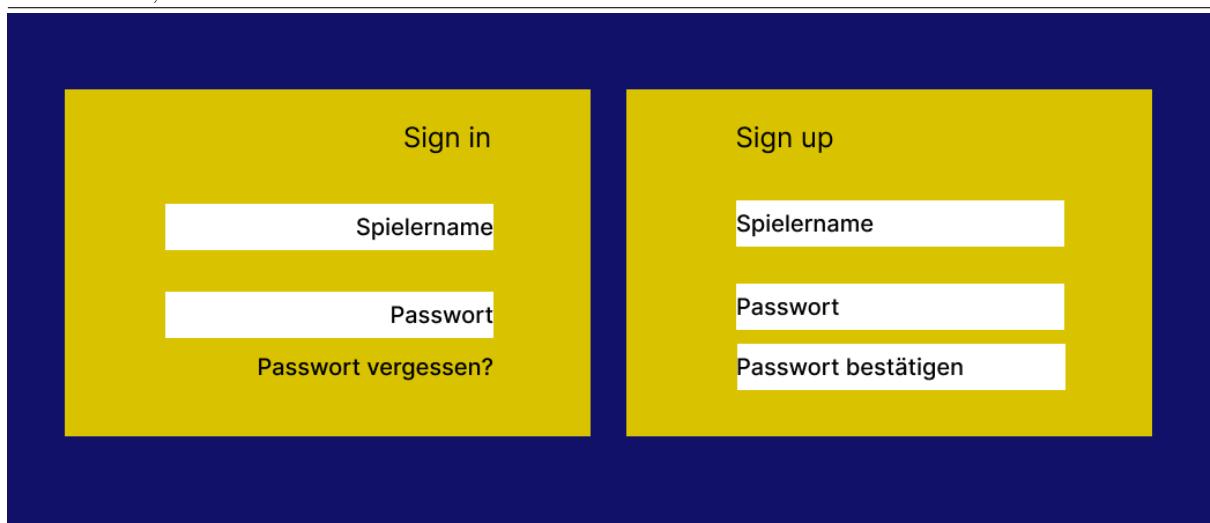


Abbildung 5: Figma Login

Auf dem Anmelde- und Registrierbildschirm ist jeweils ein umrahmtes Feld für das Anmelden und das Registrieren. Die Entscheidung ist auf eine einzige Seite gefallen, damit, damit die Webseite nicht leer erscheint. Somit wurde weiterer Arbeitsaufwand für die Frontend Umsetzung vorgebeugt.

**Profil:**

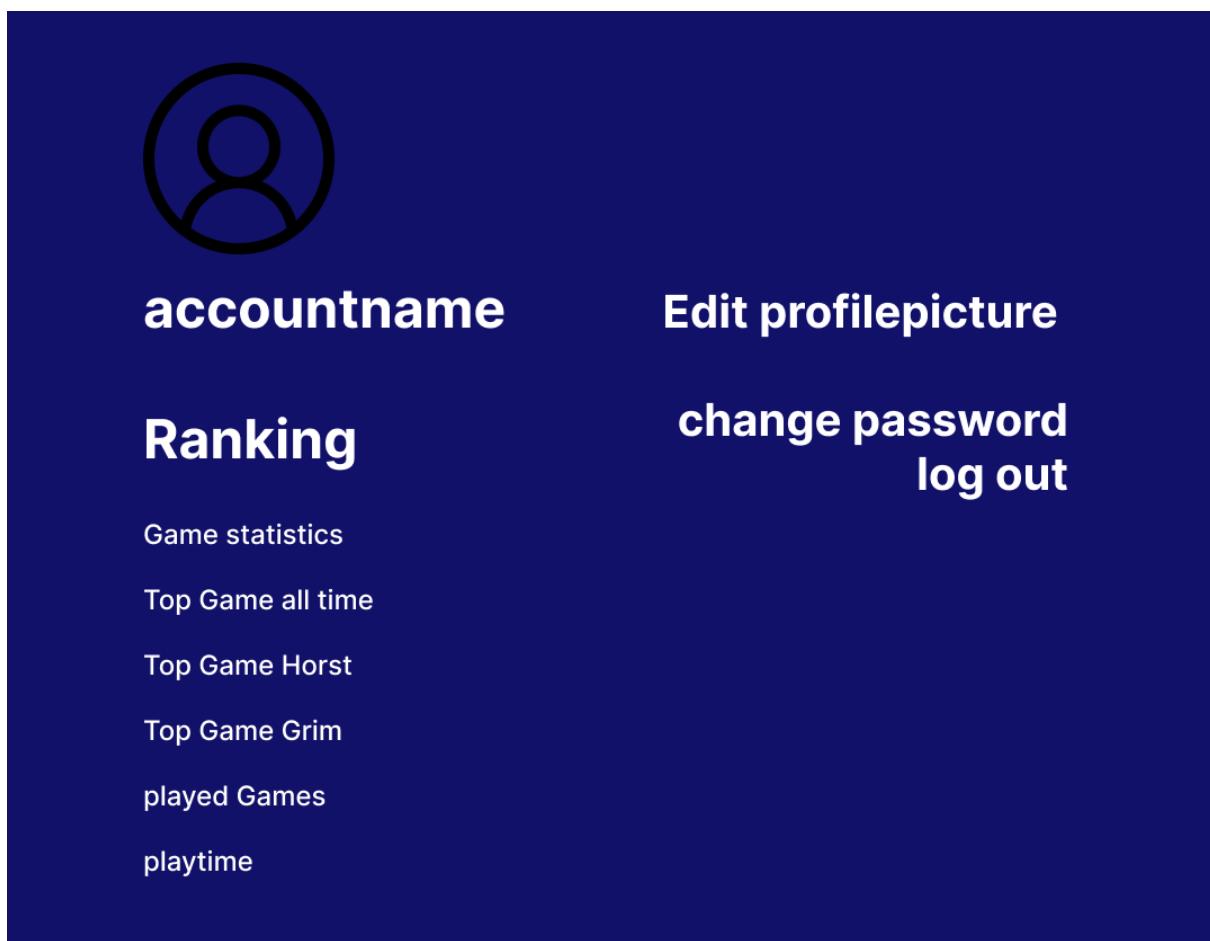


Abbildung 6: Figma Profil

Auf der linken Seite werden die Accountinformationen abgebildet. Wichtige Daten dazu werden in einer größeren Schriftgröße hervorgehoben und fett markiert. Auf der rechten Seite befinden sich alle Elemente zur Profilbearbeitung.

#### Scoreboard:

Rang	Username	win/lose difference
1.	Name 1	1,89
2.	Name 2	1,4
3.	Name 3	0,9

Abbildung 7: Figma Score

Im Scoreboard werden aktuell die besten Spieler der Reihenfolge nach dargestellt. Neben Dem Rang mit dem Username wird der Score angezeigt.

## 4.4 Umsetzung-Frontend

Mittels CSS sind die Seiten so skaliert, dass sie mit möglichst jedem Gerät ablesbar sind. Insgesamt wurden 3 Bildschirmgrößen als Vorlage genutzt. Desktop-PC und Laptop, Tablets und Smartphones.

### 4.4.1 Navigationsleiste und Footer

Jede der Webseiten enthält eine Navigationsleiste und einen Footer, die immer angezeigt werden. Im Footer stehen die Entwickler, das Impressum und die Datenschutzerklärung. In der Navigationsleiste sind Verlinkungen zu anderen Seiten der Webseite. Nach der Anmeldung des Users wird die Navbar um den Eintrag Profil ergänzt.

### 4.4.2 Startseite

Die Startseite setzt sich aus einem Video, zwei Bildern und zwei Textabschnitten zusammen. Die Bilder mit Unterbilder mit den Texten werden für gute Lesbarkeit je nach Seitenbreite angepasst.

### 4.4.3 Anmelde- und Registrierbildschirm

Die Anmeldeformulare und Registrierungsformulare sind auf einer Seite abgebildet. Links ist ein ausgefüllter Block für das Anmelden, rechts ist einer zum Registrieren. Beide Ausfüllformulare sind optisch voneinander getrennt. Bei kleiner Bildschirmskallierung werden die Module untereinander dargestellt.

### 4.4.4 Profilverwaltung

Oben Links ist eine Tabelle, in der das Profilbild des Users dargestellt wird. Rechts daneben steht der Username. Darunter befindet sich der aktuelle Rang und Score. Mittig im unteren Bereich befinden sich Knöpfe, mit denen man Das Profilbild, den Username, die E-Mail Adresse und das Passwort ändern kann. Darunter ist ein Knopf zum löschen des Users.

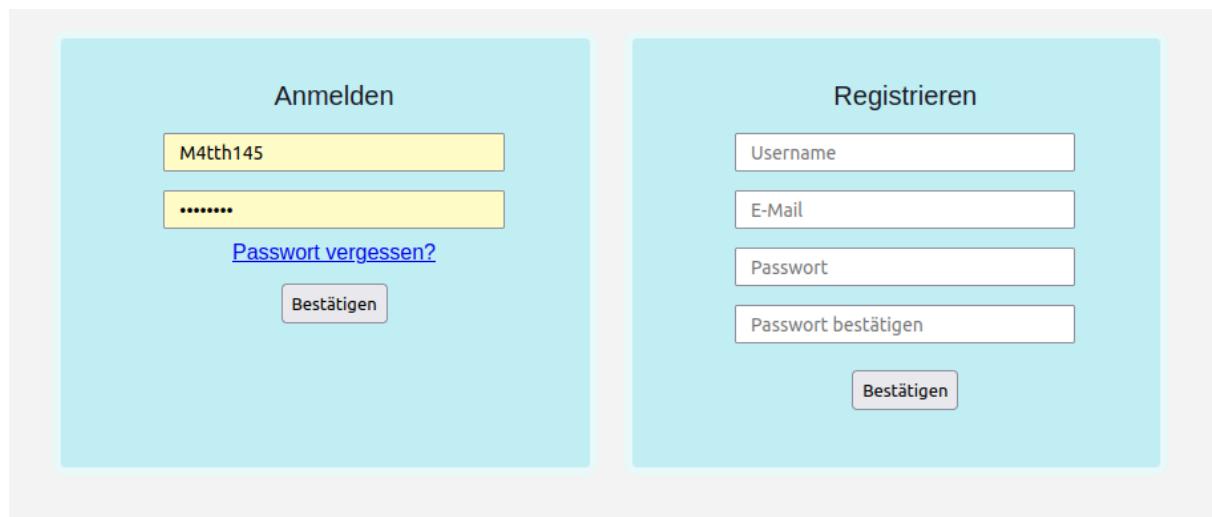


Das Abschlussprojekt findet im Laufe der schulischen Ausbildung zum informationstechnischen Assistenten, an der BBS-ME, durchgeführt. Das Projekt beginnt am 07.01.2023 und endet am 26.03.2023. Das Projekt wird von Herr Kowalewski betreut. Für das Horst Project sind 160 Schulstunden eingeplant.



Im Projekt geht es um ein Spiel, welches einem die 5 Elektrotechnikregeln spielerisch beibringen soll. Es handelt sich hierbei um ein Multiplayer-game, indem zwei Spieler gegeneinander antreten.

Abbildung 8: Result Homepage



The image shows a split-screen login interface. On the left, the 'Anmelden' (Login) form has fields for 'Username' (M4tth145) and 'Passwort' (redacted). It includes a 'Passwort vergessen?' (Forgot password?) link and a 'Bestätigen' (Confirm) button. On the right, the 'Registrieren' (Register) form has fields for 'Username', 'E-Mail', 'Passwort', and 'Passwort bestätigen' (Confirm password), each with a corresponding redacted input field. A 'Bestätigen' (Confirm) button is at the bottom.

Abbildung 9: Result Login

#### 4.4.5 Rangliste

In der Rangliste werden die besten Spieler aus der Datenbank mittels ihres Scores angezeigt. Hierfür wird eine Tabelle verwendet. In der linken Spalte ist der Rang, in der mittleren Spalte ist der Username

Aktueller Rang # 1

Win Rate 0 %

Profilbild ändern

Username ändern

E-Mail anpassen

Passwort ändern

User löschen

Abbildung 10: Result Profil

angezeigt. In der rechten Spalte wird der Score dargestellt. Früher erstelle Accounts haben einen höheren Rang.

Rang	Username	Win-Rate
1	M4tth145	0 %
2	TheUnixDaemon	0 %
3	RomanKhundadze	0 %
4	MauriceHandwerker	0 %
5	pepe	0 %

Abbildung 11: Result Score

#### 4.4.6 Chnagelog

Bei dem Changelog gibt es zwei Ansichten. Die Ansicht des normalen Nutzers, wo nur der Changelog dargestellt ist und der Changelog für die Administratoren. Er lässt neue Changelogs erstellen und bearbeiten oder vorhandene löschen. Die Änderungen werden in einer Markdown Dateien niedergeschrieben. Diese Datei wird eingebunden, umgewandelt und als HTML dargestellt. Die Formatierungen bleiben die selben.

### 4.5 Umsetzung-Backend

Um eine dynamische Website zu gewehrleisten, wird für das Backend, bei der Verbindung mit der lokalen Datenbank und dem Abrufen dessen Daten, PHP verwendet. Das Backend wird zudem objektorientiert

**Titel:** M4tth145

**Datum:** 2023-04-02

**Spielversion 0.0**

by no one

Sehr geehrter Kunde, es ist noch keine fertige Spielversion draußen. Sie müssen sich leider noch etwas gedulden.

Abbildung 12: Result Changelog User

Neuen Release erstellen

M4tth145

Game  
 Keine Datei ausgewählt.

Markdown  
 Keine Datei ausgewählt.

.....

Release entfernen

Titel

Passwort

**Titel:** M4tth145

**Datum:** 2023-04-02

**Spielversion 0.0**

by no one

Sehr geehrter Kunde, es ist noch keine fertige Spielversion draußen. Sie müssen sich leider noch etwas gedulden.

Abbildung 13: Result Changelog Admin

entwickelt, da somit eine geringere Redundanz von Quellcode verhindert werden kann und der Quellcode leichter erweitert werden kann. Auf die Klassen und deren Verwendungszweck wird im einzelnen eingegangen.

#### 4.5.1 Klassen

Es sind Klassen erstellt, damit die Website im Backend modular ist und so leichter funktional erweitert werden kann. Da es keine Main-Klasse, wie in Java, gibt, wird auch im UML auf diese verzichtet. Da PHP die Datentypen für die Variablen selbst festlegt werden diese, innerhalb des UMLs, hergeleitet anhand der zu erwartenden Datentypen, wie die aus der Datenbank. Bei unbekannten Datentypen wird der Datentyp auf „: auto“ gesetzt.

#### 4.5.1.1 Allgemeines zu den Konstruktoren

Die Konstruktoren werden allgemein bei der Website nicht näher erwähnt, da meist über diese die Klassen-Eigenschaften mithilfe der übergebenen Parameter festgelegt werden.

#### 4.5.1.2 Klasse DBconnection

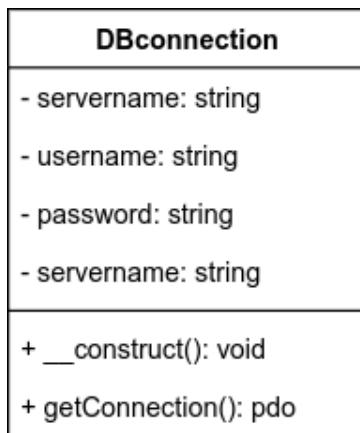


Abbildung 14: Klasse DBconnection

Die Klasse „DBconnection“ dient zur Erstellung einer Verbindung mit der SQL-Datenbank. Die Login-Daten sind innerhalb der Klasse deklariert. Die Verbindung wird mittels PDO aufgebaut, damit der Umstieg auf ein anderes Datenbanksystem(wie PostgreSQL) erleichtert wird. Zur Zeit wird als Datenbanksystem MariaDB verwendet.

#### Methode: getConnection()

Diese Methode erzeugt eine Verbindung mit der Datenbank über PDO. PDO nutzt hier die Klassen-Eigenschaften, die mittels des Konstruktors gesetzt sind, als Login-Informationen und gibt die Datenbank-Verbindung über „return“ wieder, damit Andere auf die Funktionalitäten der Datenbankverbindung zugreifen können.

#### 4.5.1.3 Klasse: login

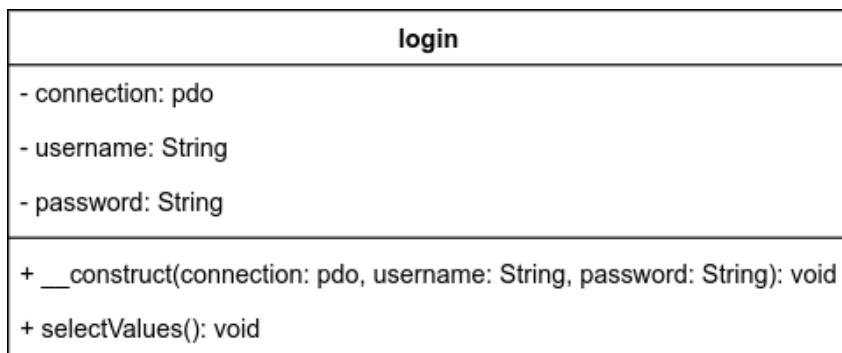


Abbildung 15: Klasse login

Die Klasse „login“ wird verwendet, um dem Client die Funktion zu bieten, sich über die Loggin-Webpage einzuloggen. Nachdem das erfolgt ist wird die „user\_id“ den nächsten Webpages, innerhalb einer Session-Variable, zur Verfügung gestellt.

#### Methode: selectValues()

Die Methode „selectValues()“ wird verwendet, um die Klassen-Eigenschaften mit der Datenbank abzugleichen. Dafür wird nach dem übergebenen Username gesucht. Wenn der Username mit einem Username aus

der Datenbank übereinstimmt, wird das übergebene Passwort gehasht und mit dem gehashten Passwort, in der Datenbank, abgeglichen. Falls diese Übereinstimmen wird die „user\_id“ des User-Kontos in einer Session-Variable gesetzt und der Client wird zur Profil-Webpage weitergeleitet.

#### 4.5.1.4 Klasse: registration

registration	
- connection: pdo	
- error: boolean	
- email: String	
- username: String	
- password: String	
- passwordVerify: String	
+ __construct(connection: pdo, email: String, username: String, password: String, passwordVerify: String): void	
+ insertValues(): void	
+ checkValues(): void	

Abbildung 16: Klasse registration

Die Klasse „registration“ dient zur Bereitstellung der Funktion, sich als Client mittels Username, Email-Adresse und einem Passwort ein User-Konto anlegen zu können. Auch hier werden die meisten Klassen-Eigenschaften mittels des Konstrukturs festgelegt. Zu Beginn ist die Klassen-Eigenschaft „error“ auf „false“.

##### **Methode: checkValues()**

Diese Methode überprüft die von dem Konstruktor gesetzten Klassen-Eigenschaften, die sich auf das anzulegende User-Konto beziehen. Darunter fällt, ob in allen Feldern etwas eingetragen wurde, ob der Username zur Verfügung steht, ob die Email-Adresse korrekt ist und das überprüfen des ersten Passwertes mit der zweiten Passwort-Eingabe. Falls bei der Überprüfung ein Fehler auftritt wird die Klassen-Eigenschaft „error“ auf „true“ gesetzt und falls keine auftritt kriegt man dementsprechend eine passende Antwort.

##### **Methode: insertValues()**

Diese Methode wird unmittelbar nach der Methode „checkValues()“ aufgerufen. Sie kann nur korrekt ausgeführt werden, wenn die Klassen-Eigenschaft „error“ nicht auf „true“ gesetzt ist. Die Aufgabe der Methode ist, die in der Klassen-Variablen gespeicherten Client-Informationen, die durch die Methode „checkValues()“ überprüft wurden, in die Datenbank einzuspeisen. Dabei werden für den User, innerhalb der Datenbank, Basiseigenschaften gesetzt, wie das Datenbanken-Attribut „rights“ und „superRights“ zu Beginn auf „false“ gesetzt sind. Zudem wird auch ein Standardprofilbild in dem Datenbanken-Attribut „user\_picture“ definiert.

#### 4.5.1.5 Klasse: ranking

Die Klasse „ranking“ dient für das Aufrufen und ermitteln der User bezogenen Daten, wie der Rang des Users und dessen letzten fünfzehn Runden. Es wird auch für das Aufrufen der Rang-Tabelle verwendet.

##### **Methode: getUserRounds()**

Dieser Methode wird eine „user\_id“ innerhalb des Parameters übergeben. Anhand dieser werden aus der Datenbank die letzten fünfzehn Runden, User bezogen, als HTML zurückgegeben. Um diese Daten

ranking
- connection: pdo
+ __construct(connection: pdo): void
+ getUserRounds(user_id: int): void
+ getUserRank(user_id: int): int
+ getRankTable(): void

Abbildung 17: Klasse ranking

aus der Datenbank zu erhalten wird bei der Abfrage INNER JOIN verwendet, um die Einträge von der Tabelle „Scores“, „Rounds“ und „Users“ miteinander auszugeben.

#### **Methode: getUserRank()**

Dieser Methode wird eine „user\_id“ innerhalb des Parameters übergeben. Es werden alle User anhand ihres „full\_score“, innerhalb der Tabelle „Users“, absteigend sortiert. Diesen wird eine Rang-Nummer zugewiesen. Daraufhin wird innerhalb der zuvor erstellten Rangliste anhand der „user\_id“ nach dem zugehörigen Rang gefiltert. und als INT mithilfe von „return“ zurückgegeben.

#### **Methode: getRankTable()**

Innerhalb dieser Methode werden alle User anhand ihres „full\_score“, innerhalb der Tabelle „Users“, absteigend sortiert. Diesen wird eine Rang-Nummer zugewiesen. Es werden absteigend die ersten dreißig Einträge innerhalb einer HTML-Tabelle, mithilfe von „echo“, ausgegeben. Diese Einträge enthalten folgende Informationen: die Rang-Nummer, den Username und die Win-/Lose-Rate.

#### **4.5.1.6 Klasse: profile**

Die Klasse „profile“ wird für die Abfrage und Manipulation von User-Informationen verwendet. Die Manipulation von Informationen, durch SETTER, wird nur von zweiten Klassen(changeData) durchgeführt. Zudem wird innerhalb dieser Klasse das Profil des jeweiligen Users geladen. Diese Klasse nutzt Methoden der Klasse „ranking“.

#### **Methode: getMyProfile()**

Anhand der Klassen-Eigenschaft „user\_id“, die von dem Konstruktor gesetzt ist, werden aus den User-Informationen, innerhalb der „Users“ Tabelle, den Rang, der mithilfe der „getUserRank()“ Methode von der Klasse „ranking“, ermittelt wird und den letzten fünfzehn Runden des Users, die mithilfe der „getUserRounds()“ Methode ,von der Klasse „ranking“, ermittelt wird, ein Profil generiert. Das Profil enthält am Ende den Username, die Win-/Lose-Rate in Prozent, das Profilbild, den aktuellen Rang und die letzten gespielten fünfzehn Runden. Zudem enthält es unten Buttons, die zu der „changeData.php“ Webpage weiterleiten, um dort seine User-Informationen anzupassen.

#### **Die GETTER-Methoden**

Innerhalb dieser Klasse gibt es diese GETTER: „getUsername()“, „getUsermail()“, „getPassword()“, „getPicture()“, „getRights()“, „getSuperRights()“ und „getFull\_score()“. Mit diesen GETTERN werden ausschließlich Datenbankinformationen aus der Tabelle „Users“ geladen und übergeben. Die Datenbank-Request wird im Konstruktor, anhand der übergebenen „user\_id“, innerhalb des Parameters, durchgeführt und in der Klassen-Eigenschaft „relresult“ gespeichert. Die GETTER greifen daraufhin auf diese Datenbank-Response, in Form von einem PDO-Array zu. Dieser ist in der Klassen-Eigenschaft „result“ gespeichert.

#### **Methode: setUserData()**

Diese Methode wird ausschließlich von den SETTER-Methoden verwendet und modifiziert ausschließlich die „Users“ Tabelle. In dieser Methode wird über einen modifizierten SQL-Query, innerhalb der PREPARE-Funktion von PDO, ein „UPDATE“ durchgeführt. Das „UPDATE“ gilt für den jeweiligen

profile
<p>- connection: pdo - result: array - user_id: int</p> <p>+ __construct(connection: pdo, user_id: int): void + getMyProfile(): void + getUsername(): String + getUsermail(): String + getPassword(): String + getPicture(): String + getRights(): boolean + getSuperRights(): boolean + getFull_score(): float - setUserData(categorie: String, data: auto): void + setUsername(username: String): void + setUsermail(user_email: String): void + setPassword(password: String): void + setPicture(user_picture: String): void + setRights(rights: boolean): void + verifyPassword(password: String): boolean + deleteUser(): void</p>

Abbildung 18: Klasse profile

User, der mit der Klassen-Eigenschaft „user\_id“ übereinstimmt. Es wird Anhand der übergebenen Parameter entschieden, welches Attribut welchen Wert erhalten soll.

### Die SETTER-Methoden

Innerhalb dieser Klasse gibt es diese SETTER: „setUsername()“, „setUsermail()“, „setPassword()“, „setPicture()“ und „setRights()“. Diesen Methoden wird der zu setzende Wert über den Parameter übergeben. Der Datentyp entspricht dem Datentyp des Attributes, innerhalb der Datenbank. Jeder SETTER ruft die Methode „setUserData()“ auf. Die SETTER übergeben an die Methode „setUserData()“ die Spalte/das Attribut und den neuen Wert, für den User spezifischen Eintrag, innerhalb der gewünschten Spalte.

### Methode: verifyPassword()

<sup>1</sup> Diese Methode überprüft das über den Parameter übermittelte Passwort mit dem echten User-Passwort ab. Das echte User-Passwort wird mit dem GETTER „getPassword()“ ermittelt. Für das Überprüfen des Passwortes wird eine PHP eigene Funktion mit der Bezeichnung „password\_verify()“ verwendet. Diese hasht das über den Parameter eingehende Passwort und gleicht draufhin die hashes ab. Fall keine Fehler auftreten gibt die Funktion ein „true“ zurück. Das Ergebniss der Funktion wird über „return“ übermittelt.

### Methode: deleteUser()

Diese Methode steht die Funktion bereit einen User-Konto zu löschen. Um das zu gewährleisten werden nacheinander die Einträge, die mit dem User-Konto in Relation stehen, gelöscht. Falls User spezifische Einträge in der Tabelle „Rounds“ und „Scores“ vorhanden sind, werden zuerst über eine WHILE nacheinander die Runden gelöscht, die innerhalb der Tabellen „Rounds“ und „Scores“ festgehalten werden. Daraufhin werden alle Einträge in der Tabelle „Users“ die in Relation mit der „user\_id“ stehen gelöscht. Das entfernen der Einträge innerhalb der Datenbank wird mittels des „DELETE“ Befehls von SQL bewerkstelligt.

#### 4.5.1.7 Klasse: changeData

changeData	
- connection: pdo	
- profile: profile	
- user_id: int	
+ __construct(connection: pdo, user_id: int): void	
- verifyPassword(password: String): boolean	
- deleteSession(): void	
+ changeProfile(password: String, user_picture: auto): void	
+ changeUsername(password: String, username: String): void	
+ changeMail(password: String, user_mail: String): void	
+ changePassword(password: String, newPassword: String): void	
+ deleteUser(password: String): void	

Abbildung 19: Klasse changeData

Die Klasse „changeData“ bietet dem User sein User-Konto anzupassen. Dazu gehört das Profilbild, der Username, die Email-Adresse und Das Passwort. Es wird auch die Möglichkeit geboten das User-Konto zu schließen. Es sind in dieser Klasse Methoden der Klasse „profile“ implementiert.

### Methode: verifyPassword()

Diese Methode überprüft das über den Parameter übermittelte Passwort mit dem echten User-Passwort ab. Das echte User-Passwort wird mit dem GETTER „getPassword()“, aus der Klasse „profile“, ermittelt. Für das Überprüfen des Passwortes wird eine PHP eigene Funktion mit der Bezeichnung „password\_verify()“

<sup>1</sup> Die Methode ist nicht in Verwendung.

verwendet. Diese hasht das über den Parameter eingehende Passwort und gleicht draufhin die hashes ab. Fall keine Fehler auftreten gibt die Funktion ein „true“ zurück. Das Ergebniss der Funktion wird über „return“ übermittelt.

#### **Methode: deleteSession()**

Diese Methode wird aufgerufen, wenn ein erneutes Anmelden erzwungen werden soll. Dies wird bewerkstelligt, indem die PHP Session beendet wird und auf die Login Webpage weitergeleitet wird. Durch das Löschen der Session werden alle Variablen in dieser mit gelöscht. Somit ist nun keine „user\_id“ bekannt. Um erneut eine „user\_id“ zu erhalten muss sich erneut anmelden.

#### **Methode: changeProfile()**

Diese Methode dient zur Anpassung des User-Profilbildes. Bei einem Aufruf dieser Methode wird im Parameter ein Passwort, zur Prüfung der Identität und eine Bild-Datei übergeben. Es wird zuerst mithilfe von der Methode „verifyPassword()“ geprüft, ob das Passwort korrekt ist. Wenn ein „true“ zurückgegeben wird, wird ein Ordner mit der Bezeichnung „user\_ud=\*\*“ erstellt, falls noch nicht vorhanden. Falls das Bild kleiner als ein MB ist wird die Bild-Datei in den zuvor erstellten Ordner gesichert. Wenn die Datei größer ist gibt es eine Fehlermeldung. Nach der Sicherung in den Ordner wird der Relative Pfad des Bildes und der Wert der Klassen-Eigenschaft „user\_id“, die von dem Konstruktor gesetzt wird, an den SETTER „setPicture()“ von der Klasse „profile“ übergeben.

## 5 Planung Website

### 5.1 Planänderungen des Projektes

Die Arbeitspakete „Zugriffe loggen“ und „Backupsystem“ sind durch eine gruppentechnische Entscheidung, dass der Server und dessen Administratoren von einem Mitglied aus dem Team GameDev, übernommen wird. Dadurch ist es aufgrund von mangelnden Zugriff auf diesen Server dem WebDev Team nicht möglich diesen zu Administrieren. Die Arbeitspakete „Firewall“ und „E Mail-Server“ fallen aus zeitlichen Gründen weg, da die Entwicklung der Website verlängert werden musste.

## 6 Organisation Spiel-Entwicklung

### 6.1 Game-Engine

Eine Game-Engine ist eine Software oder Framework was auf das Entwickeln von Spielen Spezialisiert ist. Game-Engine wörtlich übersetzt bedeutet Spiel-Motor. So kann man die Software oder Framework als Motor hinter einem Spiel sehen. Eine Game-Engine übernimmt normalerweise Aufgaben wie anzeigen von Grafiken, Laden und Speichern von Objekten und lesen der User-Eingaben. Mithilfe einer Game-Engine lässt sich so schneller und leichter Spiele Entwickeln. So gibt es auch Spezialisierte Game-Engines auf bestimmte Arten von Spielen. Einige haben 3D Unterstützung, 2D Unterstützung oder VR Unterstützung. So ist ein großer Teil der Entwicklung eines Spieles die Auswahl der Game-Engine.

#### 6.1.1 Unreal Engine

Die Unreal-Engine ist eine von großen Unternehmen benutzte aber auch von Einzel-Personen benutzte Game-Engine. Unreal-Engine ist schon seit einigen Jahren die größte Game-Engine für hohe Grafik-Qualität und Open-World Spiele. Sie benutzt C++ als Sprache und konzentriert sich auf 3D Spiele. Der große Nachteil an der Unreal-Engine ist die Komplexität durch die Freiheit die der Benutzer hat.

#### 6.1.2 Unity

Unity ist eine weit verbreitete Game-Engine im App und 2D Bereich und wird von vielen als erste Game-Engine durch den Simplen Aufbau der grund Funktionen benutzt. Unity Unterstützt 2D und 3D und kann noch mit dem selben Programm Code auf viele verschiedene Systeme Compiled werden. Durch die große Anzahl an Benutzern gibt es viele Fohren und Dokumentationen zu den Funktionen von Unity. Unity benutzt C Sharp als grund Sprache zum Schreiben des Programm Codes.

#### 6.1.3 Godot

Godot ist eine im Vergleich Junge Game-Engine die sich auf 2D Spiele konzentriert. Godot ist sehr simple von den Funktionen die angeboten werden. Der große Nachteil der Godot Engine ist das eine abgewandelte Form von Python benutzt wird. Durch diese abgewandelte Form von Python ist es schwer das Wissen von Godot an anderen Projekten anzuwenden.

#### 6.1.4 Entscheidung Game-Engine

Das Team hat sich für Unity entschieden durch die Ähnlichkeit zwischen Java und C Sharp, Große Dokumentation und Vertretung im Internet, breit verteilte Funktionen und der leichten Bedienung für Anfänger.

### 6.2 Zielsystem

Das Spiel wird für den Computer entwickelt und es wird sich in erster Linie auf Windowssysteme konzentriert, da diese eine größere Nutzerbasis bieten als z.B. das Linux System.

## 7 Client-Kommunikation

### 7.1 Kommunikations-Software

Um das Projekt in der Angegebenen Zeit fertig stellen zu können hat sich das Team entschieden eine Dritt-Anbieter Software für die Kommunikation zwischen den Clients zu benutzen. Dies bietet den Vorteil das kein System entwickelt werden muss was die Daten übers Internet überträgt.

#### 7.1.1 Riptide

Riptide ist eine von der Unity-Community erstelle Software die Client-Server Kommunikation übernimmt und eine stabile Basis für die Entwickler bietet. Eine der Vorteile von Riptide ist das es Open-Source ist und dadurch leicht zu verstehen ist durch den direkten Zugriff auf den Programmcode. Der größte Nachteil zu dem Zeitpunkt des Projektes ist das die Dokumentation des Riptide Projektes und dessen Anwendung nicht auf dem neusten Stand und nicht ausreichend ist.

#### 7.1.2 Netcode for GameObjects (NGO)

Netcode for GameObjects oder auf Deutsch Netcode für Spiel-Objekte ist eine von Unity entwickelte Möglichkeit mit einer „High-Level-Networking-Bibliothek“<sup>1</sup> die Multiplayer Funktion in ein Spiel zu implementieren. Dadurch das NGO auch von Unity Entwickelt wird ist die Zusammenarbeit stabil, sicher und effektiv. Der einzige Nachteil ist das NGO bis Ende 2022 noch in der BETA Version war und dadurch noch viele Baustellen hat bevor es ganz stabil ist.

#### 7.1.3 Entscheidung Kommunikations-Software

Durch die bessere Dokumentation hat sich das Team entschieden Netcode für Spiel-Objekte für das Projekt zu benutzen. Dadurch das NGO schon seit einigen Jahren frei verfügbar ist und von Unity selbst entwickelt wird sind viele Foren-Posts und Lern-Artikel im Internet zu finden. Dadurch ist es für das Team leichter NGO ins Projekt zu implementieren. Dies ist ein großes Problem bei Riptide was keine große Community von Benutzern bieten kann.

---

<sup>1</sup>Zitat Unity 3D Multiplayer Docs

## 8 Kommunikations-Typ

Für die Multiplayer funktion des Spiels gibt es zwei verschiedene Kommunikations-Typen. Diese beiden Typen bieten ihre eigenen Vor- und Nachteile.

### 8.1 Dedicated Game Server (DGS)

<sup>1</sup> Ein DGS ist eine Version des Spiels ohne grafische Oberfläche. In der Dokumentation von Unity steht als Erklärung zu DGS „Dedizierte Server simulieren Spielwelten, ohne direkte Eingabe oder Ausgabe zu unterstützen, mit Ausnahme dessen, was für ihre Verwaltung erforderlich ist. Spieler müssen sich mit separaten Client-Programmen mit dem Server verbinden, um das Spiel zu sehen und mit ihm zu interagieren.“ <sup>2</sup>. Das positive an einem DGS ist die Pflicht das die Clients nur mit einem Gameserver Funktionieren. Dadurch kann garantiert werden das die neuste Version benutzt wird. Allerdings ist es schwer ohne genaues wissen über eine Server-Client Kommunikation diese gut und zuverlässig umzusetzen.

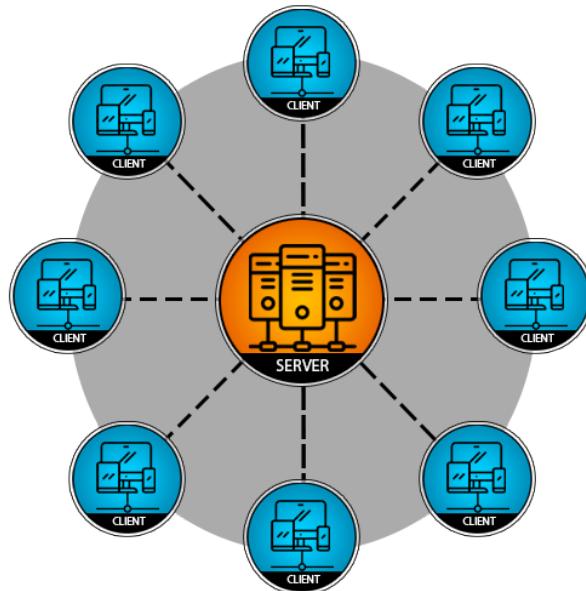


Abbildung 20: Dedicated Game Server (DGS)

### 8.2 Client Hosted (Listen Server)

Client Hosted ist die Umsetzung das ein Client seine Session <sup>3</sup> als Server und Client benutzt. Dadurch muss kein extra Server dauerhaft laufen und das spiel kann auch nach Beendung der Entwicklung gespielt werden. Das besondere an den Listen Server ist das auf der Host Session die selbe Applikation läuft wie auf den anderen Clients. Der einzige unterschied ist das der Host als Schnittstelle zwischen allen Clients agiert. Durch diesen voreilt ergibt sich auch der größte Nachteil. Dadurch das der Host als Server agiert wird die durch die höhere Benutzung der Netzwerk Verbindung Röhrgeschwindigkeit des Netzwerkes langsamer. Deswegen sind Listen Server bei größeren Spielen so selten. Ein weiterer Nachteil ist es das der Host seine Internet Verbindung teilen muss damit die Clients den Host übers Internet finden können. Um dieses große Sicherheits-Problem zu schließen und dem Host eine größere Sicherheit zu bieten, bietet Unity ein Service namens „Relay Service“ an.

### 8.3 Relay Service

Der „Relay Service“ ist ein Service der von Unity angeboten wird. Der Service kann bis zu einem gewissen Punkt kostenlos benutzt werden. In der Kostenlosen Version haben die Entwickler die Möglichkeit bis zu

<sup>1</sup> Dedizierter Gameserver

<sup>2</sup> Zitat : Unity 3D Multiplayer Docs

<sup>3</sup> Eine Sitzung des Clients

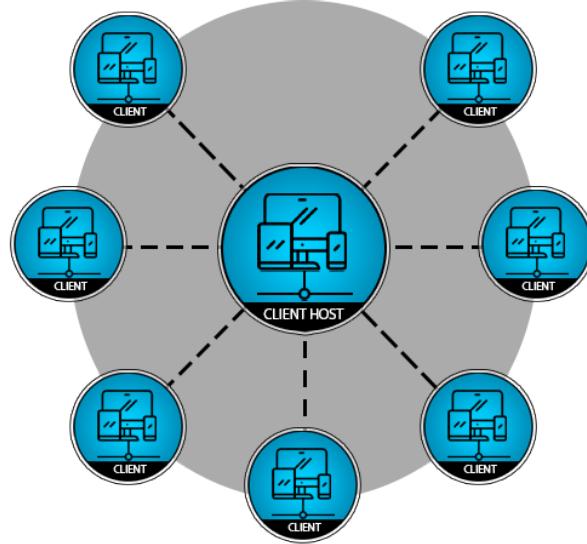


Abbildung 21: Client Hosted (Listen Server)

einer Transfer Benutzung von 150GiB Monatlich zu benutzen. „Der Relay-Server-Ansatz verwendet dies, um Daten zwischen zwei Spielern zu senden. In einem Listen-Server-Szenario würden sich der Host und alle Clients mit demselben Listen-Server verbinden. Die Clients würden dann Pakete aneinander senden, indem sie sie an den Relay-Server senden und ihm mitteilen, dass er sie an den richtigen Client umleiten soll.“<sup>3</sup>. Die hauptfunktion des Relay Services in Horst besteht daraus dem Client die Möglichkeit zu bieten sich übers Internet mit einem Host zu verbinden und zu kommunizieren. Sollte kein Relay Service benutzt werden könnte der Client nur dann sich mit einem Host verbinden wen dieser einen Port in sein Netzwerk freischaltet. Das freischalten eines Ports öffnet eine unsichere Lücke in der Sicherheit des Netzwerkes wodurch jeder die Geräte die mit diesem verbunden sind angreifen kann.

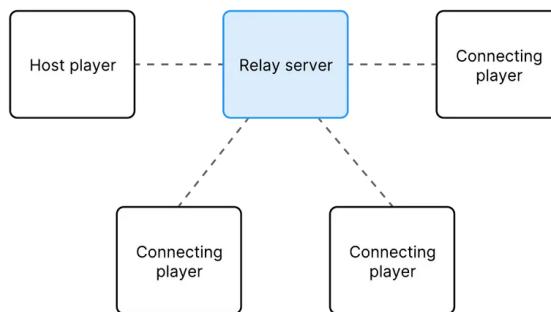


Abbildung 22: Relay Server

#### 8.4 Entscheidung Kommunikations-Typ

Damit das Spiel so lange wie möglich spielbar ist und die Entwicklung so zeiteffizient wie möglich ist hat sich das Team dazu entschieden ein Listen Server zu entwickeln. Dies bietet einige Vorteile in der Entwicklung des Programmcodes und der Spieler Erfahrung. Die Größten Vorteile und gründe der Entscheidung sind die Kosten die durch ein Listen Server gespart werden und die Leichtere Entwicklung durch das bessere Verständnis eines Listen Servers des Teams. Welche nicht im DGS geboten ist.

<sup>3</sup>Quelle: Unity 3D Multiplayer Docs

## 9 Implementation Client-Kommunikation

Während der Implementation der Client-Kommunikation ist das Team auf mehrere Probleme gestoßen.

### 9.1 Implementation Netcode for GameObject (NGO)

Beim Implementieren von NGO war das größte Problem die mangelhafte Dokumentation der von Unity verifizierten Videos, Artikel und Dokumentation's-Einträge. Durch diese mangelhafte Dokumentation sind einige Fehler aufgetreten die viel zeit zum herausfinden beansprucht haben.

### 9.2 Implementation Relay Service

Das Implementieren des Relay Services ist ohne probleme durch die genaue Dokumentation der Unity Services verlaufen.

#### 9.2.1 Klasse: RealyManager

Die Klasse RelayManager erbt von Singleton<RelayManager>, damit diese Klasse nur einmal auf jeder Instanz laufen kann. Dies ist eine sicherheits-Vorkehrung damit es zu keinen konflikten kommt sollte ein Spieler versuchen ein Host und Client zugleich zu sein. Um zwischen verschiedenen Entwicklung Stadien zu wechseln bietet der Relay Service von Unity die Möglichkeit sogenannte „Environments“ oder Umgebungen zu erstellen. Dies ist vor allem nützlich sollte die Entwicklung nach Veröffentlichung des Spiels weitergehen das die Spieler nicht auf den Selben Relay Servern verbunden werden wie die Entwickler. Diese Umgebung wird mit dem „\_enviroment“ String festgelegt. Um für den Host einen wert fest zu geben wird „\_maxConnections“ benutzt. Diese Nummer legt fest wie viele Clients sich maximal auf einem Host verbinden dürfen. Als weitere sicherheits-Vorkehrung wird beim starten der klasse ein check abgefragt ob schon eine Relay Verbindung besteht. Sollte eine Verbindung bestehen wird der versuch eine neue Verbindung aufzubauen abgebrochen. Dieser Check wird von „isRelayEnabled“ ausgeführt. Zur Kommunikation zwischen geräten wird immer ein Transport Protokoll benutzt. Um über Unity mit anderen Geräten kommunizieren zu können bietet Unity ihr eigenes Transport Protokoll. Dies bedeutet für die Entwickler das sie dies nich selbst Neuschreiben müssen und das Unity auf jedenfall versteht was bei den Clients ankommt. Dieses Transport Protokoll wird in „transport“ festgelegt.

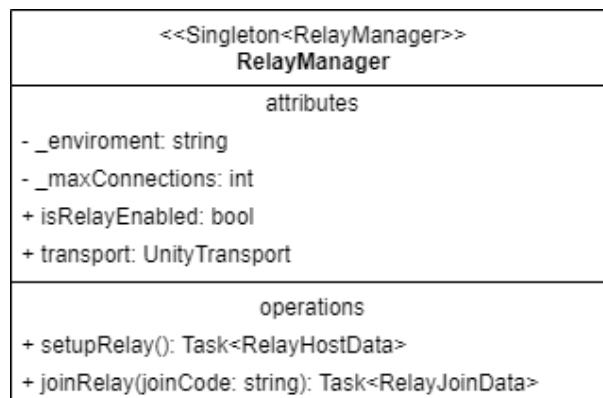


Abbildung 23: UML RelayManager

#### 9.2.1.1 Funktion: setupRelay()

Die Funktion „setupRelay“ erstellt eine Verbindung mit dem RelayService von Unity als Host und hat damit mehr rechte innerhalb der Verbindung als die Clients. Zu den Sonderrechten hat der Host zugriff auf die ID, Name und weiteren Daten die zum verifizieren genutzt werden. Innerhalb der Funktion wird ein Objekt erstellt was alle Daten die zur Verbindung zwischen dem Host und Relay Server gebraucht werden. Wie die IP-Adresse des Hosts, des Ports die der Host benutzt zum übertragen und die Verbindung ID die zum identifizieren der Relay Instanz genutzt wird.

### 9.2.1.2 Funktion: joinRelay()

Zum Beitreten einer Relay Instanz wird die „joinRelay“ Funktion benutzt. Diese Funktion benötigt einen „Lobby Code“. Dieser Lobby Code wird beim ausführen der „setupRelay“ Funktion dem Host gegeben. Wie der Spieler den Lobby Code bekommt bleibt dem Spieler übrig. Mithilfe des Lobby Codes wird nach einer Aktiven Relay Instanz mit demselben Lobby Code gesucht. Sollte einer gefunden werden wird der Spieler seine IP Adresse und Port an den Relay Server senden. Mithilfe dieser Daten kann der Relay Server verifizieren das immer noch die Selbe Person verbunden ist und nicht ersetzt wurde.

### 9.2.1.3 Programm Ablauf setupRelay()

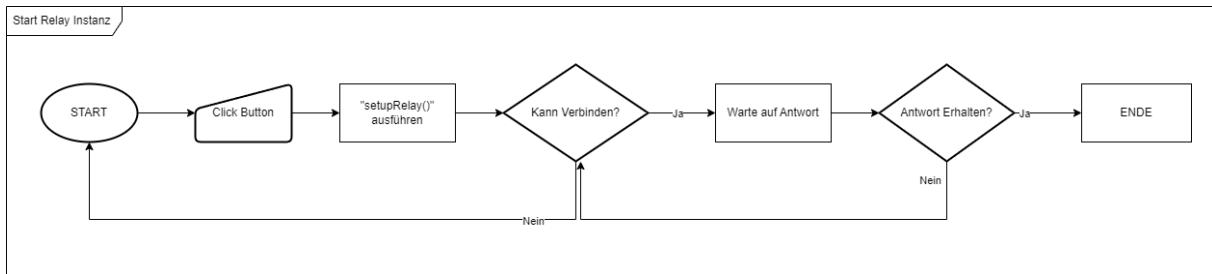


Abbildung 24: Programm Ablauf Diagramm setupRelay

Beim Start muss der Spieler den „Start Lobby“ Button betätigen um den Ablauf zu starten. Nachdem betätigen des Buttons wird die Funktion „setupRelay“ ausgeführt. Innerhalb dieser Funktion versucht die Applikation sich über Internet mit den Unity Relay Servern zu verbinden und eine Relay Instanz zu erstellen. Sollte das Erstellen fehlschlagen oder der Spieler keine Verbindung mit den Unity Relay Servern aufbauen können wird eine Fehlermeldung ausgegeben die im Log gespeichert wird. Bei einer erfolgreichen Verbindung und erfolgreichen erstellen einer Relay Instanz schickt der Unity Relay Server eine Antwort mit allen erforderlichen Daten an den Spieler zurück. Sollte die Antwort nach 10 Sekunden nicht erhalten worden sein versucht die Applikation eine neue Verbindung mit dem Unity Relay Server aufzubauen und eine neue Instanz zu erstellen. Beim erfolgreichem erhalten der Antwort ist der Spieler der Besitzer der Lobby und der Relay Instanz und wechselt zur Lobby Scene.

### 9.2.1.4 Programm Ablauf joinRelay()

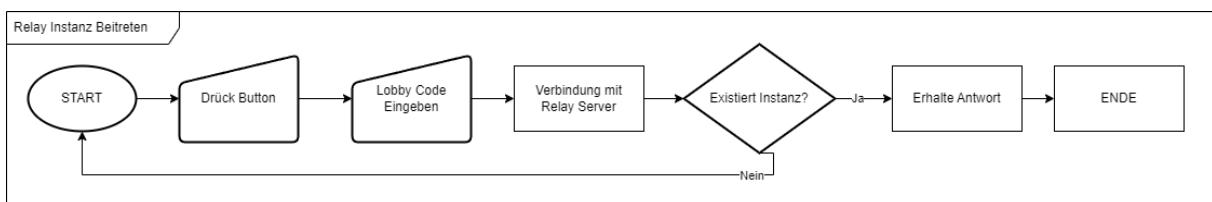


Abbildung 25: Programm Ablauf Diagramm joinRelay

Um einer Lobby Beizutreten muss der Spieler zum Start den „Join Lobby“ Button Betätigen. Mit dem Betätigen des Buttons öffnet sich ein Menu zum eingeben des „Lobby Codes“. Nach dem Eingeben des „Lobby Codes“ erstellt die Applikation eine temporäre Verbindung mit den Unity Relay Server und überprüft ob derzeit eine Instanz mit dem passenden „Lobby Code“ existiert. Sollte keine Instanz mit dem vom Spieler eingegebenen „Lobby Code“ existieren wird der Spieler wieder zurück zum Hauptmenu der Applikation geleitet mit einer Fehlermeldung das derzeit keine Instanz mit dem angegebenen „Lobby Code“ existiert. Sollte eine Instanz mit dem „Lobby Code“ existieren wird die temporäre Verbindung mit den Unity Relay Server zu einer festen Verbindung gewechselt. Nach dem Wechsel zu einer festen Verbindung wird der Applikation alle freigegebenen Daten übertragen. Nachdem die Applikation alle Daten bekommen hat ist sie mit dem Unity Relay Server verbunden bis ein Verbindungsabbruch stattfindet oder der Spieler die Verbindung mit dem „Verlassen“ Button beendet.

## 9.3 Implementation Listen Server

Bei der Implementation des Listen Servers sind durch mangelhafte Dokumentation des NGO Pakets einige Fehler aufgetreten.

### 9.3.1 Verbindungs-Fehler

Dieser Fehler ist vor der Implementation des Unity Relay Service aufgetreten. Nachdem die Implementation der Server-Client Kommunikation nur lokal getestet wurde ist dem Team aufgefallen das die Kommunikation Lokal Funktioniert allerdings nicht übers Internet Online. Daraufhin musste das Team die Server-Client Kommunikation vom Grund auf überarbeiten und mit dem Relay Server zusammen neu Implementieren.

### 9.3.2 RPC-Fehler

Bei der Implementation der UI-Synchronisation sind Probleme mit den RPC Funktionen von NGO aufgetreten. Dieses Problem hat das Team für 6 Wochen aufgehalten bis es identifiziert werden konnte. Der Grund des Problems ist die Struktur der Spieler-Server Aufteilung. Dadurch das Horst das erste Multiplayer Spiel für das Team darstellt ist die Struktur zwischen den Server-Funktionen und Spieler-Funktionen unklar. Diese fehlende Server-Spieler Struktur ist der Grund warum die Implementation der RPC's nicht funktioniert.

## 9.4 Implementation Remote Procedure Calls (RPC's)

Remote Procedure Calls oder RPC's ist eine weit verbreitete Methode zwischen Clients und Server zu Kommunizieren. NGO bietet eine eingebaute Methode um Server und Client RPC's zu senden und Empfangen. Ein RPC ist eine Funktion die auf einer festzulegenden Instanz den enthaltenden Programm Code ausführt. In NGO sind zwei Instanzen festgelegt. Server-RPC's und Client-RPC's. Ein Server-RPC wird von einem Client geschickt und wird auf dem Server oder Host ausgeführt. Ein Client-RPC wird dafür entweder an Alle Clients oder an einen bestimmten Client gesendet und ausgeführt. Vor allem sind RPC's viel benutzt durch die geringe Netzwerk Auslastung. Dadurch das die RPC's nur dann aufgerufen werden wenn eine Aktion erfordert wird gibt es keine überflüssige Nutzung der Netzwerk Verbindung.

### 9.4.1 Server-RPC

Beim aufrufen eines Server-RPC's wird übers Netzwerk und den Server der Verbindung den Inhalt der Funktion gesendet damit dieser den ausführt. Dies ist nützlich für Rechnungen, hochladen von Einträgen oder verifizieren von Eingaben des Spielers.

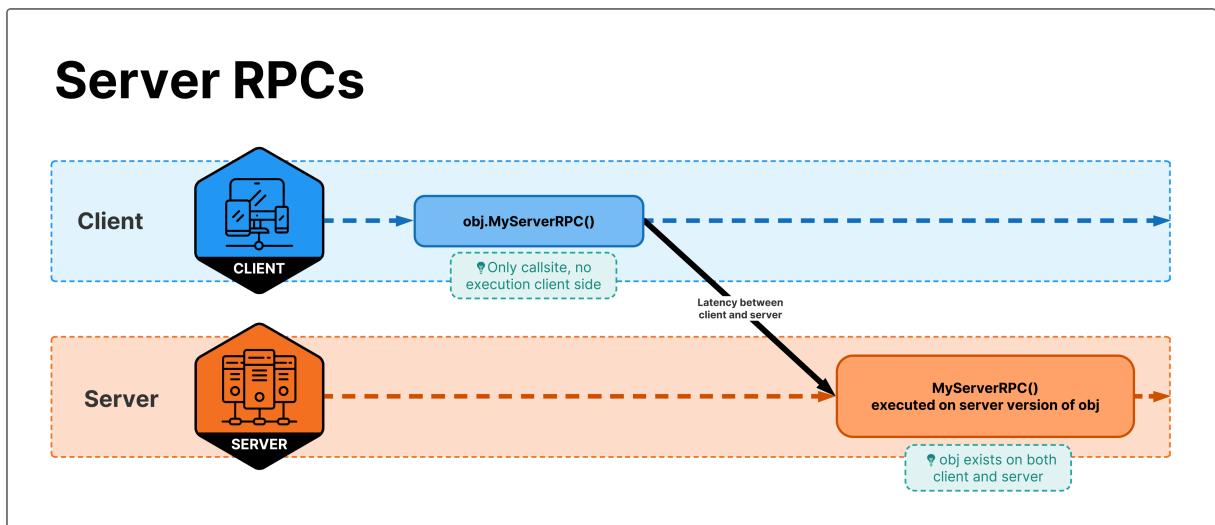


Abbildung 26: Server RPC Grafik

In Fall von Horst gibt es keinen Server sondern ein Host und Clients. Aus diesem Grund sieht die Kommunikation anders aus. Ein Server-RPC wird von einem Client zum Server/Host abgesendet. In einem

RPC ist immer ein Code-Block und eventuell Parameter enthalten. Nach langer Recherche hat das Team herausgefunden das Server-RPC's immer von einem Player-Prefab gestartet werden müssen. Deswegen gab es große Probleme mit der Spieler Struktur. Aufgrund dessen gab es Probleme die Server-Funktionen und Client-Funktionen auseinander zu halten.

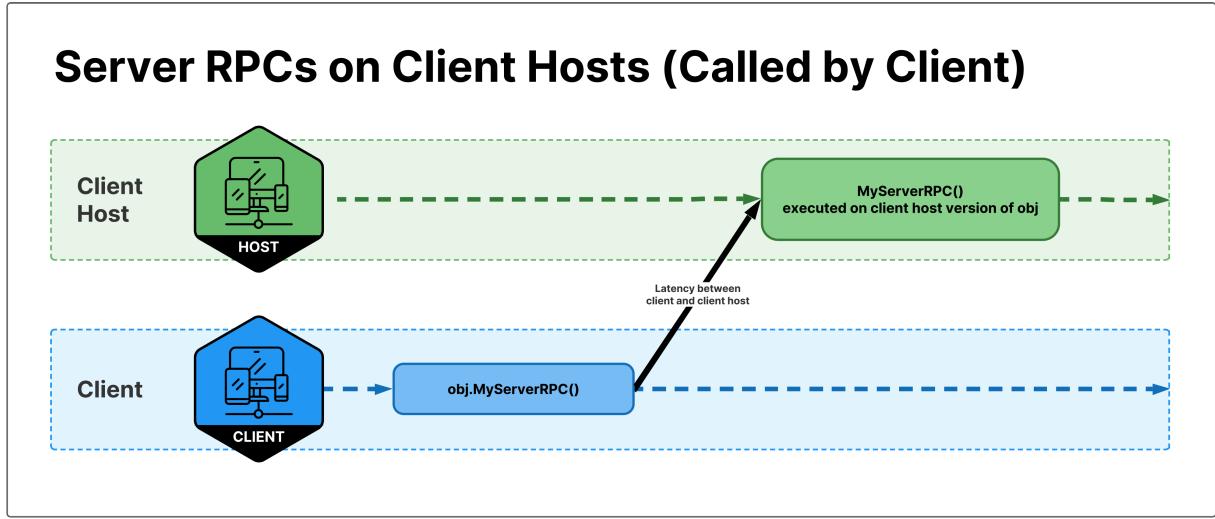


Abbildung 27: Client zu Host

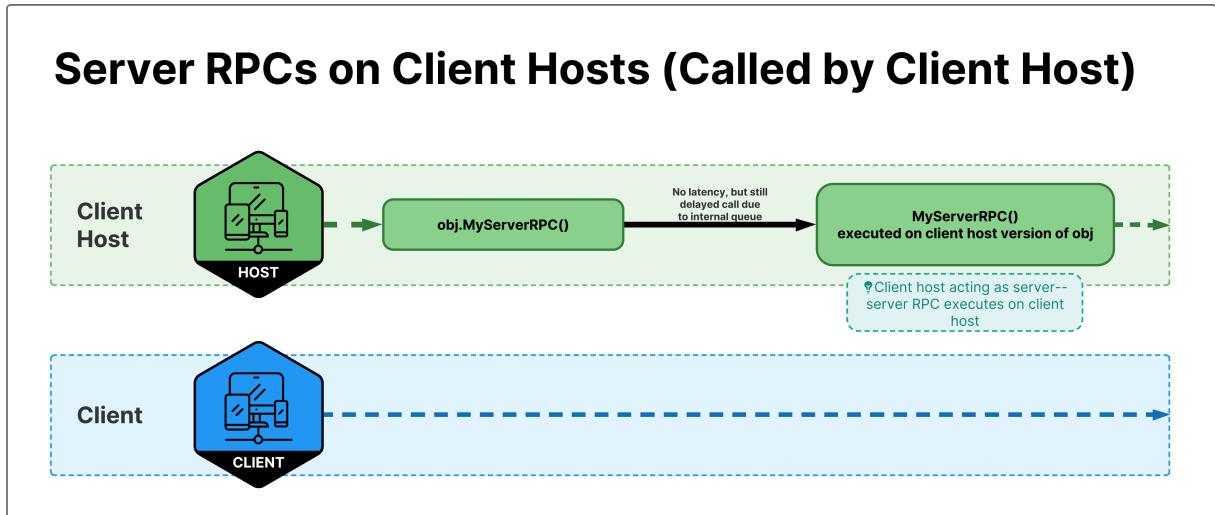


Abbildung 28: Server-RPC | Horst zu Horst

#### 9.4.2 Client-RPC

Ein Client-RP sendet von einem Server zu allen oder einem bestimmten Client Code-Blöcke und eventuell Parameter. Diese Code-Blöcke oder Parameter werden auf den Clients lokal ausgeführt.

Dadurch das im Fall von Horst kein Server sondern ein Host verwendet wird sieht die Umsetzung anders aus.

Mit einem Client-RPC ist es dem Server/Host auch möglich einen Bestimmten Client einen RPC zu senden. Dies würde so dargestellt werden:

## Client RPCs

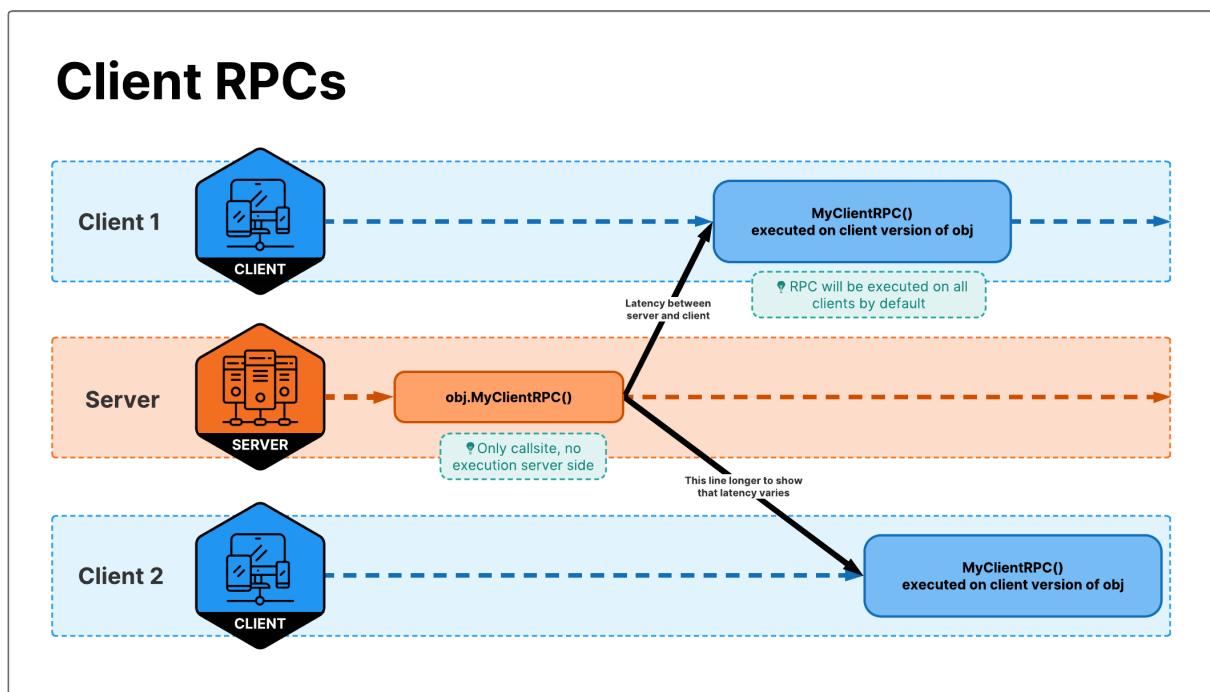


Abbildung 29: Client RPC

## Client RPCs on Client Hosts (Called by Client Host)

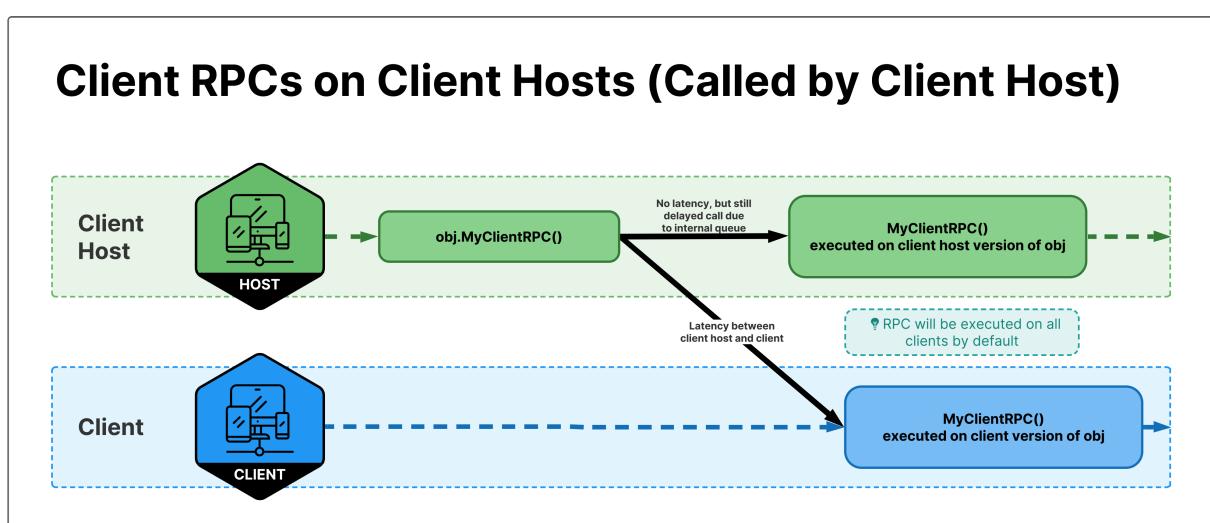


Abbildung 30: Client RPC von Horst

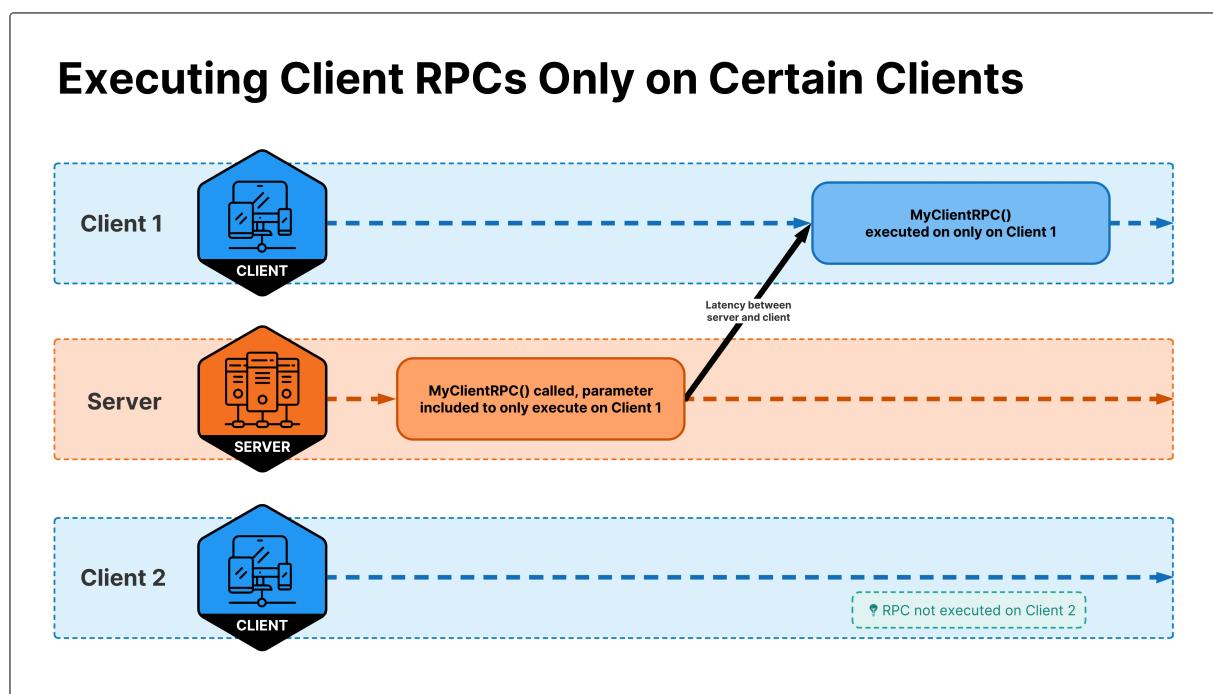


Abbildung 31: Client RPC zu Bestimmten Client

## 10 Interne Kommunikation

Zur internen Kommunikation gehört die von dem Web-Dev entwickelte API. Damit die Clients sich einloggen können und ihre Scores lesen und hochladen können.

### 10.1 Implementation API

Zum anfragen und senden von Daten mit der API wurde ein „API Manager“ entwickelt.

#### 10.1.1 Klasse apiManager

Die Klasse apiManager hat die Aufgabe sich um alle API-Calls zu kümmern. Zu diesem Zeitpunkt besitzt sie nur eine Beispiel Funktion. In er Klasse wird die Antwort der API als json gespeichert und in einem Struct umgewandelt. Dieser Struct kann dann innerhalb anderer Klassen als Datentyp benutzt werden. Der Private String „\_baseAdress“ besitzt die API Adresse worüber die API-Calls getätigt werden.

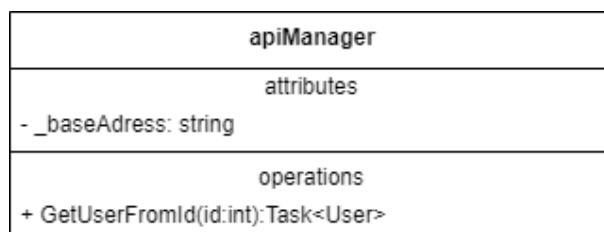


Abbildung 32: Klasse apiManager UML

##### 10.1.1.1 Funktion: GetUserFromId()

Die Funktion „GetUserFromId()“ ist vom Typ „async Task<User>“. Die Funktion muss asynchron laufen können durch die Externe Kommunikation übers internet mit der API. Durch diese Externe Kommunikation können Zeitverschiebungen auftreten die dazu führen würden das kein Wert zurück gegeben werden kann sollte die Funktion nicht asynchron sein. Mithilfe der UnityWebRequest Klasse die von NGO gestellt wird kann ein API-Call ganz simple an die API gesendet werden. Dadurch das die API ein Json String zurück gibt ist es nicht möglich diesen als Datentyp zu verwenden. Aus diesem Grund wird ein erstellter Struct dafür verwendet.

#### 10.1.2 Struct User

Ein Struct ist ein Datentyp der selbst zusammen gestellt werden kann und die Inhalte anpassen kann. So wurde für das Managen des Eingeloggten users ein User Struct erstellt. Dieser Struct enthält die User ID, User Profile Link und Username. Mithilfe dieses Structs wird der Spieler identifiziert. Um den Struct mithilfe von RPC's an alle Clients und den Host zu senden muss er Serialisierbar sein. Um den Struct zu Serialisieren gibt NGO ein Interface die das Serialisieren übernimmt. Dieses Interface ist „INetworkSerializable“

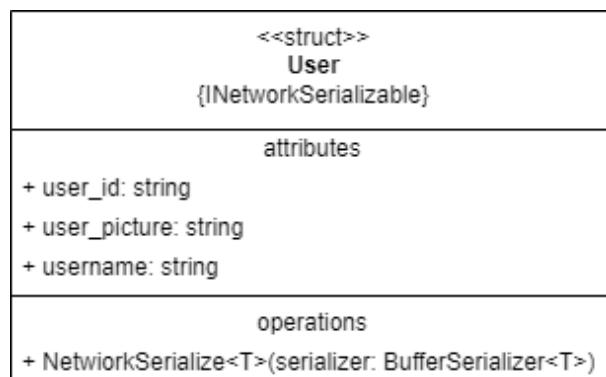


Abbildung 33: User Struct UML

## 11 Nutzeroberfläche

### 11.1 Inventar UI

Das Inventar ist auf 4 Gegenstände beschränkt und wird als 4 boxen oben links dargestellt. Beim scrollen durch das Inventar wird bei dem aktiven Gegenstand eine gelbe Markierung als Hintergrund hinzugefügt. Durch diese Markierung ist es dem Spieler möglich schnell ohne genaues hinschauen zu erkennen welches Gegenstand gerade ausgerüstet ist.

### 11.2 Developer Tool

#### 11.2.1 LogDisplay

Um auch in den Developer Builds <sup>1</sup> den Log genauso wie in Unity auslesen zu können hat das Team ein Simples Tool entwickelt was diesen in der linken unteren ecke des Builds ausgibt. Durch dieses Tool ist es möglich auf allen Clients gleichzeitig den Log auszulesen. Damit spart das Team zeit in der Entwicklung durch das nur einmalige ausführen um das Problem auszulösen.

---

<sup>1</sup> Kompilierte Versionen die nur für die Entwickler erstellt werden. Developer Builds geben auch genaue Fehlermeldungen aus.

## 12 Gameplay

### 12.1 Inventar System

Damit der Spieler mit seinem Umfeld wurde von dem Team ein Inventar-System entwickelt. Der Spieler kann mit ausgewählten Gegenständen in der Spiel-Welt mithilfe Maus-Klicks interagieren. Die Gegenstände werden mithilfe der Kamera und „RayHits“ Identifiziert. Ein „RayHit“ sendet eine Unsichtbare Linie von der Kamera zu dem durch Mausklick Ausgewählten Punkt. Nachdem das Objekt Erfasst wurde wird überprüft das dieses Interagierbar ist.

#### 12.1.1 Umsetzung Inventar-System

##### 12.1.1.1 Klasse interactionManager

interactionManager	
attributes	
- [SerializeField] _interactRadius: float	
- [SerializeField] _hasPickUpItem: bool	
- [SerializeField] _itemHolder: GameObject	
- [SerializeField] _interactableTag: string	
- [SerializeField] _inventory: GameObject[]	
- [SerializeField] _inventorySize: int	
- _activeInventorySlot: int	
- _inputs: UserInputs	
operations	
- mouseInteraction()	
- pickUpItem(objectRaycastHitInfo)	
- holdItem(item: GameObject)	
- dropItem()	
- mouseScrollValue()	
- inventorySlotSelection(selection: bool)	
- storeItems(selectedObject: GameObject)	
- unpackItem()	
- setActiveInventorySlot(num: int)	

Abbildung 34: I\_interactable UML

Die Klasse interactionManager kümmert sich um das Interagieren mit Objekten und liegt auf dem Spieler. Der float „\_interactRadius“ legt den Radius fest indem der Spieler mit Gegenständen interagieren kann. Sollte ein Objekt weiter entfernt sein als durch den float angegeben wird nichts passieren und der Spieler wird nicht mit dem Objekt Interagieren. Der bool „\_hasPickUpItem“ ist zur Überprüfung ob der Spieler derzeit schon ein Aktives Objekt besitzt. Sollte der Spieler ein Aktives Objekt besitzen wird abgebrochen und der Spieler Interagiert nicht mit dem Objekt. Das GameObject „\_itemHolder“ ist der „Transform“-Punkt wo das Aktive Item sollte es „Gehalten“ werden können angezeigt wird. Der string „\_interactableTag“ legt den Tag fest der für Interagierbare Objekte benutzt wird. Das GameObject Array „\_inventory“ ist das Inventar selbst. Innerhalb dieses Arrays werden alle Gegenstände des Spielers gespeichert. Die Größe des Inventars wird durch den Int „\_inventorySize“ festgelegt. Der Aktive Inventar Slot wird mithilfe des Ints „\_activeInventorySlot“ festgelegt. Dieser Int ist veränderbar. Die Eingaben des Spielers werden mithilfe des UserInputs „\_inputs“ übernommen.

### **Funktion mouseInteraction()**

Die Funktion mouseInteraction() kümmert sich um das Auswählen eines Gegenstandes mit der Maus. Sollte das Ausgewählte Gegenstand kein „I\_Interactable“ Komponente Besitzen wird nichts passieren. Sollte das Objekt ein PickUp Gegenstand sein so wird es ins Inventar hinzugefügt. Sollte es keines sein so wird die Action Funktion ausgeführt.

### **Funktion pickUpItem()**

Die Funktion kümmert sich um das ordentliche hinzufügen eines Gegenstandes ins Inventar.

### **Funktion holdItem()**

Die holdItem Funktion ladet das aktive Gegenstand in den „\_itemHolder“ und zeigt damit das Objekt „in der Hand“ des Spielers an.

### **Funktion dropItem()**

Mithilfe der dropItem Funktion kann das Aktive Gegenstand „Fallengelassen“ werden und aus dem Inventar entfernt werden.

### **Funktion mouseScrollValue()**

Die Einzige Aufgabe der Funktion mouseScrollValue ist es den wert des Mausrads in einen True/False wert zu übersetzen und damit die Funktion inventorySlotSelection Auszuführen.

### **Funktion inventorySlotSelection()**

In der Funktion inventorySlotSelection wird mithilfe eines Übergebenen bool wertes entschieden ob der Aktive Inventar Slot einen weiter oder zurück auswählen soll.

### **Funktion storeItems()**

Die Funktion storeItems wird benutzt um ein Item in das Inventar zu speichern und ins Array hinzuzufügen.

### **Funktion unpackItem()**

Die Funktion unpackItem ist ein teil der dropItem Funktion und kümmert sich um das entfernen des Objekts aus dem Array und platzieren in der Welt.

### **Funktion setActiveInventorySlot()**

Mit der Funktion setActiveInventorySlot hat der Spieler die Möglichkeit mit den Zahlentasten direkt Gegenstände im Inventar auszuwählen.

## **12.2 Spielsteuerung**

### **12.2.1 Inventar Steuerung**

Um durch das Inventar zu steuern hat der Spieler zwei verschiedene Möglichkeiten. Einmal kann der Spieler mit dem Maus-Rad nach oben oder unten durch das Inventar scrollen oder mit den Zahlen tasten 1 bis 4. Somit kann jeder Spieler mit seiner beliebigen Steuerung spielen.

## 13 KI

### 13.1 Defienition KI

Künstliche Intelligenz, wird verstanden als die Eigenschaft, die ein Wesen befähigt, angemessen und vorausschauend in seiner Umgebung zu agieren dazu gehört die Fähigkeit, Situationen wahrzunehmen und darauf zu reagieren, Informationen aufzunehmen, zu verarbeiten und als Wissen zu speichern, Sprache zu verstehen und zu erzeugen, Probleme zu lösen und Ziele zu erreichen.

### 13.2 Anforderungen

Im GDD <sup>1</sup> werden Mitarbeiter defieniert welche als Nicht-Spieler-Charaktere defieniert werden. Das bedeutet das diese Charaktere nicht von einem Spieler sondern vom Spiel selbst gesteuert werden. Die Mitarbeiter müssen Dynamische Entscheidungen treffen. Folgende werden im GDD erwähnt:

- Die Mitarbeiter sollen sich selbständig bewegen.
- Die Mitarbeiter sollen selbständig aktive elektro sicherheits Verstöße erkennen und auf diese reagieren.
- Die Mitarbeiter sollen aufgaben erledigen welche als Basis für eine Routen bildung dienen.

Die Anforderungen für die selbständige Bewegung der Mitarbeiter wurden in einem Späteren Standup <sup>2</sup> verschärft:

- Die Mitarbeiter sollen sich selbständig bewegen.
- Während sie sich bewegen sollen sie statische Hindernisse wie Wände und Möbel ausweichen.
- Während die Mitarbeiter sich bewegen sollen sie zu dem Dynamischen Hindernisse wie andere Arbeiter oder Spieler Ausweichen.

Um diese Anforderungen zu erfüllen wurde entschieden das eine KI für die Mitarbeiter entwickelt werden soll. Die erste Methode um die KI zu entwickeln die in Erwähnung gezogen wurde sind motion Behaviors Basierend auf [4, Reynolds99].

### 13.3 Behaviours

#### 13.3.1 Defienition Behaviors

Der Begriff Behaviours referenziert hier improvisierte oder lebens ähnliche aktionen die die KI ausführen kann. Die KI verfügt zur zeit nur über motion Behaviors.

#### 13.3.2 Motion Behaviors

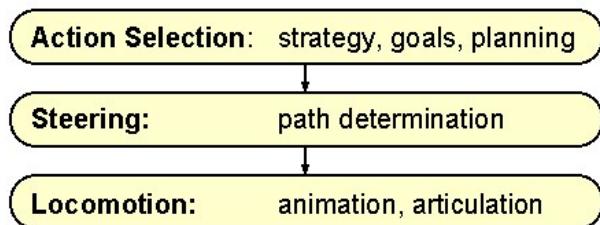


Abbildung 35: Motion Behaviour Ebenen (Deutsche Version benötigt)

Die oben gezeigte Hierarchie unterteilt das motion Behaviour in drei Ebenen. Diese Ebenen heißen Action Selection, Steering und Locomotion. Wichtig anzumerken ist das diese Struktur gut für motion Behaviors geeignet ist, aber es ist nicht zu empfehlen diese Struktur in anderen behaviors zu verwenden wie zum Beispiel Conversational Behaviors einer Chat KI welche eine dramatisch andere Struktur benötigen würde.

<sup>1</sup>Game Desing Dokument ist im Anhang enthalten

<sup>2</sup>Standup ist eine Meeting form in Scrumm im GDD sind mehr infos enthalten

### 13.3.3 Action Selection

In der Action Selection <sup>1</sup> wird die KI den Status der Welt überwachen und basierend auf den Status änderungen der Welt der Steering Ebene anweisungen geben welche von der Steering Ebene ausgeführt werden.

#### Beispiel:

In diesem Scenario verfügt der Arbeiter über eine Liste an Aufgaben die er Erledigen soll diese Aufgaben werden immer wieder gemacht und können von anderen Arbeitern belegt werden. Wenn die Action Selection des Arbeiters bemerkt das alle Aufgaben belegt sind wird diese der Steering Ebene befehlen umher zu wandern, falls die Action Selection dann merkt dass es eine freie Aufgabe existiert wird die Action Selection der Steering Ebene den Befehl geben zu dieser Aufgabe zu navigieren und diese zu Erledigen.

### 13.3.4 Steering

Die Steering <sup>2</sup> Ebene bricht die Anweisungen der Action Selection in mehrere Aktionen auf und trifft auf basis dieser Aktionen Entscheidungen die von der Locomotion ausgeführt werden.

#### Beispiel:

<sup>3</sup> Wenn die Steering Ebene von der Action Selection den Befehl eine Aufgabe zu Erledigen erhält, wird dieser Befehl in Mehreren Schritten Erledigt. Um sich zur Aufgabe zu Bewegen wird die Steering Ebene einen Pfad zur Aufgabe Anfordern da dieser nicht permanent von jeder Position bekannt ist <sup>4</sup>. Die Steering Ebene wird Dann der Locomotion befehlen sich entlang dieses Pfades zu Bewegen und weicht mit Hilfe von Steering Verhalten Objekten aus die der Pfad nicht umgeht hatt. Diese ausweich versuche werden auch von der Locomotion ebene Umgesetzt.

### 13.3.5 Locomotion

Die Locomotion Ebene ist für die eigentliche Umsetzung der Entscheidungen der Steering Ebene zuständig. Sie kann dafür Bewegung und Animationen verwenden um eine Umsetzung dem Spieler sichtbar zu machen.

### 13.3.6 Vorteile

Das Oben vorgestellte system bassiert auf dem motion Behavior System von [4, Reynolds99]. Dieses System stellt einen schnell zu implementierenden Kern mit emergenten Verhalten <sup>5</sup> aus simplen Komponenten.

### 13.3.7 Nachteile

Es gibt situationen in dem dieses Modell Fehler aufweist. Wenn die KI über Kollisionsvermeidung und eine konsistente Bewegung verfügen soll, wird die Größe der Steering Behavior Komponenten als konsequenz Unverhaltensmäßig Groß. Diese können dann auch zu eng mit eineander verknüpft sein was die Entitätsbewegung dann zerbrechlich und schwer modifizierbar macht.

#### Beispiel:

Wenn eine KI einem Pfad folgen soll und durch eine Zustandsänderung in der Welt den Pfad verlassen soll kann es passieren das die anderen Verhalten sich mit diesem Modell Streiten. Das kann dan wie folgt aussehen: Eine einzelne Steuerungsverhaltenskomponente wird gebeten, einen Vektor zurückzugeben, der ihre Entscheidung unter Berücksichtigung des aktuellen Zustands der Welt repräsentiert. Das Framework versucht dann, mehrere Entscheidungen zu vereinen. Es gibt jedoch einfach nicht genügend Informationen, um das Zusammenführen dieser Entscheidungen möglich zu machen. Durch Hinzufügen von Priorisierung oder Gewichtung wird versucht, das Zusammenführen durch Hinzufügen von mehr Informationen zum Ergebnis des Verhaltens einfacher zu machen, aber es führt zu lauterem Schreien anstatt einer nuancierteren Debatte. Das kann dazu führen das die KI eine Entscheidung trifft die keinen oder

<sup>1</sup> Action Selection ist Englisch und Bedeutet so viel wie Aktion Auswahl

<sup>2</sup> Steering ist Englisch und Bedeutet so viel wie Steuern

<sup>3</sup> Dieses Beispiel ist eine Vortersetzung des Beispiels 13.3.3

<sup>4</sup> Mehr zu der Pfadfindung kann im Kapitel gefunden werden

<sup>5</sup> Emergente Verhalten stehen für Verhalten die die eigentlichen Komponenten nicht besitzen.

kaum einen Effekt auf die KI hatt und stillsteht, im schlimsten Fall wird die KI nicht mehr ausweichen sondern über Längere zeit mit einem Hinderniss kollidieren dass im weg der Pfadabweichung ist.

#### Analyse:

Manchmal ist der Kontext, in dem die Entscheidung getroffen wurde, genauso wichtig wie die Entscheidung selbst. Dies ist ein besonderes Problem für Kollisionsvermeidungsverhalten, weil sie nur in der Sprache der gewünschten Geschwindigkeit und nicht in der unerwünschten Geschwindigkeit kommunizieren können.

#### 13.3.8 Lösung

Es ist klar geworden, dass die Rückgabe einer Entscheidung, auch wenn sie mit Metadaten versehen ist, nicht funktionieren wird. Stattdessen können wir versuchen, das Verhalten um Kontextinformationen für die Entscheidungsfindung zu bitten und die eigentliche Entscheidungsphase zu überspringen. Wenn wir dann alle diese Kontexte irgendwie zusammenführen könnten, könnte ein externer, verhaltensagnostischer Prozessor eine endgültige Entscheidung treffen, der sich vollständig aller Informationen bewusst ist.

Der Kontext für "avoid" könnte beispielsweise lauten: "Ich bin ziemlich sicher, dass wir nicht nach Süden gehen sollten", während der Kontext für "chase" lauten könnte: "Es ist etwas interessant, nach Westen und ziemlich interessant, nach Süden zu gehen". Es handelt sich hierbei um eine ganzheitliche Betrachtung und nicht um eine definitive Entscheidung. Das System kombiniert dann diese Kontexte auf magische Weise und offenbart, dass die ideale Entscheidung darin besteht, nach Westen zu gehen.

Das Ergebnis ist, als ob "chase" Hindernisse berücksichtigt und sein bevorzugtes Ziel ignoriert hätte, weil es blockiert war, obwohl die einzelnen Verhaltensweisen nur auf ihre eigenen Anliegen konzentriert waren. Das System ist emergent und bietet eine konsistente Kollisionsvermeidung sowie kleine, zustandslose Verhaltensweisen.

### 13.4 Context Steering

#### 13.5 Pfadfindung

##### 13.5.1 Anforderungen

Damit die KI sich so erfolgreich wie möglich über die Map bewegen kann, muss ein Pfadfindungssystem erstellt werden. Dieses System soll folgende Kriterien erfüllen:

- Die Pfadfindung muss einen Pfad zwischen zwei Punkten finden.
- Die Pfadfindung muss optimiert genug sein, um flüssig im Mehrspieler zu laufen.
- Es wird damit gerechnet, dass die Pfadberechnung bis zu 15 mal pro Frame abgerufen wird. Unter dieser Last darf das Spiel nicht unter eine festgelegte Aktualisierungszeit fallen.

Um dieses Ziel zu erreichen muss ein Algorithmus verwendet werden, und auf der Basis dieses Algorithmus wird dann ein Pfadsystem gebaut.

##### 13.5.2 Alternativen

Neben einem selbst entwickelten Pfadfindungssystem gibt es auch noch die Möglichkeit, ein externes Navigations-System zu verwenden. Der Nachteil von diesen externen Lösungen liegt aber darin, dass die meisten nicht genügend Einfluss auf die Pfadgenerierung erlauben, was zu Problemen führen kann. Aus diesem Grund wird eine selbst programmierte Lösung bevorzugt.

##### 13.5.3 A\* Algorithmus

Der A\* Algorithmus ist eine Erweiterung des Dijkstra's Algorithmus mit einigen charakteristiken des breadth-first search Algorithmus. Er erlaubt es, einen Pfad zwischen zwei Punkten in einem Raster zu berechnen ohne weiteres Wissen über den Gesamtzustand des Rasters. Wichtig anzumerken ist, dass der A\* Algorithmus einen heuristischen Teil hat, welcher später erwähnt wird. Das bedeutet, dass das Ergebnis des A\* Algorithmus in manchen Szenarien immer eine Abweichung des zur hundertprozent korrekten Lösung haben wird. Diese Abweichung ist aber nicht so groß, dass sie die KI in dramatischer

<sup>1</sup> heuristisch wird in dem Dokument erklärt.

Weise beeinflussen würde. Die Pfade bestehen aus verschiedenen Schritten, A\* berechnet den Wert eines Schrittes mit Hilfe der folgenden Formel:

$$f(n) = g(n) + h(n), \quad (1)$$

Die  $f(n)$  funktion steht hier für die gesammt kosten eines schrittes. Diese werden aus  $g(n)$  den Kosten zum Schritt und  $h(n)$  den Kosten bis zum ziel ausgerechnet.

**$g(n)$  erklären:**

$g(n)$  setzt sich aus den Kosten aller Schritte die bis zum Schritt begangen werden müssen zusammen.

**$h(n)$  erklären:**

$h(n)$  ist der heuristische Teil des A\* algorithmus. Dieser Wert soll idealer weise die exakten kosten des Pfades vom Schritte bis zum Ziel repräsentieren. Das ist aber nicht Möglich da die KI nicht über genug Informationen verfügt um schnell genug die Kosten jedes schrittes zu errechnen. Um trotzdem einen Wert zu erhalten muss die Funktion basierend auf den zur Verfügung stehenden Informationen eine Schätzung ausführen. Diese Schätzung ist die erwähnte heuristik. Sie ist zudem kirtisch in der Leistung des A\* Algorithmus. Um dieses Problem zu Lösen muss eine Methode gewählt werden, die manchmal den genauen Wert liefert, z.B. wenn ohne Hindernisse auf einer geraden Linie ein Pfad erstellt werden soll. Dies führt in einem solchen Fall zu einer perfekten Leistung von A\*.

**13.5.3.1 Vorteile:**

Ein Selbstgeschrieben A\* System bietet volle kontrolle über die Pfadgeneration.

**13.5.3.2 Nachteile:**

Die Entwicklungszeit des Spieles wird dramatisch verlängert.

**13.5.3.3 Probleme Während der Entwicklung:**

Das System Konnte nicht Schnell genug vertiggestellt werden. Aus diesem Grund wurde es Entschieden die schon von Unity zur verfüzung Gestellten Systeme zu verwenden um den Entwicklungs Prozzess zu Beschleunigen

**13.5.4 Unity Navigation System**

Das Unity Navigation System ist ein Toolset, das Entwicklern hilft, künstliche Intelligenz-gesteuerte Agenten in einer 3D-Umgebung zu navigieren. Das System bietet eine Reihe von Funktionen und Komponenten, die bei der Erstellung von Navigationssystemen helfen, einschließlich einer Navigation Mesh, die die begehbareren Bereiche der Szene definiert, und einem NavMesh Agent, der ein Objekt in der Szene darstellt, das navigiert werden soll. Das Navigationssystem bietet auch ein Pathfinding-System, das es den Agenten ermöglicht, den kürzesten Weg zwischen zwei Punkten auf der Navigation Mesh zu finden. Dieses System ist anpassbar und kann für verschiedene Arten von Szenen und Agenten konfiguriert werden. Darüber hinaus bietet das Navigationssystem Funktionen für die Vermeidung von Hindernissen, das Ausweichen von anderen Agenten und die Verfolgung von Zielen. Entwickler können auch benutzerdefinierte Skripte erstellen, um die Funktionalität des Navigationssystems zu erweitern und anzupassen.

**13.5.4.1 Agent:**

test

**13.5.4.2 Navmesh und Pfadfindung:**

test

**13.5.4.3 Vorteile:**

Das Unity Navigation System bietet viele Vorteile, wie die einfache Bedienbarkeit, die Anpassbarkeit und das Pathfinding-System. Es bietet auch Funktionen zur Vermeidung von Hindernissen und zum Ausweichen von anderen Agenten sowie die Möglichkeit, die Funktionalität des Systems durch benutzerdefinierte Skripte zu erweitern.

#### **13.5.4.4 Nachteile:**

Allerdings kann das Navigationssystem in großen Szenen mit vielen Agenten und Hindernissen zu Leistungsproblemen führen und das Erstellen einer Navigation Mesh kann bei komplexen Szenen zeitaufwendig und schwierig sein. Außerdem bietet das System möglicherweise nicht alle Funktionen, die für spezifische Szenarien erforderlich sind.

## 14 Mapdesign

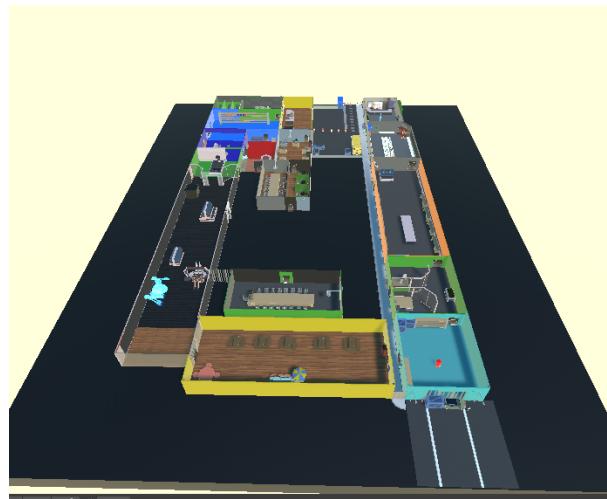


Abbildung 36: image

Das Mapdesign basiert grundsätzlich auf einer Herstellungs-Fabrik, oder auf Deutsch; eine Zusammenbau Fabrik mit unterschiedlichen Bereichen; von einer Rezeption bis zu einem Klassenraum und dazu wollten wir das Map-Layout simpel halten und kein Labyrinth daraus machen. Unter uns gab es keine große Diskussion bezüglich des Layouts, wir haben uns letztendlich auf das obige Layout festgelegt. Das Layout der Map sollte, wie im obigen Absatz schon erwähnt, einfach gehalten werden. Die ersten Konzept Layouts bestanden aus einem Gang und 3 Räumen welches ungefähr so aussah:

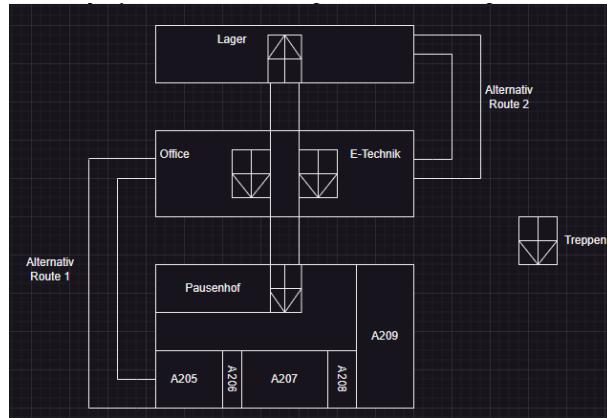


Abbildung 37: image (2)

Das erste Design hatte einige nicht ausbalancierte Aspekte, die Ideen für die 3 Areale waren nicht passend und wir habe versucht das Beste daraus zu machen und habe das fertige Konzept im Meeting vorgeschlagen. Nachdem wir den „Gameplay Flow“ der Map angeschaut haben, haben wir etwas Brainstorming gemacht und sind auf diese Fabrik Grundrisse gestoßen:

Wir haben uns diese angeschaut, und es wurde beschlossen, dass wir mit dem v3.2 Layout arbeiten werden, da es nicht zu klein und nicht zu groß ist, es ist nicht zusammengequetscht und es hat viele Möglichkeiten um die Räume zu füllen. Das Layout, da das Bild recht klein war, haben wir als erstes in Draw.io nachgezeichnet und mit dem Aufbau der Map angefangen.

Für den Aufbau der Map wurde freigestellt ob kostenlose Unity Assets verwendet werden können oder nicht, Hauptsache die anderen Gruppenmitglieder wurden darüber informiert falls kostenlose Unity Assets importiert wurden. Da auch der ausgewählte Grundriss wenige Komplikationen hatte, wurde das ausgewählte Layout ein wenig angepasst und sah letztendlich so aus:

Teile der Map wo ein KI Mitarbeiter oder wo der Spieler selbst von der Map fallen kann, wurden entsprechend blockiert mit zum Beispiel einem Zaun oder einem Straßenschild das Selbe wurde für mög-

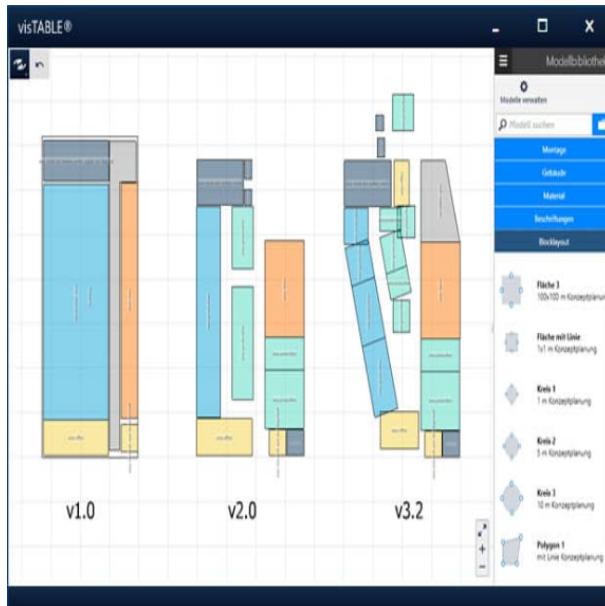


Abbildung 38: image (3)



Abbildung 39: image (4)

liche Out of Bounds (außerhalb des Spielbereiches) Szenarien getan und die Räume sind entsprechend eingerichtet mit Objekten, dass die KI Mitarbeiter und der Spieler nicht feststecken oder aus der Map raus fallen.

#### 14.1 Türen

Normalerweise ist keine Map vollständig ohne Türen, darum wurde für dieses Projekt ein Türen Skript programmiert. Die grobe Funktion ist, dass die Tür sich in einer definierten Richtung öffnet sobald ein Spieler oder NPC in der Nähe des Sphären Colliders ist. Die Codierung des Skripts besteht aus 6 SerializedFields, welche als Private deklariert sind. Jedes SerializedField hat eine unterschiedliche Funktion

im Endprodukt. `_openSpeedCurve` ist eine einstellbare Regelkurve für die Öffnungsgeschwindigkeit der Tür, die `_OpenDirection` ist die Richtung wo sich die Tür öffnen soll (x,y,z). Die `_openDistance` ist die Distanz wie weit die Tür in die angegebene Öffnungsrichtung滑den soll, der `_openSpeedMultiplier` ist der Multiplikator der AnimationCurve bzw. `_openSpeedCurve` wie schnell die Tür sich öffnen soll. Als letztes müssen wir der Tür auch ein Objekt zuweisen oder auch „DoorBody“ genannt. Dies kann ein Skalierter Würfel sein oder ähnliches. Zudem können die Türen von den NPCs geöffnet werden und der DoorBody braucht einen Sphären Collider als Radius Indikator und einen Trigger damit die Tür sich von Spielern und den NPCs öffnen lässt. Ich habe mich für eine seitlich öffnende Tür entschieden, da „normale“ Türen nicht sehr starke Computer Systeme fast zum abstürzen bringen. Unten sind zwei Beispielbilder der Türen, einmal wie es im Editor aussieht und wie es aussieht, wenn ein Spieler in der Nähe der Tür Stehen würde:

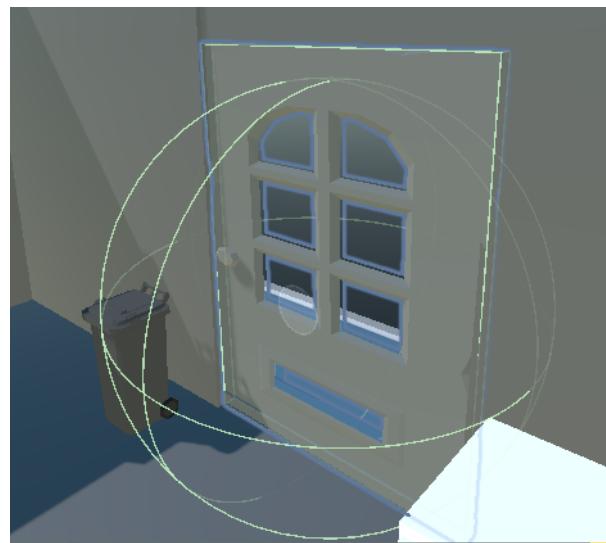


Abbildung 40: image (6)

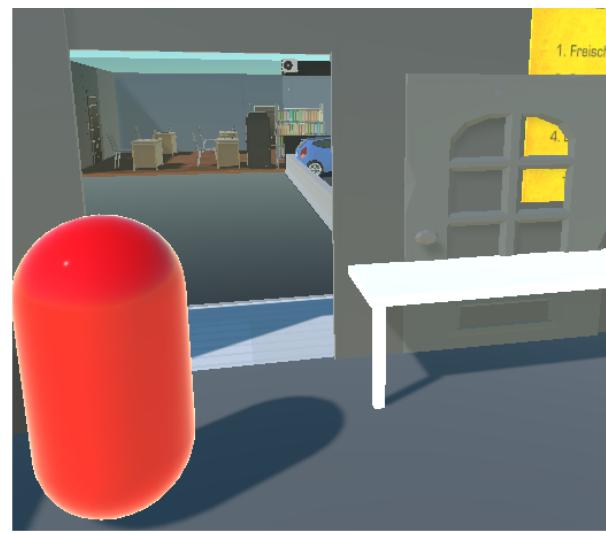


Abbildung 41: image (7)

## 15 Unity Assets

Wie mehrere Male erwähnt: für die Gestaltung des Spiel Environments wurden „Unity Assets“ benutzt. Kurz gefasst: Unity Assets sind von der Unity Community erstellte Materialien, Szenen, Prefabs, Scripts und sogar Tools. Einige Unity Assets sind kostenpflichtig von Kostenlos bis 1500\$, wir haben uns aber dagegen entschieden Assets zu kaufen, weil die kostenlosen Assets schon genug waren. Zudem gibt es relativ viele kostenlose Assets im Asset Store; 3067. (Stand 5/5/2023) Die meistbenutzten Assets in diesem Projekt sind:

Die School Assets für die Modellierung des Fachpraxis Raumes und des Klassenraumes.



Abbildung 42: school

Das CITY Package für die Straßen, Mauern, Zäune und Straßenschilder.



Abbildung 43: city

Das LowPoly Server Room Props Pack für die Monitore, Server und PCs in den Büros



Abbildung 44: 01c02c4a-59b6-4c31-99e2-9e4127c35801

Das POLYGON Starter Pack - Low Poly 3D Art by Synty für Türen, Fenster, Wände und PKWs.  
Das Apartment Kit für die Gestaltung der Büros, Human Resources Abteilung und der Rezeption.  
Und das letzte benutzte Asset ist das Sci-Fi Styled Modular Pack für Kleinigkeiten wie zum Beispiel: Trennwände, Fenster, Texturen und Modelle.



Abbildung 45: lowpoly



Abbildung 46: apartment



Abbildung 47: scifi

## Literatur

- [1] Robin Hunicke, Marc LeBlanc, Robert Zubek, *A Formal Approach to Game Design and Game Research*, <https://users.cs.northwestern.edu/~hunicke/MDA.pdf>
- [2] Penny de Bill, *A cours on AI in Unity* <https://learn.unity.com/tutorial/complex-behaviours-1?uv=2019.4&courseId=5dd851beedbc2a1bf7b72bed&projectId=5e0b9220edbc2a14eb8c9356#>
- [3] Andrew Fray, *Behavior-Driven Steering at the Macro Scale* [https://www.gameaiopro.com/GameAIPro2/GameAIPro2\\_Chapter18\\_Context\\_Steering\\_Behavior-Driven\\_Steering\\_at\\_the\\_Macro\\_Scale.pdf](https://www.gameaiopro.com/GameAIPro2/GameAIPro2_Chapter18_Context_Steering_Behavior-Driven_Steering_at_the_Macro_Scale.pdf)
- [4] Craig W. Reynolds *Steering Behaviors For Autonomous Characters* <https://www.red3d.com/cwr/steer/gdc99/>

## Abbildungsverzeichnis

1	ER-Diagramm - User . . . . .	6
2	ER-Diagramm - Game . . . . .	8
3	API Klassen-UML . . . . .	9
4	Figma Homepage . . . . .	13
5	Figma Login . . . . .	14
6	Figma Profil . . . . .	14
7	Figma Score . . . . .	15
8	Result Homepage . . . . .	16
9	Result Login . . . . .	16
10	Result Profil . . . . .	17
11	Result Score . . . . .	17
12	Result Changelog User . . . . .	18
13	Result Changelog Admin . . . . .	18
14	Klasse DBconnection . . . . .	19
15	Klasse login . . . . .	19
16	Klasse registration . . . . .	20
17	Klasse ranking . . . . .	21
18	Klasse profile . . . . .	22
19	Klasse changeData . . . . .	23
20	Dedicated Game Server (DGS) . . . . .	28
21	Client Hosted (Listen Server) . . . . .	29
22	Relay Server . . . . .	29
23	UML RelayManager . . . . .	30
24	Programm Ablauf Diagramm setupRelay . . . . .	31
25	Programm Ablauf Diagramm joinRelay . . . . .	31
26	Server RPC Grafik . . . . .	32
27	Client zu Host . . . . .	33
28	Server-RPC   Horst zu Horst . . . . .	33
29	Client RPC . . . . .	34
30	Client RPC von Horst . . . . .	34
31	Client RPC zu Bestimmten Client . . . . .	35
32	Klasse apiManager UML . . . . .	36
33	User Struct UML . . . . .	37
34	I_interactable UML . . . . .	39
35	Motion Behaviour Ebenen (Deutsche Version benötigt) . . . . .	41
36	image . . . . .	46
37	image (2) . . . . .	46
38	image (3) . . . . .	47
39	image (4) . . . . .	47
40	image (6) . . . . .	48
41	image (7) . . . . .	48
42	school . . . . .	49
43	city . . . . .	49
44	01c02c4a-59b6-4c31-99e2-9e4127c35801 . . . . .	49
45	lowpoly . . . . .	50
46	apartment . . . . .	50
47	scifi . . . . .	50