IPA 2022

"Endless Runner" Spiel für Android

Artem Kägi
Supercomputing Systems AG
Technoparkstrasse 1
CH-8005 Zürich

Inhaltsverzeichnis

Inhaltsverzeichnis

1 Teil 1

1.1 Vorwort

Diese Dokumentation gehört zur IPA von Artem Kägi, welcher bei der Firma Supercomputing System AG angestellt ist. In diesem Dokument ist die Arbeit und das Vorgehen dokumentiert.

Die Dokumentation ist in 2 Teile gegliedert

Im ersten Teil werden die detaillierte Aufgabenstellung und der Ablauf der Arbeit aufgezeigt. Hier werden die vorhandenen Vorkenntnisse und die benutzten Mittel aufgezählt

Im zweiten Teil wird das Projekt auch wirklich dokumentiert und beschrieben, es wird auch erwähnt welche Probleme aufgetreten sind.

Der Ganze Sourcecode und Assets werden in einem Git repository hochgeladen.

https://github.com/ArtemKaegi/IPA-2022-Endless-Runner

Die Dokumente der IPA werden in eigenen Ordnern täglich abgelegt.

Durch die ganze Arbeit hinweg wurde die Projektplanungsmethode IPERKA eingesetzt.

1.2 Aufgabenstellung

Das Ziel der IPA ist ein Android Spiel vom Typ "Endless Runner". Das Grundkonzept dieser Spiele ist folgendes: Der Avatar bewegt sich automatisch durch eine Spielwelt. Der Spieler steuert seinen Avatar und versucht den darin vorkommenden Hindernissen so lange wie möglich auszuweichen. Je länger der Spieler dies schafft, umso mehr Punkte werden gesammelt. Dazu stehen dem Spieler diverse Bewegungen wie z.B. springen, ducken, nach Links bzw. Rechts ausweichen etc. zur Verfügung. Sobald der Spieler mit einem Hindernis kollidiert, ist das Spiel zu Ende. Das Ziel des Spieles ist es also, so lange wie möglich zu überleben, d.h. Kollision zu vermeiden.

Die Mobile-Applikation wird mit einem zentralen Backend kommunizieren. Das Backend verwaltet die Spieler und deren erreichte Punkte. Damit können die Spieler ihre erreichten Highscores miteinander vergleichen.

Der Spieler kann während seinem Run eine Währung, sogenannte Coins, sammeln. Mit diesen kann er "Skins" kaufen. Diese Skins erlauben es dem Spieler das Aussehen seines Avatars im Spiel zu verändern.

Für die Assets im Spiel (z.B. Avatar, Skins, Objekte in den Levels) können bestehende Assets oder selbst erstellte Assets verwendet werden.

Die Mobile-Applikation schickt die erreichten Highscores an ein zentrales Backend, wo diese mit dem Spielerpseudonym persistiert werden. In der Mobile-Applikation sind neben den Highscores auch die freigeschalteten Skins einsehbar.

Voraussichtlich verwendete Technologien: C# (Unity) für Android Mobile Spiel, Python REST Server für Backend und MySQL Datenbank im Backend.

1.2.1 App

Menusystem: Die App hat ein Menu mit den Inhalten "Spiel Starten", "Highscore" und "Skins".

- Spiel Starten: startet einen neuen Run.
- Highscore: zeigt die Top 10 Highscores über alle Spieler an. Der eigene Highscore wird separat angezeigt. Wie diese Highscores dargestellt werden, wird während der IPA evaluiert.
- Skins: Alle im Spiel vorhandenen Skins sind hier dargestellt. Bei jedem Skin ist ersichtlich, ob der Spieler ihn bereits freigeschaltet hat oder nicht. Skins können in diesem Menu mit den gesammelten Coins freigeschaltet und aktiviert werden.

1.2.2 Gameplay

Die Spielwelt wird während dem Spiel zufällig generiert.

Der Avatar bewegt sich automatisch durch die Spielwelt. Der Run ist zu Ende, sobald der Avatar mit einem Hindernis kollidiert. Der Spieler hat folgende Aktionen zur Auswahl:

- Springen: Der Avatar weicht einem Hindernis aus, in dem er darüber springt.
- Sliden: Der Avatar weicht einem Hindernis aus, in dem er sich darunter durch bückt.
- Links: Der Avatar weicht einem Hindernis aus, in dem er nach links ausweicht.
- Rechts: Der Avatar weicht einem Hindernis aus, in dem er nach rechts ausweicht.
- Coins sammeln: kollidiert der Avatar mit einem Coin so wird dieser aufgenommen.
- Sterben: kollidiert der Avatar mit einem Hindernis ist der Run zu Ende. Der Spieler wird aufgefordert seinen Namen einzugeben. Wenn der Spieler will, kann der erreichte Highscore öffentlich gemacht werden.
- Punkte: der Spieler erhält Punkte je länger der Run dauert. Pro Frame, das der Avatar rennt, erhält der Spieler einen Punkt.

1.2.3 Backend

Das Backend bietet eine REST Schnittstelle mit den folgenden Operationen

- Highscore erfassen und abrufen
- Coins erfassen und abrufen
- Skins freischalten und abrufen

Dieses REST API genau zu spezifizieren ist Teil der IPA.

Das Backend wird mit Python umgesetzt. Es verbindet mit einer MySQL Datenbank.

Die Datenbank speichert alle notwendigen Daten, um die Informationen in den Menüpunkten Highscore und Skins darstellen zu können.

1.2.4 Nicht-Funktionale Anforderungen

Das Backend enthält ein Logging welches zeigt, wann welche Requests an das Backend geschickt wurden.

Das Backend läuft zentral auf einem Laptop im gleichen WLAN wie das Mobiltelefon (d.h. es ist kein Hosting via Internet notwendig).

Das System verfügt über ein Error-Handling. Die Mobile App soll auch dann funktionieren, wenn das Backend nicht erreichbar ist oder sich fehlerhaft verhält.

Die individuellen Beurteilungskriterien findet man im PkOrg.

1.3 Projektaufbauorganisation

Lehrbetrieb und Durchführungsort:
Supercomputing Systems AG
Technoparkstrasse 1
8005 Zürich
043 456 16 00
Info@scs.ch

Kandidat:
Artem Kägi
Agnesstrasse 8
078 705 64 27
artem.kaegi@scs.ch

Berufsbildner:
Christian Walder
Benedict Schule
Militärstrasse 106
8004 Zürich
044 298 17 83
christian.walder@benedict.ch

Verantwortliche Fachkraft:
Pascal Iselin
Supercomputing Systems AG
Technoparkstrasse 1
8005 Zürich
079 215 62 37
pascal.iselin@scs.ch

Nebenexpert:
Rinoy Manavalan
rinoy.manavalan@gmail.com
078 226 80 19

Hauptexpert:
Marko Tepeluk
martepeluk@gmail.com
076 497 58 88

1.4 Vorkenntnisse

Kandidat hat bei der SCS schon Handy Apps entwickelt und zu einer auch ein Python Backend erstellt. Er hat Interesse für Spiele Entwicklung und hat in seiner Freizeit schon mit Unity Spiele konstruiert. Für ein weiteres Projekt hat er auch eine MySql Datenbank mit einem Python Rest interface verbunden.

- Unity: Unregelmässig mit Unity, PC-Spiele erstellt. Noch nie ein Handygame entwickelt.
- Python: Rest Schnittstelle für Flutter App schon einmal erstellt.
- MySql: Datenbank für ein Python Backend erstellt.

1.5 Vorarbeit

Für die IPA wurden folgende Massnahmen vorgenommen:

- Unity wurde auf dem Arbeitslaptop installiert
- Es wurde ein «Testspiel» auf ein Android Handy gebracht.

1.5.1 In den letzten 6 Monaten

Entwicklung von Handy Apps und Python Applikationen:

- OFS: Eine App mit einem Python Rest backend
- .Net Applikation für SBB intern.

Werkzeuge:

- JetBrains: Rider, Intellij, PyCharm
- MySql Workbench

1.6 Arbeitsprotokoll

Starttermin: 28.02.2022

Abgabedatum: 15.03.2022

1.6.1 Mo 28.02.2022

Ausgeführte Arbeiten	• Zeitplan erstellt • Aufbau der Dokumentation definiert • Leitfragen und Aufgabenstellung studiert
Ungeplante Arbeiten	
Erreichte Ziele	• Der Zeitplan steht • Aufbau der Dokumentation ist klar
Beanspruchte Hilfestellungen	
Aufgetretene Probleme	
Reflexion	Ich konnte gut in meine IPA starten, jedoch habe ich ein paar fragen und Unsicherheiten. Die fragen kann mir hoffentlich mein Hauptexperte morgen beim besuch beantworten. Meine grösste Unsicherheit bezieht sich auf die Dokumentation, ich bin mir unsicher, wie ich dieses Umsetzen soll. Ich habe es jetzt nach Leitfaden gemacht, jedoch endet der Streng definierte teil hier und ich muss mir was ausdenken. Ich hoffe, dass auch das beim Gespräch mit dem Hauptexperten morgen etwas klarer wird.

1.6.2 Di 01.03.2022

Ausgeführte Arbeiten	• Realisierungskonzept erstellen • Expertenbesuch durchführen • Datenmodell erstellen
Ungeplante Arbeiten	
Erreichte Ziele	• Ein Realisierungskonzept steht • Expertenbesuch ist gut gelaufen • Ein Datenmodell für die Datenbank ist erstellt
Beanspruchte Hilfestellungen	• Berufsbildner Pascal Iselin hat seine Meinung zu meinem Dokumentationsaufbau geäussert und erwähnt, wie dies andere Leute machen.
Aufgetretene Probleme	
Reflexion	Es läuft malwieder gut, das Erstellen des Realisierungskonzepts ging länger als gedacht. Dafür ging das Erstellen des Datenmodell schneller als gedacht. Ich bin mir jetzt im Klaren, wie genau ich die Dokumentation strukturieren will. Morgen werde ich mit der ersten praktischen Aufgabe anfangen.

1.6.3 Mi 02.03.2022

Ausgeführte Arbeiten	• Git für das Projekt erstellen. • Bewegung realisiert. • GameManager entwickeln. • MenuSystem einrichten.
Ungeplante Arbeiten	
Erreichte Ziele	• Git für das Projekt ist da • Bewegung ist realisiert • GameManager Steht.
Beanspruchte Hilfestellungen	
Aufgetretene Probleme	• Musste ein Teil des Menu Systems schon umsetzten, um diese an den GameManager zu testen
Reflexion	Es läuft gut, ich konnte die Praktische Arbeit heute anfangen. Die Bewegung hat mir ein paar Mal Kopfschmerzen bereitet, da ich mir nicht sicher war, wie ich das Springen umsetzen wollte, jedoch denke ich, ich habe eine gute Lösung gefunden.

1.6.4 Do 03.03.2022

Ausgeführte Arbeiten	• MenuSystem einrichten • Sterben
Ungeplante Arbeiten	• Versucht die Inputaufnahmen zu verbessern.
Erreichte Ziele	• MenuSystem eingerichtet • Der Spieler kann jetzt sterben
Beanspruchte Hilfestellungen	• Ein Tutorial angesehen zu Menu Systemen von Brackeys: https://www.youtube.com/watch?v=zc8ac_qUXQY&ab_channel=Brackeys
Aufgetretene Probleme	• Die Bewegung des Spielers wird nicht richtig aufgenommen, vermutlich wegen der Inputaufnahme.
Reflexion	Es läuft immer noch gut, ich konnte das Menu System mit dem GameManger integrieren und das Sterben des Spielers entwickeln. Jetzt wird das gesamte Spiel «Neugestartet» ohne dass die Szene neugestartet werden muss. Die eingaben vom Spieler werden scheinbar nicht immer aufgenommen, das sollte ich mir ansehen, denn es ruiniert die gesamte Spielerfahrung.

1.6.5 Mo 07.03.2022

Ausgeführte Arbeiten	• Skin System • Coins und Punktesystem entwickeln
Ungeplante Arbeiten	• Inputaufnahme verbessert
Erreichte Ziele	• Skin System steht • Coins und Punktesystem entwickelt.
Beanspruchte Hilfestellungen	
Aufgetretene Probleme	• Die Bewegung des Spieler wird nicht richtig aufgenommen, vermutlich wegen der Inputaufnahme.
Reflexion	Es läuft sehr gut. Ich konnte heute das Skin und Coin/Punkte System einrichten. Es funktioniert alles einwandfrei. Dazu konnte ich das Problem für die Inputaufnahmen feststellen und beseitigen. Es lag daran das der Input im FixedUpdate genommen wurde welcher viel zu selten geprüft hat, jetzt ist es in Update.

Di 08.03.2022

Ausgeführte Arbeiten	• Datenbank einrichten • Backend Schnittstelle aufsetzten
Ungeplante Arbeiten	
Erreichte Ziele	• Datenbank eingerichtet
Beanspruchte Hilfestellungen	• Pascal um Hilfe beim SQL-Problem gefragt, dieser hat als Vorschlag gebracht das die Query commit werden sollte, dies war auch richtig.
Aufgetretene Probleme	• Python wollte keine Änderungen an der Datenbank machen.
Reflexion	Es läuft ok. Die Schnittstelle hat viel mehr zeit verbraucht als erwartet, zum Glück habe ich ein bisschen Puffer übriggelassen, also hat es mich nicht zu stark aus dem Konzept gebracht. Die Datenbank war einfach aufzusetzen und ich sollte morgen auch einfach die Verbindung zwischen der App und dem Backend einrichten können.

1.6.6 Mi 09.03.2022

Ausgeführte Arbeiten	• Backend Schnittstelle aufsetzten • Backend zu Frontend Anbindung • Unittest Backend
Ungeplante Arbeiten	
Erreichte Ziele	• Backend Schnittstelle aufgesetzt • Backend zu Frontend Anbindung steht
Beanspruchte Hilfestellungen	• Pascal darum gebeten meinen bisherigen IPA-Bericht durchzulesen und mir eine Rückmeldung zu geben
Aufgetretene Probleme	
Reflexion	Es läuft wieder mal gut, Ich habe mich dazu entschieden die Backend Anbindung zuerst zu machen, weil das wichtiger ist für dieses Projekt als die Unittests. Dennoch konnte ich heute die Unittests anfangen, bin mir aber nicht sicher, ob ich diese zum Laufen bringe.

1.6.7 Do 10.03.2022

Ausgeführte Arbeiten	• Assets Kreiert • Unittests Backend
Ungeplante Arbeiten	
Erreichte Ziele	• Ein paar Assets Kreiert
Beanspruchte Hilfestellungen	• Pascal hat mir eine Rückmeldung zu meinem Bericht gegeben. Mir fehlen noch ein paar Grafiken, das Realisierungskonzept ist nicht detailliert genug und bestimmte Kapitel müssen erweitert werden
Aufgetretene Probleme	
Reflexion	Es läuft wieder nur Ok. Die Unittest laufen nicht und ich bin mir nicht sicher, wie ich diese Umsetzten kann. Ich habe wenig zeit und verlege diese auf später. Dafür konnte ich sehr gut mit den Assets vorankommen. Ich war schnell genug, um diese bei dem Expertenbesuch vorzustellen.

1.6.8 Mo 14.03.2022

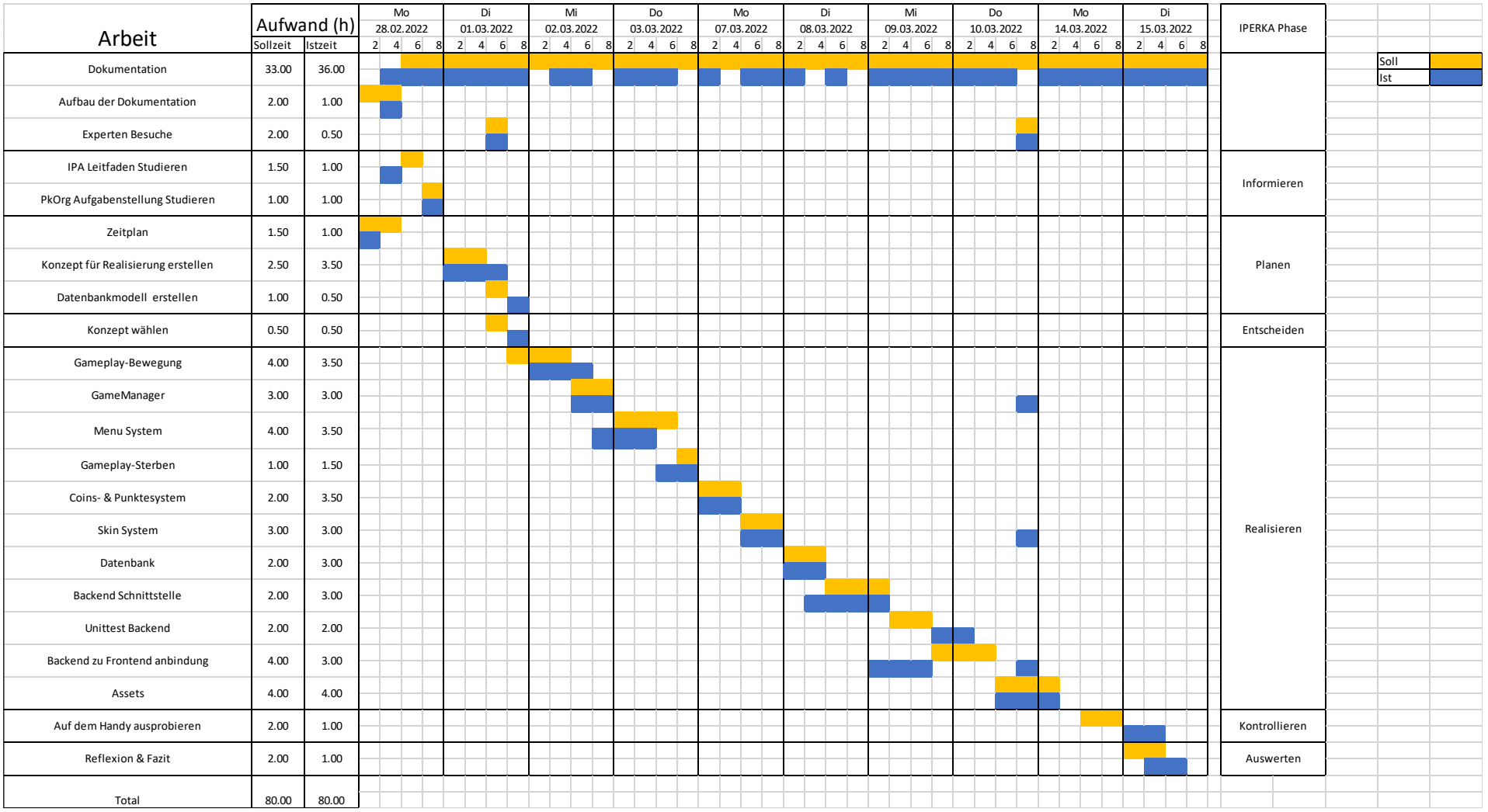
Ausgeführte Arbeiten	• Doku weiterschreiben • Anpassungen Assets
Ungeplante Arbeiten	
Erreichte Ziele	• Doku weitergeschrieben • Assets angepasst
Beanspruchte Hilfestellungen	•
Aufgetretene Probleme	
Reflexion	Es läuft wieder nur Ok. Ich bin jetzt langsam unter Zeitdruck, mir ist aufgefallen das ich die Skins nicht an das Backend sende und ich vermute das ich dafür auch keine Zeit mehr haben werde. Ich habe heute vor allem an der Doku weitergeschrieben. Diese habe ich unterschätzt und muss dies jetzt nachholen.

1.6.9 Di 14.03.2022

Ausgeführte Arbeiten	• Doku Fertigschreiben • Kriterienkatalog anschauen
Ungeplante Arbeiten	
Erreichte Ziele	• Doku Fertiggestellt • Kriterienkatalog angeschaut
Beanspruchte Hilfestellungen	
Aufgetretene Probleme	
Reflexion	Der letzte Tag verlief reibungslos, ich konnte mich gestern komplett auf die Doku konzentrieren und viel nachholen. Heute konnte ich noch die Doku fertigschreiben und mit dem Kriterienkatalog abgleichen. Ich bin sehr zufrieden mit dem Spiel und der geleisteten Arbeit.

1.7 Zeitplan

Arbeit	Aufwand (h) Sollzeit	Aufwand (h) Istzeit	IPERKA Phase
Dokumentation	33.00	36.00	
Aufbau der Dokumentation	2.00	1.00	
Experten Besuche	2.00	0.50	
IPA Leitfaden Studieren	1.50	1.00	Informieren
PkOrg Aufgabenstellung Studieren	1.00	1.00	
Zeitplan	1.50	1.00	Planen
Konzept für Realisierung erstellen	2.50	3.50	
Datenbankmodell erstellen	1.00	0.50	
Konzept wählen	0.50	0.50	Entscheiden
Gameplay-Bewegung	4.00	3.50	Realisieren
GameManager	3.00	3.00	
Menu System	4.00	3.50	
Gameplay-Sterben	1.00	1.50	
Coins- & Punktesystem	2.00	3.50	
Skin System	3.00	3.00	
Datenbank	2.00	3.00	
Backend Schnittstelle	2.00	3.00	
Unittest Backend	2.00	2.00	
Backend zu Frontend anbindung	4.00	3.00	
Assets	4.00	4.00	
Auf dem Handy ausprobieren	2.00	1.00	Kontrollieren
Reflexion & Fazit	2.00	1.00	Auswerten
Total	80.00	80.00	



2 Projekt

2.1 Zusammenfassung

Diese Kurzfassung richtet sich an die fachlich kompetenten Leser mit Fachwissen in der Informatik. Sie soll den Einstieg für das Verständnis dieser Arbeit erleichtern und einen ersten Überblick liefern.

2.1.1 Ausgangslage

Es soll ein Android Spiel entwickelt werden. In diesem hat der Spieler eine Spielfigur, welche er kontrolliert und mit welcher er versuchen muss Hindernissen auszuweichen während sich die Figur von selben immer weiter nach vorne bewegt. Während des Spieles kriegt der Spieler Punkte basierend darauf, wie lange dieser überlebt hat. Das alles passiert in einer Spielewelt, welche im verlauf des Spiels generiert wird. Neben den Hindernissen kann der Spieler auch «Coins» in der Spielewelt finden, mit diesen kann er später Skins kaufen. Wenn der Spieler mit einem der Hindernisse kollidiert, stirbt dieser und das Spiel ist zu ende. Das Ziel ist es so lange wie möglich zu überleben.

Nach dem Tod des Spielers wird sein Highscore an die REST Schnittstelle geschickt, um diesen in der Datenbank zu speichern. Die gekauften Skins werden ebenfalls in der Datenbank festgehalten. In der App kann seinen Highscore und den Highscore von anderen einsehen.

2.1.2 Umsetzung

Das Projekt wird in 3 Teile gegliedert. Frontend mit dem Handy Spiel, die Schnittstelle und die Datenbank.

Das Android Spiel wird mit der Game Engine Unity Entwickelt, wobei C# als Sprache benutzt wird. Für die Codebearbeitung wird JetBrains Rider benutzt, da dieser Unity Plugins besitzt. 3D Assets für das Spiel werden in Blender erstellt.

Die Schnittstelle wird mit dem Flask Framework für Python erstellt. Zum Programmieren und Ausführen wird PyCharm benutzt mit der neuesten Python version.

Die Datenbank wird mit SQL auf die beine gestellt. Für die Datenbank und das Modell wird MySql Workbench benutzt.

2.1.3 Ergebnis

Das Endprodukt besteht aus 3 Teilen. Die Unity App, die Flask Schnittstelle in Python und die MySql Datenbank.

Die Unity App ist ein einfaches SubwaySurfers-artiges Spiel, in dem man Hindernissen ausweichen muss, während man immer schneller wird. Das Spiel versendet SpielDaten wie SpielerNamen, Highscore und Gekaufte Skins an das Backend.

Die Flask Schnittstelle läuft auf einem Laptop im gleichen Netzwerk wie das Spiel. Es konvertiert entweder die Daten, die es von der App kriegt zu Queries welche es dann an die Datenbank versendet oder es führt eigene Queries aus um dann die Informationen aus der Datenbank an die App zu senden.

Die Datenbank besteht aus 3 Tabellen (players, highscores, skins). Hier werden alle Queries die von der Flask Schnittstelle kommen, ausgeführt.

2.2 Informieren

Der erste Teil der IPERKA-Methode ist Informieren. In dieser Phase wird der Auftrag untersucht und die Erkenntnisse ausgewertet. Die wesentlichen Punkte werden so früh wie möglich erkannt und allfällige Fragen geklärt.

2.2.1 Ziele

Ein Android Spiel mit einer kontinuierlich generierten Welt.

Der Spieler bewegt sich durch die Welt und weicht Hindernissen aus.

Der Spieler kann Springen, Sliden, nach Links und nach rechts ausweichen.

Der Spieler stirbt, wenn er mit einem Hindernis kollidiert.

Der Spieler sammelt einen Coin, wenn er mit einem Coin kollidiert.

Der Spieler sammelt Punkte basierend darauf, wie lange er überlebt hat.

Das Spiel hat 3 Seiten. «Spiel Starten», «Highscore» und «Skins» zwischen dennen man mit einem Menu navigieren kann.

Der Spieler kann Skins mit Coins kaufen.

Die Punkte und Skins werden an ein REST Backend geschickt und in einer Datenbank gespeichert.

Das Backend verfügt über einen Log, worin alle Events und Errors an der REST Schnittstelle sowie Zugriffe auf die Datenbank geloggt werden. Das dazugehörige Logfile soll im Filesystem der Maschine, auf der das Backend und die DB laufen, abgelegt werden.

Die Highscores von anderen Spielern sind im Spiel einsehbar.

2.2.2 Vorgaben

App:

- Läuft auf einem Android Handy.
- Greift auf Backend im gleichen Netz zu.
- Verfügt über Error-Handling und funktioniert ohne Backend.

Backend:

- Benutzt Python für eine REST Schnittstelle.
- Datenbank speichert alle notwendigen Daten, um die Informationen in den Menüpunkten Highscore und Skins darstellen zu können.
- Enthält ein Logging.
- Läuft zentral auf einem Laptop.

2.3 Planen

Das Planen ist die zweite Phase der IPERKA-Methode. In dieser Phase wird der Zeitplan fertiggestellt und das Testkonzept sowie das Realisierungskonzept erstellt. Das Datenmodell mit den jeweiligen Tabellen und Feldern wird entworfen.

2.3.1 Realisierungskonzept

Es wird ein neues Unity Projekt, ein Python Projekt und eine MySql Datenbank erstellt.

1. GameManager

Über das gesamte Spiel wird es einen so genannten GameManger geben. Dieser wird ein GameObject sein mit einem GameManager Script welcher, wie der Name schon impliziert, das Spiel Verwalten wird. Hier wird die Generierung der Welt, die Navigation durchs Menu und das Neustarten des Spieles, sobald der Spieler stirbt, gehandhabt.

2. Bewegung

Der Spieler hat die Möglichkeit sich zwischen 3 Bahnen Links und Rechts zu bewegen. Der Spieler kann auch Springen, um über Hindernisse zu kommen und Sliden/Ducken, um unter Hindernissen durchzukommen.

Den Input des Spielers wird durch den Touchscreen aufgenommen. Wir benutzen das Wischen des Fingers über den Bildschirm, um den Spieler zu kontrollieren. Das machen wir in dem wir die Start- und Endposition des Fingers nehmen und diese vergleichen, um herauszufinden in welche Richtung gewischt wurde. Basierend darauf ändern wir die Position des Spielers.

3. Menu

In Unity gibt es Buttons, diese haben einen OnPress() Event welches beim Drücken des Buttons aktiviert wird. Diese kann man mit Funktionen aus dem GameManager verbinden.

Das Menu würde als UI über dem Spiel angezeigt werden. Während dessen könnte der

Hintergrund das eigentliche Spiel anzeigen.

Für das Anzeigen der Skins kann man eine neue Kameraposition einsetzten, mit der Man die 3D Skins ansieht. Beim zurückwechseln zu Menu, wechselt man auch die Kameraposition zu der vom Anfang. Für die Top 10 Highscores zeigt man vermutlich auch nur UI an, deswegen muss sich die Kameraposition nicht ändern.

4. Sterben

Hier haben wir die Möglichkeit ein OnCollisionEnter() void zu benutzen, welcher aufgerufen wird wenn der eigene Collider mit einem anderem Collider in Berührung kommt. Wir können dann überprüfen womit wir kollidiert sind und falls nötig das Spiel beenden.

Nach dem Tod des Spielers kann eine Methode die Informationen über den Highscore an das backend senden.

Danach kann man das Spiel neustarten oder zurück ins Menu gehen. Hier hat man mehrere Möglichkeiten. Man könnte die Szene neu laden, das würde alles zurücksetzten, dazu müsste man aber das Menu in einer anderen Szene haben, damit man dieses nicht auch zurückgesetzt. Stattdessen könnte man die Welt neugenerieren und die Spieler Position zurücksetzten, beides wird durch eine Funktion im GameManager ausgeführt, um den gleichen Effekt zu erreichen. Diese Methode braucht mehr Code aber ist leichter für das System.

5. Coins & Punkte

Die Punkte werden über zeit hinzugefügt und am Oberen Rand des Bildschirmes angezeigt. Das wird vom GameManager gehandhabt. Die Coins hingegen werden GameObjects sein welche der Spieler finden kann. Sobald der Spieler mit einem Coin kollidiert, wird dieser zu seinen Coins hinzugefügt. Die Anzahl Coins die der Spieler hat werden lokal gespeichert.

6. Skins

Skins sind die 3D Objekt die den Spieler darstellen. Diese können in ihrem eigenen Abteil gekauft und ausgerüstet werden. Die Skins kosten eine bestimmte Anzahl von Coins. Sobald ein Skin gekauft wurde, wird das backend darüber informiert, damit dieses weiss welche Skins schon gekauft wurden.

7. Datenbank

Die Datenbank wird eine einfache MySql Datenbank sein. Diese wird wie im Datenmodell abgebildet implementiert.

8. Backend Schnittstelle

Die Backend Schnittstelle wird mit Python und dem Flask framwork gelöst. Dieses wird

9. Unittest backend

Unittests werden mit der «unittest» Library von Python erstellt.

10. Backend zu Frontend anbindung

Unity wird webrequests an das Python Backend versenden. Das Python Backend wird Sql Queries ausführen, um die Informationen von der Datenbank zu kriegen und schickt diese in Form von Json strings zurück an Unity.

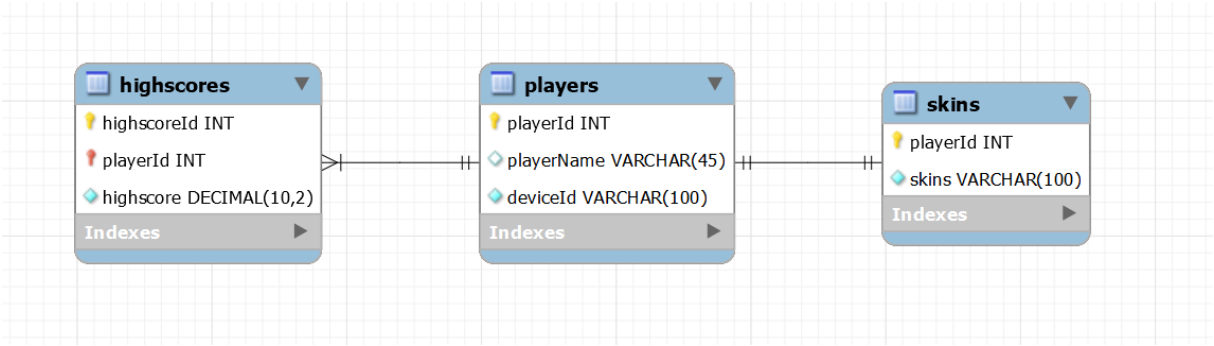
11. Assets

Alle 3D Assets werden in Blender modelliert und dann zu Unity Exportiert.

2.3.2 Datenmodell

Die Datenbank läuft lokal auf dem Arbeitslaptop des Kandidaten.

Es werden insgesamt 3 Tabellen benötigt, dies sind players, highscores und skins.



Tabellen:

- players:
  - playerId: INT AI
  - playerName: VARCHAR(45)
  - deviceId: VARCHAR(100)
- skins
  - playerId: INT Foreign Key (players.playerId)
  - skins: VARCHAR(100)
- highscores
  - highscoreId: INT AI
  - playerId: INT Foreign Key (players.playerId)
  - highscore: DECIMAL(10,2)

2.3.3 Testkonzept

Es wird die Black Box Testingmethode eingesetzt. Dies ist eine Testingmethode welche die Funktionalität einer Anwendung untersucht, ohne in ihre internen Strukturen oder Funktionsweisen einzudringen. So können wir einen klaren Einblick in die Userexperience schaffen.

Die Tests werden auf einem Android Handy durchgeführt. Basierend auf dem Test wird das Backen auf dem Laptop laufen und eine Verbindung mit dem Hotspot des Laptops wird stehen.

Testfall-Nr:	**1**
Anforderung:	Erstellen vom User muss funktionieren
Beschreibung:	Der Spieler sollte beim ersten Öffnen der App eine Aufforderung erhalten einen Spieler zu erstellen
Voraussetzung:	1.) Eine Verbindung zum Backend steht 2.) Der Name bleibt nicht leer
Erwartetes Resultat:	Ein neuer User wurde in der Datenbank erstellt

Testfall-Nr:	**2**
Anforderung:	Der Spieler kann das Spiel starten.
Beschreibung:	Der Spieler soll das Spiel Starten können, wenn er auf den Start-Knopf drückt.
Voraussetzung:	1.) Der Spieler-Name ist schon definiert
Erwartetes Resultat:	Das Spiel Startet

Testfall-Nr:	**3**
Anforderung:	Der Spieler kann die Skins einsehen.
Beschreibung:	Dem Spieler werden Skins Angezeigt wenn er auf den Skins-Knopf drückt.
Voraussetzung:	1.) Der Spieler-Name ist schon definiert
Erwartetes Resultat:	Die Skins werden Angezeigt

Testfall-Nr:	4
Anforderung:	Der Spieler kann einen Skin Ausrüsten.
Beschreibung:	Der Spieler kann auf den Equip-Knopf drücken um einen Skin auszurüsten.
Voraussetzung:	1.) Der Spieler-Name ist schon definiert 2.) Der Spieler besitzt mehrere Skins
Erwartetes Resultat:	Der Spieler hat den Skin ausgerüstet

Testfall-Nr:	5
Anforderung:	Der Spieler kann einen Skin Kaufen.
Beschreibung:	Der Spieler kann auf den Buy-Knopf drücken, um einen Skin zu kaufen.
Voraussetzung:	1.) Der Spieler-Name ist schon definiert 2.) Der Spieler hat genug Geld
Erwartetes Resultat:	Der Spieler hat den Skin gekauft

Testfall-Nr:	6
Anforderung:	Die Skins werden ans Backend versendet
Beschreibung:	Wenn ein Skin gekauft wurde, so wird das backend darüber informiert.
Voraussetzung:	1.) Der Spieler-Name ist schon definiert 2.) Der Spieler hat genug Geld 3.) Das Backend läuft 4.) Eine verbindung zum Backend steht
Erwartetes Resultat:	Die Info wurde an das Backend versendet

Testfall-Nr:	7
Anforderung:	Der Spieler kann Highscores einsehen
Beschreibung:	Der Spieler kann auf den Highscores-Knopf drücken um sowohl seinen als auch den Highscore von Anderen einsehen.
Voraussetzung:	1.) Der Spieler-Name ist schon definiert 2.) Die Verbindung zum Backend Steht 3.) Das Backend Läuft
Erwartetes Resultat:	Der Spieler kann die Highscores einsehen

Testfall-Nr:	8
Anforderung:	Der Spieler kann sich nach Links, Rechts, oben und unten Bewegen
Beschreibung:	Der Spieler kann in eine Richtung swipen um die Spielfigur in diese Richtung zu bewegen
Voraussetzung:	1.) Der Spieler-Name ist schon definiert 2.) Das Spiel läuft
Erwartetes Resultat:	Der Spieler kann sich in alle Richtungen bewegen

Testfall-Nr:	9
Anforderung:	Der Spieler kann Punkte verdienen.
Beschreibung:	Der Spieler verdient einen Punkt, wenn er ein Hindernis passiert.
Voraussetzung:	1.) Der Spieler-Name ist schon definiert.
Erwartetes Resultat:	Der Spieler verdient Punkte für das Passieren von Hindernissen.

Testfall-Nr:	10
Anforderung:	Der Spieler kann Sterben.
Beschreibung:	Der Spieler soll sterben, wenn er mit einem Hindernis kollidiert.
Voraussetzung:	1.) Der Spieler-Name ist schon definiert.
Erwartetes Resultat:	Der Spieler stirbt, wenn er mit einem Hindernis kollidiert.

Testfall-Nr:	11
Anforderung:	Der Score soll an das Backend versendet werden.
Beschreibung:	Der Score soll beim Tod an das Backend versendet.
Voraussetzung:	1.) Der Spieler-Name ist schon definiert. 2.) Die Verbindung zum Backend steht. 3.) Das Backend läuft
Erwartetes Resultat:	Der Score wird versendet.

2.4 Entscheiden

Das Entscheiden ist die dritte Phase der IPERKA-Methode.

In diese Phase wird entschieden welche Lösungsvarianten am sinnvollsten sind und geprüft, ob sie in dieser Form umgesetzt werden können.

2.4.1 Varianten

Wie man im Realisierungskonzept gesehen hat, gibt es für einige Ziele, welche über mehrere Lösungsmöglichkeiten verfügen. Wir schauen uns diese hier an:

Sterben

Variante 1: Unser gesamtes Spiel finden in einer Unity Szene statt. Man könnte diese neu laden um alles auf den Anfang zurück zu setzten. Ein Vorteil hier wäre das es so gut wie keinen Code brauchen würde, wir müssten einfach die Szene angeben, die geladen werden soll. Es würde auch garantiert alles zurücksetzen inklusive aller Komponenten in dieser Szene. Ein Nachteil wäre, dass das Spiel sich dann verhalten würde als hätten wir es neugestartet, man könnte nicht direkt neustarten, da das Spiel im Menu anfängt.

Variante 2: Diese Variante würde beinhalten das wir eine Funktion schreiben, die versucht alles wieder auf den Anfang zu setzten, das heisst wir müssen verfolgen, wo der Spieler startet und was wir bis Ende Spiel generiert haben, um alles wieder löschen zu können. Dies würde viel mehr code brauchen.

Bewegung

Variante 1: Unity enthält eine Komponente namens Rigidbody. Diese Komponente kann jedem Objekt Physische Qualitäten verleihen. Den Rigidbody könnten wir ansprechen und mit den richtigen Kräften ihn bewegen. Der Vorteil dieser Lösung ist eine reibungslose Bewegung, mit einer klaren Beschleunigung. Vor allem in der Vorwärtsbewegung könnte das ein Gefühl von wahrer Geschwindigkeit im Spieler auslösen. Dazu kommt auch die verbesserte Kollisionserkennung, die ein Rigidbody aufweist. Der Nachteil ist das man nur schwer eine kurze und schnelle Bewegung hinkriegt, da ein Rigidbody eine Beschleunigungs- und eine Bremszeit hat.

Variante 2: Komplett ohne neue Komponente kann man die Position eines Unity Objekts einfach direkt manipulieren. Durch ein paar variablen könnte man einfach eine rigidbodyartige Beschleunigung simulieren. Ein Vorteil wäre, dass man keine Unnötigen Physikkalkulation durchführen müsste, die mit dem Rigidbody kommen. Ein anderer wäre eine sehr knackige Bewegung, was vor allem für das Ändern der Bahnen nützlich sein könnte. Ein Nachteil wäre die Kollisionserkennung, welche ohne Rigidbody leiden würde, aber unser Spieler sollte sich nicht zu schnell bewegen als dass das ein wirkliches Problem werden würde.

Variante 3: Als Letztes könnte man eine Hybridlösung anwenden, wobei man die Rechts- & Linksbewegung Statisch mit direkter Manipulation durchführt, die Vorwärtsbewegung hingegen mit Physik durchführt. Das würde am meisten Code brauchen, würde wahrscheinlich aber auch zur besten Spielerfahrung führen. Man hätte dazu auch noch die verbesserte Kollisionserkennung des Rigidbody und zugleich auch die knackige Bahnen Änderung.

Menu

Für das Menu können wir zwischen zwei Varianten entscheiden. Entweder wir kreieren eine eigene Szene für das Menu. Das würde sich gut mischen mit der ersten Sterben Variante. Dann könnten wir das Neustarten des Spieles gut umsetzen, jedoch müssten wir alle Menu Optionen zwischen den Szenen übergeben, was mehr Code brauchen würde.

Sonst könnten wir ein Menu in der gleichen Szene erstellen wie das eigentliche Spiel, dazu müssten wir die zweite Sterben Variante benutzen. Ein Vorteil wäre das man keine Szenen neu laden müsste, was wahrscheinlich weniger Leistung brauchen würde, jedoch malwieder mehr Code. Es würde auch einen viel saubereren Übergang zwischen Menu und Spiel machen.

2.4.2 Entscheid

Beim Sterben habe ich mich dafür entschieden einfach eine Funktion zu konstruieren, da diese es mir einfacher macht das Menu System zu entwickeln für das ich mich entschieden habe.

Bei der Bewegung finde ich das eine Hybridlösung am besten für unsere Zwecke passen sollte, da es für eine glatte vorwärts Beschleunigung benutzt werden kann und dennoch einen sehr schnellen und plötzlichen Positionswechsel für die Input des Spielers liefert. Das mach diese Lösung die schwerste zum Umsetzen, aber sie sollte sich durch ein gutes Spielgefühl auszeichnen.

Zu guter Letzt wäre da noch das Menu. Hier entscheide ich mich für ein Menu in der gleichen Szene wie das Spiel, da dies zu einem nahtlosen Übergang zwischen Spiel und Menu führen sollte.

2.5 Realisierung

Das Realisieren ist die vierte Phase der IPERKA-Methode.

In dieser Phase wird vor allem die Programmierarbeit dokumentiert.

2.5.1 Projektumgebung

Wie in der Planung bereits beschrieben wird die App in Unity entwickelt, das Rest backend wird über python konstruiert und die Datenbank wird mit MySql umgesetzt.

In Unity wurde ein Leeres «3D Core» Projekt erstellt. Dieses enthält einen Leeren 3D Raum und keine nicht integrierten Packages.

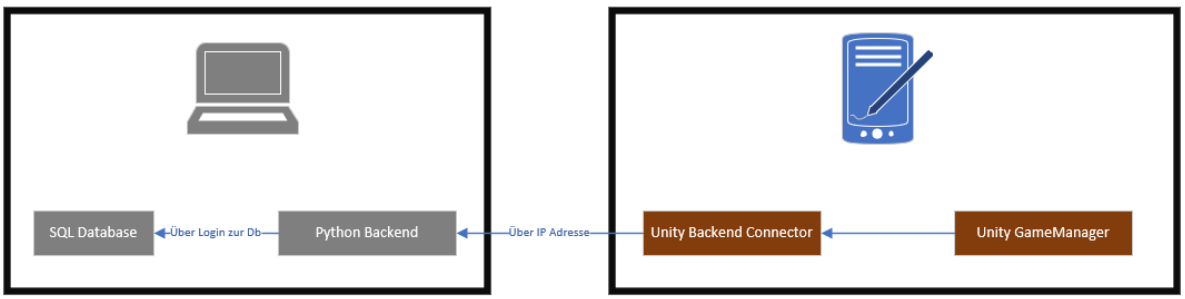
Installierte Packages:

- **Universal Render Pipeline**
- **Post Processing**
- **Visual Effect Graph**
- **Shader Graph**

Alle installierten Packages wurden nur zur Verbesserung der Grafik benutzt.

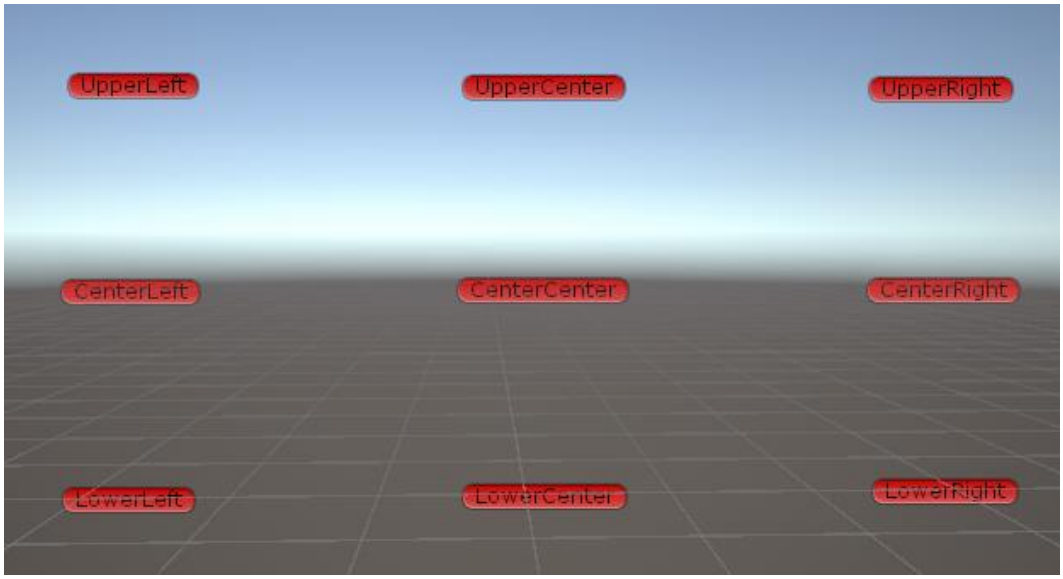
2.5.2 Set Up

Das gesamte System besteht aus dem App Frontend, dem Python Backend und der SQL Database. Das Python Backend und die SQL-Datenbank befinden sich beide auf einem Laptop. Der Laptop hat seinen Hotspot eingeschalten. Das Handy verbindet sich mit dem Hotspot des Laptops. Die App greift über das Python Backend auf die DB zu. Dazu hat es im Frontend eine BackendConnector Klasse, welche die anfragen auf das Backend ausführt. Das Python backend führt die passende SQL Query aus und Logged alle Zugriffe auf die Datenbank.



2.5.3 Bewegung

Die Bewegung wird durch den PlayerController Skript ermöglicht. Der Spieler hat 3 Bahnen, auf denen er sich bewegen kann und auf jeder kann er Springen und Slider. Das heisst das der Spieler insgesamt 9 verschiedene Position einnehmen kann. Aus diesem Grund existieren 9 Leere Objekte, die diese 9 Positionen repräsentieren. Sie bewegen sich mit dem Spieler und werden in einem von 3 «PositionCluster» Structs gespeichert, wo sie dann in einem Array abgelegt werden. Dies ergibt etwas sehr Ähnliches zu einem 2D Array. Die Postitionen werden spezifisch angeordnet abgelegt damit man sich logisch durch diese bewegen kann. Das Spielerobjekt versucht immer eine Zielposition zu erreichen welche einem dieser 9 Positionen entspricht. Der SpielerController hat auch zwei Variablen namens currentVerticalPosition und currentHorizontalPosition mit denen die derzeitige Zielposition im 2D Array ausgelesen und die Zielposition des Spielers verändert werden kann.



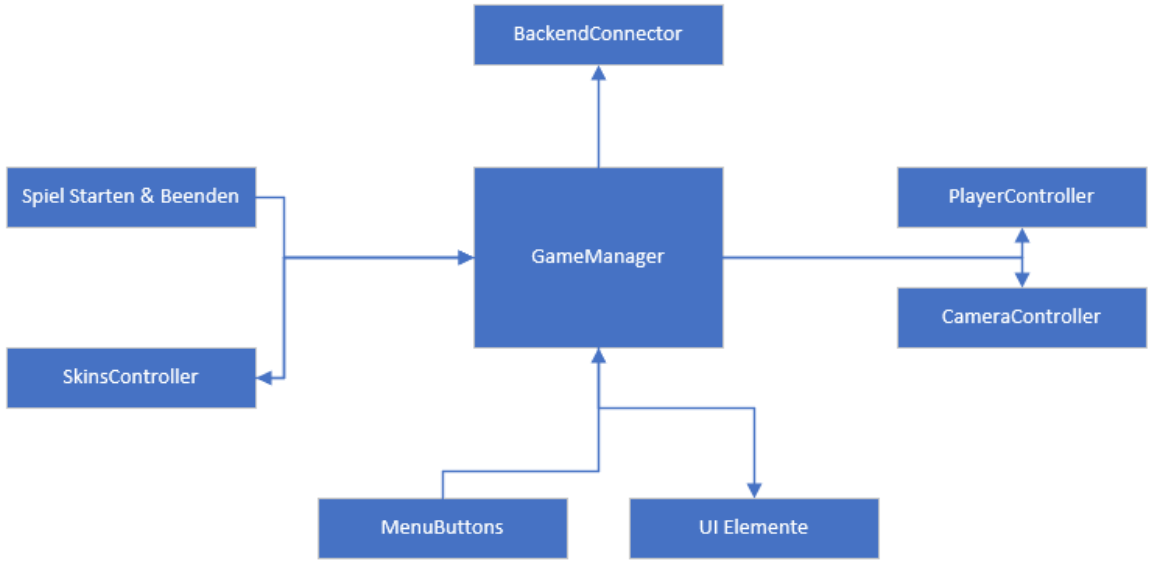
Im PlayerController Skript wird überprüft ob und in welche Richtung der Spieler geswiped hat. Dies wird durch eine Kontrolle der Position des Touch Input gemacht, wo dieser Angefangen hat und wo er jetzt ist. Wenn sich die Position genug geändert hat, wird die derzeitige Position des Spielers passend geändert. Die Richtung des swipe wird durch einen einfachen Grössenvergleich der Horizontalen und Vertikalen Bewegung ermittelt.

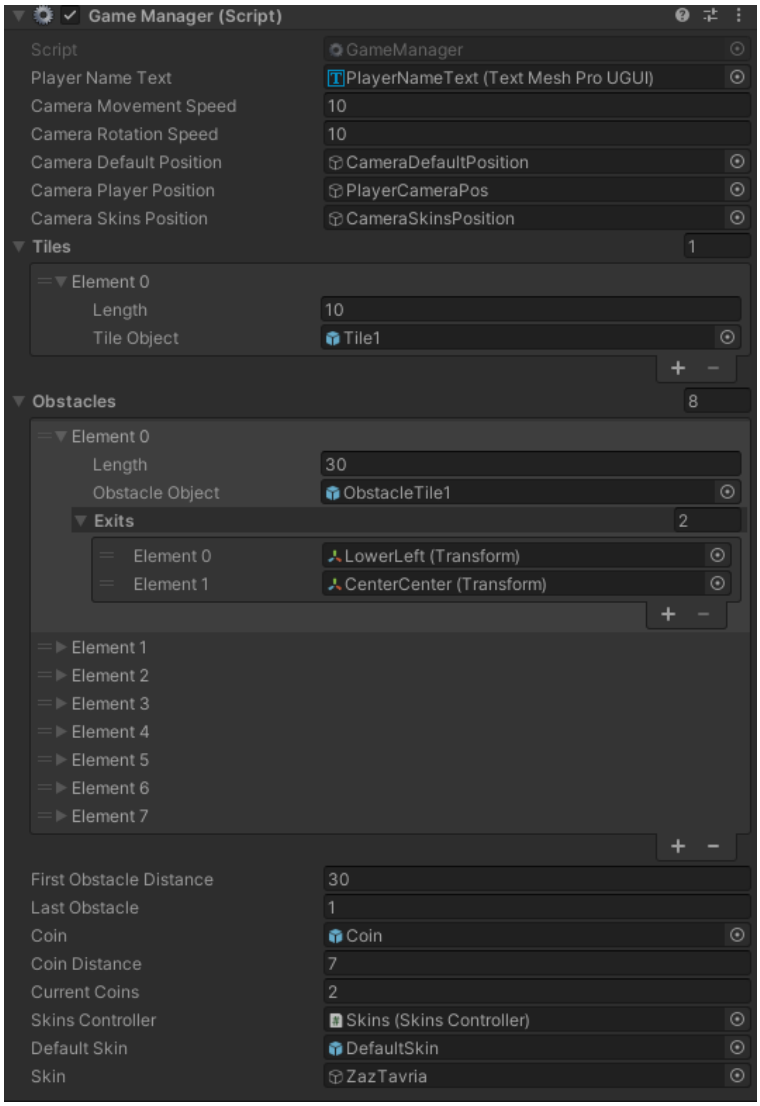
```
switch (touch.phase)
{
    case TouchPhase.Began:
        firstPosition = touch.position;
        lastPosition = touch.position;
        break;
    case TouchPhase.Moved:

        if (!moved)
        {
            lastPosition = touch.position;

            if (Mathf.Abs(lastPosition.x - firstPosition.x) + 20 > dragDistance ||
                Mathf.Abs(lastPosition.y - firstPosition.y) > dragDistance)
            {
                moved = true;
                if (Mathf.Abs(lastPosition.x - firstPosition.x) > Mathf.Abs(lastPosition.y -
                                                                           firstPosition.y))
                {
                    if ((lastPosition.x > firstPosition.x))
                    {
                        if (currentHorizontalPosition < horizontalPositions.Length - 1)
                        {
                            currentHorizontalPosition++;
                        }
                    }
                    else
                    {
                        if (currentHorizontalPosition > 0)
                        {
                            currentHorizontalPosition--;
                        }
                    }
                }
                else
                {
                    if (lastPosition.y > firstPosition.y)
                    {
                        if (!isJumping && !isDucking)
                        {
                            StartJump();
                        }
                    }
                    else
                    {
                        if (!isJumping && !isDucking)
                        {
                            StartDucking();
                        }
                    }
                }

                currentTransform = horizontalPositions[currentHorizontalPosition]
                    .verticalPositions[currentVerticalPosition].transform;
            }
        }

        break;
    case TouchPhase.Ended:
    {
        moved = false;
        break;
    }
}
```

PlayerController: CheckInput() funktion

2.5.4 Game Manager

Der GameManager ist für alles um den Spieler herum zuständig, die Navigation durchs Menu, das Versenden von Daten an das Backend. Dazu hat der GameManager verschiedene Funktionen, die von anderen Systemen angesprochen werden. Hier erkläre ich nur die Funktionen, die nirgendwo sonst benutzt werden.


BackendConnector
Spiel Starten & Beenden
GameManager
PlayerController
CameraController
SkinsController
MenuButtons
UI Elemente


Game Manager (Script)
Script
GameManager
Player Name Text
PlayerNameText (Text Mesh Pro UGUI)
Camera Movement Speed
10
Camera Rotation Speed
10
Camera Default Position
CameraDefaultPosition
Camera Player Position
PlayerCameraPos
Camera Skins Position
CameraSkinsPosition
Tiles
1
Element 0
Length
10
Tile Object
Tile1
Obstacles
8
Element 0
Length
30
Obstacle Object
ObstacleTile1
Exits
2
Element 0
LowerLeft (Transform)
Element 1
CenterCenter (Transform)
Element 1
Element 2
Element 3
Element 4
Element 5
Element 6
Element 7
First Obstacle Distance
30
Last Obstacle
1
Coin
Coin
Coin Distance
7
Current Coins
2
Skins Controller
Skins (Skins Controller)
Default Skin
DefaultSkin
Skin
ZazTavria

Structs
Ein Struct ist eine C++-Datenstruktur, die verwendet werden kann, um Elemente unterschiedlicher Datentypen zusammen zu speichern. In Unserem GameManager haben wir 2 Structs diese sind Obstacle und Tile.

Obstacle:

- Length: Abstand zum nächsten Hindernis.
- obstacleObject: das zu generierende Hindernis.
- exits: Ein Array für die möglichen Durchgänge bei einem Hinderniss.

```
public struct Obstacle
{
    public float length;
    public GameObject obstacleObject;
    public Transform[] exits;
}
```
GameManager: Obstacle Struct

Tile:

- Length: Abstand zum nächsten Tile.
- tileObject: Die zu generierende Tile.

```
public struct Tile
{
    public float length;
    public GameObject tileObject;
}
```
GameManager: Tile Struct

Die grösste Funktion, die der GameManager während des Spiels ausführt, ist die Generation der Spielwelt. Er lässt die Bahnen, die Hindernisse und die Coins auf dem weg des Spielers entstehen. Dies Passiert alles durch das Aufrufen der richtigen Funktionen durch den Spieler. Wenn der Spieler ein Hindernis passiert, in dem er den unsichtbaren Box Collider berührt, so wird die Funktion «GenerateRandomObstacle» ausgeführt, wobei eine Zufallszahl generiert wird, welche nicht grösser sein kann als die Anzahl von Hindernissen im Array. Danach wird das Hindernis der mit der Zufallszahl übereinstimmt in der Welt platziert auf den Koordinaten [0, 0, currentGenerationZ]. currentGenerationZ ist die derzeitige Vorwärtsposition des Hindernisses, diese wird dann mit dem Abstand von diesem Hinderniss erweitert. Dazu wird der Score des Spielers auch um 1 erhöht.

```
public void GenerateRandomObstacle()
{
    int i = Random.Range(0, obstacles.Length);
    GenerateObstacle(i);
}

public void GenerateObstacle(int i)
{
    GameObject x = Instantiate(obstacles[i].obstacleObject,
        transform.position + Vector3.forward * currentObstacleGenerationZ, transform.rotation,
        null);
    generatedObstacles.Add(x);

    GenerateCoins(i);

    currentObstacleGenerationZ += obstacles[i].length;
    if (GameStarted)
    {
        if (player.GetComponent<PlayerController>().currentHorizontalPosition == 1)
        {
            score += 1;
        }

        score += 1;
        scoreText.text = score.ToString();
    }
}
```

GameManager: GenerateRandomObstacle() und GenerateObstacle(int i) funktionen.

Wenn der Spieler eine Bahn passiert, dann wird «GenerateRandomTile» ausgeführt, wobei es bei den Bahnen genau gleich durchgeführt wird wie bei den Hindernissen, ausser dass der Spieler keinen Score dazu kriegt.

```
public void GenerateRandomTile()
{
    int i = Random.Range(0, tiles.Length);
    GenerateTile(i);
}

private void GenerateTile(int i)
{
    GameObject x = Instantiate(tiles[i].tileObject,
        transform.position + Vector3.forward * currentTileGenerationZ,
transform.rotation,
        null);
    generatedTiles.Add(x);
    currentTileGenerationZ += tiles[i].length;
}
```

GameManager: GenerateRandomTile() und GenerateTile(int i) funktionen

Bei den Coins wird es wieder komplizierter. Die Coins werden immer mit den Hindernissen generiert. Der GenerateCoins funktion wird das derzeitig generierte Hindernis übergeben. Die Anzahl der Coins die generiert werden sollen wird ausgerechnet mit:

[Anzahl Coins] = [Die Länge des übergebenen Hindernisses] / [Abstand zwischen Coins]

Die Position der Coins wird herausgefunden in dem man einen zufälligen Eingang des derzeitigen Hindernisses nimmt und einen Zufälligen Ausgang des vorherigen Hindernisses nimmt und dessen Positionen vergleicht. Wenn der vertikale Abstand zwischen ihnen kleiner als oder gleich zu 1 ist, dann werden 2 Linien von Coins generiert mit ½ der zu generierenden Coins auf einer Linie was c.a. ½ der eigentlichen Länge des übergebenen Hindernisses entspricht. Wenn der Abstand grösser ist, dann werden 3 Linien mit ⅓ der Coins und Länge generiert. Danach werden das übergebene Hindernis und der zufällige Eingang als letztes Hindernis und letzter Eingang gespeichert.

```
private void GenerateCoins(int currentObstacle)
{
    int amountOfCoins = (int) (obstacles[currentObstacle].length / coinDistance);
    if (generatedObstacles.Count > 2)
    {
        if (lastExit == null)
        {
            lastExit = obstacles[lastObstacle].exits[Random.Range(0,
                                                obstacles[lastObstacle].exits.Length)];
        }

        Transform currentExit = obstacles[currentObstacle]
            .exits[Random.Range(0, obstacles[currentObstacle].exits.Length)];
        if (1 <= Mathf.Abs(lastExit.position.y - currentExit.position.y))
        {
            for (int currentCoin = 0; currentCoin < amountOfCoins / 2; currentCoin++)
            {
                generatedCoins.Add(Instantiate(coin,
                    new Vector3(lastExit.position.x, lastExit.position.y,
                        (currentObstacleGenerationZ - obstacles[lastObstacle].length) +
                        coinDistance * currentCoin), transform.rotation, null));
            }

            for (int currentCoin = 0; currentCoin < amountOfCoins / 2; currentCoin++)
            {
                generatedCoins.Add(Instantiate(coin,
                    new Vector3(currentExit.position.x, currentExit.position.y,
                        (currentObstacleGenerationZ - obstacles[lastObstacle].length) +
                    coinDistance * (currentCoin + amountOfCoins / 2)),transform.rotation, null));
            }
        }
        else
        {
            for (int currentCoin = 0; currentCoin < amountOfCoins / 3; currentCoin++)
            {
                generatedCoins.Add(Instantiate(coin,
                    new Vector3(lastExit.position.x, lastExit.position.y,
                        (currentObstacleGenerationZ - obstacles[lastObstacle].length) +
                                    coinDistance * currentCoin),transform.rotation, null));
            }

            if (1 < Mathf.Abs(lastExit.position.x - currentExit.position.x))
            {
                for (int currentCoin = 0; currentCoin < amountOfCoins / 3; currentCoin++)
                {
                    generatedCoins.Add(Instantiate(coin,
                        new Vector3(lastExit.position.x, lastExit.position.y,
                            (currentObstacleGenerationZ - obstacles[lastObstacle].length) +
                    coinDistance * (currentCoin + amountOfCoins / 3)), transform.rotation, null));
                }
            }
            else
            {
                for (int currentCoin = 0; currentCoin < amountOfCoins / 3; currentCoin++)
                {
                    generatedCoins.Add(Instantiate(coin,
                        new Vector3(lastExit.position.x,
                            player.GetComponent<PlayerController>().startPosition.y + 1.5f,
                            (currentObstacleGenerationZ - obstacles[lastObstacle].length) +
                    coinDistance * (currentCoin + amountOfCoins / 3)), transform.rotation, null));
                }
            }

            for (int currentCoin = 0; currentCoin < amountOfCoins / 3 + 1; currentCoin++)
            {
                generatedCoins.Add(Instantiate(coin,
                    new Vector3(currentExit.position.x, currentExit.position.y,
                        (currentObstacleGenerationZ - obstacles[lastObstacle].length) +
          coinDistance * (currentCoin + (amountOfCoins / 3) * 2)), transform.rotation, null));
            }
        }

        lastExit = currentExit;
    }

    lastObstacle = currentObstacle;
}
```

GameManager: GenerateCoins(int currentObstacle) Funktion.

2.5.5 Kamera

Die Kamera ändert ihre Position während der App Benutzung mehrmals. Dazu hat die Kamera einen CameraController. In diesem kann man für die Kamera eine Zielposition definieren und die Kamera wird sich zu dieser mit Vector3.Lerp bewegen.

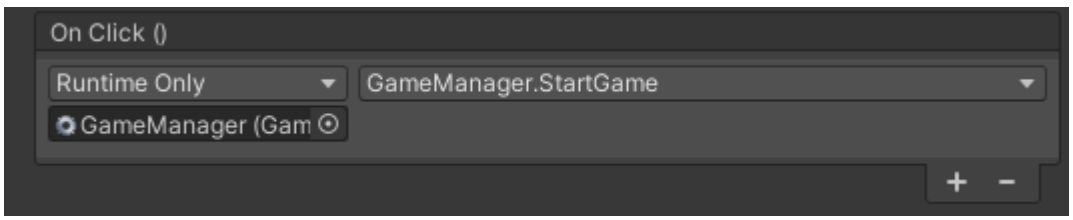
Dazu hat die Kamera auch eine «Shake» Funktion die einen Schütteleffekt bei der Kamera auslöst.

```
public class CameraController : MonoBehaviour
{
    public GameObject currentCameraPosition;
    public float cameraMovementSpeed;
    public float cameraRotationSpeed;
    public float elapsed = 0;
    public float duration = 0.05f;
    public float magnitude = 0;
    private void FixedUpdate()
    {

        transform.position = Vector3.Lerp(transform.position,
currentCameraPosition.transform.position, cameraMovementSpeed * Time.deltaTime);
        transform.rotation = Quaternion.Lerp(transform.rotation,
currentCameraPosition.transform.rotation,
            cameraRotationSpeed * Time.deltaTime);
    }

    private void Update()
    {
        if (elapsed < duration)
        {
            float x = Random.Range(-1f, 1f) * magnitude;
            float y = Random.Range(-1f, 1f) * magnitude;

            transform.position += new Vector3(x, y);

            elapsed += Time.deltaTime;
            Debug.Log("Shaking");

        }
    }

    public void Shake(float elapsed, float magnitude)
    {
        this.elapsed = elapsed;
        this.magnitude = magnitude;
    }
}
```

CameraController: CameraController Klasse.

2.5.6 Menu

Das Menu System wird vor allem durch die Unity Buttons gesteuert. Diese haben ein «OnClick» Event welche verschiedenen Funktionen aus dem GameManager ausführen.



Das Navigieren durch das Menu, wird durch das Aktivieren und Deaktivieren von UI Objekten ermöglicht, was diese UI Objekte auch neben anderem Unsichtbar macht.

Das Menu hat 3 Seiten und das Login & Game Over Ui.

Auf dem Start Menu befinden sich 3 Buttons.

Spiel Starten button, welcher die «StartGame» Funktion im GameManager ausführt.

```
public void StartGame()
{
    currentObstacleGenerationZ = firstObstacleDistance;
    menu.SetActive(false);
    player.GetComponent<PlayerController>().selectedSkin = skin != null ? skin : defaultSkin;
    player.SetActive(true);
    camera.currentCameraPosition = cameraPlayerPosition;
    GenerateStart();
    GameStarted = true;
    gameUi.SetActive(true);
}
```

GameManager: StartGame() funktion.

Die Funktion deaktiviert das Menu, aktiviert den Spieler und setzt die Kamera Position zu dem hinter dem Spieler.

Skins, was die «ToSkinsFromMenu» Funktion ausführt.

```
public void ToSkinsFromMenu()
{
    camera.currentCameraPosition = cameraSkinsPosition;
    menu.SetActive(false);
    skinSelector.SetActive(true);
    skinsController.UpdateCoinsText();
    skinsController.UpdateBuyEquipText();
}
```

GameManager: ToSkinsFromMenu() funktion.

Diese Funktion deaktiviert wieder das Menu und setzt die Kamera Position zu der bei der Skinbühne.

Auf dieser Seite können wir uns die Skins anschauen und durch sie Navigieren, sie kaufen und ausrüsten. Hier sehen wir auch die gesamte Anzahl von Coins des Spielers. Durch das Drücken der Rechts- und Linksbuttons werden die «NextSkin» und «PreviousSkin» Funktionen aufgerufen.

```
public void NextSkin()
{
    if (currentlyDisplayedSkin < skins.Length - 1)
    {
        skins[currentlyDisplayedSkin].gameObject.SetActive(false);
        currentlyDisplayedSkin += 1;
        skins[currentlyDisplayedSkin].gameObject.SetActive(true);
        UpdateBuyEquipText();
    }
}

public void PreviousSkin()
{
    if (currentlyDisplayedSkin > 0)
    {
        skins[currentlyDisplayedSkin].gameObject.SetActive(false);
        currentlyDisplayedSkin -= 1;
        skins[currentlyDisplayedSkin].gameObject.SetActive(true);
        UpdateBuyEquipText();
    }
}
```

SkinsController: NextSkin() und PreviousSkin() funktionen.

Highscore, was die Funktion «ToHighscoreFromSkins» aufruft.

```
public void ToHighscoreFromMenu()
{
    highscoreUi.SetActive(true);
    menu.SetActive(false);
    connector.getHighscores(highscoreUiContent);
}
```

GameManager: ToHighscoreFromMenu() funktion.

Neben dem typischen Aktivieren und Deaktivieren, wird auch im BackendConnector die funktion «GetHighscores» ausgeführt. Diese fragt das Backen nach Highscores ab und instanziert dann UIObjekte für jeden Highscore.

Das Menu enthält auch eine Login-Seite, welche aufgeht, wenn der Spielername nicht in den PlayerPrefs gesetzt ist.

Der Spieler wird aufgefordert einen Namen einzugeben und dieser wird dann mit der DeviceId an das Backend versendet.

Wenn der Spieler stirb so wird das GameOver Ui aufgerufen.

Dieses enthält einen «Restart» Button und einen «Back To Menu» Button.

Restart ruft die «Reset» funktion im GameManager auf.

```
public void Restart()
{
    ResetGame();
    StartGame();
}
```

GameManager: Restart() funktion.

Diese Ruft die «ResetGame» und «StartGame» (siehe s.31) Funktionen auf.

```
private void ResetGame()
{
    currentCoins = 0;
    player.SetActive(true);
    player.transform.position = player.GetComponent<PlayerController>().startPosition;
    player.GetComponent<PlayerController>().maxSpeed = defaultMaxSpeed;
    player.SetActive(false);
    gameOverUi.SetActive(false);
    gameUi.SetActive(false);
    foreach (GameObject x in generatedTiles)
    {
        Destroy(x);
    }

    foreach (GameObject x in generatedObstacles)
    {
        Destroy(x);
    }

    foreach (GameObject x in generatedCoins)
    {
        Destroy(x);
    }

    generatedTiles = new ArrayList();
    generatedObstacles = new ArrayList();
    generatedCoins = new ArrayList();
    currentObstacleGenerationZ = 0;
    currentTileGenerationZ = 0;
    score = 0;
    scoreText.text = score.ToString();
    GameStarted = false;
}
```

GameManager: ResetGame() Funktion.

Die «ResetGame» Funktion setzt alle spielrelevanten Variabeln auf ihren Standard werte.

2.5.7 Sterben

Der Spieler stirbt, wenn er mit einem Hindernis kollidiert. Der Check, ob der Spieler kollidiert, passiert im Spieler-Objekt. Wenn die «OnCollisionEnter» Funktion des Spielers aufgerufen wird. So überprüft der Spieler, ob er mit einem Objekt kollidiert, ist was einen «Obstacle» tag draufhatte. Wenn das so ist, wird die «Die» Funktion aufgerufen.

```
private void OnCollisionEnter(Collision collision)
{
    if (collision.gameObject.CompareTag("Obstacle"))
    {
        Die();
        Debug.Log(collision.gameObject.transform.position);
    }
}
```

PlayerController: OnCollisionEnter (Collision collision) Funktion

In der «Die» Funktion wird die Position und Zielposition des Spielers auf die Zentrale Position gesetzt, das Spielerobjekt deaktiviert und die «GameEnd» Funktion des GameManagers aufgerufen.

```
private void Die()
{
    gm.GameEnd();
    GameObject currentExplosion =
        Instantiate(explosionObject, playerTransform.position,
playerTransform.rotation, null);
    Destroy(currentExplosion, 10);
    playerTransform.position =
horizontalPositions[1].verticalPositions[1].transform.position;
    currentTransform =
horizontalPositions[1].verticalPositions[1].transform;
    transform.position = startPosition;
    currentHorizontalPosition = 1;
    currentVerticalPosition = 1;
    gameObject.SetActive(false);
}
```

PlayerController: Die() Funktion.

Die GameEnd funktion im GameManager speichert die gesammelten Coins, Aktiviert das GameOver UI, setzt eine neue Position für die Kamera und verschickt den Highscore an das Backend.

2.5.8 Coins

Die Coins werden als Objekte in der Spielwelt generiert (Siehe Game Manager). Wenn der Spieler mit einem der Coins kollidiert, so wird dies dem GameManager übergeben und diese fügt eine Anzahl von Coins zu der Gesamtanzahl von Coins hinzu. Dazu wird auch ein kleines Partikeleffekt instanziiert.

```
public void Pop()
{
    gm.AddCoins(value);
    GameObject currentDeath=Instantiate(death, transform.position, transform.rotation, null);
    Destroy(currentDeath, 2);
    Destroy(gameObject);
}
```

CoinController: Pop() Funktion.

Die Anzahl der gesammelten Coins wird am Ende des Spiels angezeigt und zu der gesamten Anzahl Coins die der Spieler besitzt, hinzugefügt. Am Ende von jeder Runde werden die Coins auch lokal in den PlayerPrefs gespeichert.

2.5.9 Score

Der Score ist die Punktezahl, die der Spieler erreicht hat. Der Spieler kriegt einen Punkt für jedes Hindernis, welches er überwältigt. Diese werden am Ende des Spieles angezeigt und an das Backend versendet.

```
public void GameEnd()
{
    allCoins += currentCoins;
    PlayerPrefs.SetInt("Coins", allCoins);
    Destroy(cameraDeathPosition);
    cameraDeathPosition = new GameObject();
    var dathPos = cameraDeathPosition.transform.position;
    dathPos = camera.currentCameraPosition.transform.position;
    cameraDeathPosition.transform.rotation = camera.currentCameraPosition.transform.rotation;
    dathPos = new Vector3(dathPos.x, dathPos.y, dathPos.z - 5);
    cameraDeathPosition.transform.position = dathPos;
    camera.currentCameraPosition = cameraDeathPosition;
    gameOverUi.SetActive(true);
    coinText.text = currentCoins.ToString();
    if (GameStarted)
    {
        camera.Shake(-1, 0.3f);
    }
    SetNewHighscore(score);
}
```

GameManager: GameEnd() Funktion.

2.5.10 Skins

Skins sind in unserem Spiel die 3D Modelle, die unseren Spieler darstellen. Diese werden mit dem Spiel heruntergeladen und sind nur lokal verfügbar. Die Skin Objekte werden mit einem SkinObjectController ausgestattet. Die SkinObjectController werden in einem Array im SkinController gespeichert. Der Spieler kann sich Skins kaufen mit den Coins die er während des Spieles sammelt. Wenn der Spieler einen Skin kauft, so wird das in den PlayerPrefs mit Keys und im Backend mit einem JsonBooleanArray festgehalten. Der SkinsController enthält ein Boolean Array, welcher die gleiche länge hat wie die Anzahl Skins. Wenn der Boolean auf True ist so besitzt der Spieler den Skin, welcher auf der gleichen Position in seinem Array sitzt. Wenn nicht, dann muss sich der Spieler den Skin zuerst Kaufen für den Preis, welcher im SkinObjectController definiert ist.

```
public class SkinObjectController : MonoBehaviour
{
    public GameObject skin;
    public int price;

    void Update()
    {
        skin.transform.Rotate(new Vector3(0, 30 * Time.deltaTime, 0));
    }
}
```

SkinObjectController: SkinObjectController Klasse

Neben der Aufgabe, die wichtigen Information über den Skin zu speichern, rotiert die Klasse auch den Skin kontinuierlich.

2.5.11 Datenbank

Die Datenbank befindet sich auf einem Laptop. Auf die Datenbank wird über ein Python Rest Interface zugegriffen (für Datenmodell siehe Datenmodell).

2.5.12 Backend Schnittstelle

Die Backend Schnittstelle wurde mit dem Flask Framwork für Python umgesetzt. Die Schnittstelle kommuniziert mit der Datenbank, welche sich auf dem gleichen Gerät, befindet.

Es benutzt die mysql-connector-python Library, um sich mit der MySql Datenbank zu verbinden.

Dieses Backend hat 4 Pfade, die man ansprechen kann.

Pfade:

- /createNewPlayer/<playerName>/<deviceId>
  - Fügt einen neuen Spieler in die Datenbank hinzu und gibt einen String zurück
  - Die deviceId wird benutzt, um den Spieler zu finden
  - Bei einem Fehler gibt es einen Error String zurück

```
@app.route("/createNewPlayer/<playerName>/<deviceId>")
def create_new_player(playerName, deviceId):
    mycursor = mydb.cursor()
    try:
        mycursor.execute("INSERT INTO players(playerName, deviceId) VALUES('" + playerName +
"', '" + deviceId + "')")
        mycursor.close()
        mydb.commit()
        logging.info("Successfully ran: INSERT INTO players(playerName, deviceId) VALUES('" +
playerName + "', '" + deviceId + "')")
        return "Success"
    except mysql.connector.Error as err:
        logging.error("Something went wrong: {}".format(err))
        return "Failure"
```

Python Main: create_new_player(playerName, deviceId) Funktion.

- /setNewHighscore/<deviceId>/<highscore>
  - Fügt einen neuen Highscore in die Datenbank hinzu und gibt einen String zurück
  - Die deviceId wird benutzt, um den Spieler zu finden
  - Bei einem Fehler gibt es einen Error String zurück

```
@app.route("/setNewHighscore/<deviceId>/<highscore>")
def create_new_highscore(deviceId, highscore):
    try:
        mycursor = mydb.cursor()
        mycursor.execute("SELECT playerId FROM players WHERE deviceId = '" + deviceId + "'")
        x = mycursor.fetchall()
        print("INSERT INTO highscores(playerId, highscore) VALUES(" + str(x[0][0]) + ", " +
highscore + ")")
        mycursor.execute("INSERT INTO highscores(playerId, highscore) VALUES(" + str(x[0][0])
+ ", " + highscore + ")")
        mycursor.close()
        mydb.commit()
        logging.info("Successfully ran: INSERT INTO highscores(playerId, highscore) VALUES(" +
str(x[0][0]) + ", " + highscore + ")")
        return str(x)
    except mysql.connector.Error as err:
        logging.error("Something went wrong: {}".format(err))
        return "Failure"
```

Python Main: create_new_highscore(deviceId, highscore) Funktion.

- /getHighscores
  - Gibt die Top 100 Highscores in einem Json-String zurück
  - Bei einem Fehler gibt es einen Error String zurück

```
@app.route("/getHighscores")
def get_highscores():
    try:
        mycursor = mydb.cursor()
        mycursor.execute("SELECT players.playerName, highscores.highscore FROM highscores
INNER JOIN players ON players.playerId = highscores.playerId ORDER BY highscore DESC LIMIT
100")
        result = mycursor.fetchall()
        mycursor.close()
        logging.info("Successfully ran: SELECT * FROM highscores ORDER BY highscore DESC LIMIT
100")
        return json.dumps(result)
    except mysql.connector.Error as err:
        logging.error("Something went wrong: {}".format(err))
        return "Failure"
```

Python Main: get_highscores() Funktion.

- /skins/<deviceId>/<skins>
    - Fügt ein oder ändert den Skins Json-String in der Datenbank
    - Bei einem Fehler gibt es einen Error String zurück

```
@app.route("/skins/<deviceId>/<skins>")
def set_skins(deviceId, skins):
    try:
        mycursor = mydb.cursor()
        mycursor.execute("SELECT playerId FROM players WHERE deviceId = '" + deviceId + "'")
        x = mycursor.fetchall()
        mycursor.execute("INSERT INTO skins(playerId, highscore) VALUES(" + str(x[0][0]) + ",
" + skins + ")")
        result = mycursor.fetchall()
        mycursor.close
        return "success"
    except mysql.connector.Error as err:
        logging.error("Something went wrong: {}".format(err))
        return "Failure"
```

Python Main: set_skins(deviceId, skins) funktion.

Die «set_skins» Funktion wird nie eingesetzt, da diese Funktionalität aus Zeitgründen nie eingebaut wurde.

Hier wird auch die logging Library von Python eingesetzt. Diese logged alle Ereignisse in einem Logfile im gleichen Ordner.

2.5.13 Backend zu Frontend Anbindung

Die Anbindung an das Backend wird von der BackendConnector Klasse geregelt. Diese führt WebRequests aus, mit denen es das Python Backend erreichen versucht.

Es hat 4 IEnumerator Funktionen, welche vom GameManager ausgeführt werden.

```
private IEnumerator CreateNewPlayer(string playerName, string deviceId)
{
    Debug.Log("SetUP");
    using (UnityWebRequest webRequest =
           UnityWebRequest.Get(URI + "/createNewPlayer/" + playerName + "/" + deviceId))
    {
        yield return webRequest.SendWebRequest();


        switch (webRequest.result)
        {
            case UnityWebRequest.Result.ConnectionError:
            case UnityWebRequest.Result.DataProcessingError:
                Debug.LogError("Error: " + webRequest.error);
                break;
            case UnityWebRequest.Result.ProtocolError:
                Debug.LogError("HTTP Error: " + webRequest.error);
                break;
            case UnityWebRequest.Result.Success:
                Debug.Log("Received: " + webRequest.downloadHandler.text);
                break;
        }
    }
}
```

BackendConnector: CreateNewPlayer(string playerName, string deviceId) funktion.

Falls dies nicht funktioniert, wird ein Error geloggt. Das Spiel wird nicht unterbrochen und der Spieler merkt so gut wie nichts davon. Eine Ausnahme ist die Highscore Seite, auf der nichts angezeigt wird.

2.6 Kontrollieren

Das Kontrollieren ist die fünfte Phase der IPERKA-Methode. Hier werden Korrekturen sowie gefundene Fehler beschrieben. Mit dem Testkonzept, das in der Phase Planung erstellt wurde, wird die App und das Backend getestet.

2.6.1 Testprotokoll

Angaben:

Handy	Oppo Find X2 Pro
Laptop	**HP Elitebook 840 G6**

Testfall	Resultat	Datum	Tester	Bemerkung	Unterschrift
1	Erfolgt	15.03.2022	Artem K.	Bei der Eingabe eines Leeren Namen wird beim Neustarten die Login-Seite wieder aufgerufen	Artem
2	Erfolgt	15.03.2022	Artem K.	-	Artem
3	Erfolgt	15.03.2022	Artem K.	-	Artem
4	Erfolgt	15.03.2022	Artem K.	-	Artem
6	Gescheitert	15.03.2022	Artem K.	Diese Funktionalität wurde nicht eingebaut	Artem
7	Erfolg	15.03.2022	Artem K.	-	Artem
8	Erfolg	15.03.2022	Artem K.	-	Artem
9	Erfolg	15.03.2022	Artem K.	-	Artem
10	Erfolg	15.03.2022	Artem K.	-	Artem
11	Erfolg	15.03.2022	Artem K.		Artem

2.6.2 Testbericht

Testresultate sehen gut aus, abgesehen von Test 6. Das liegt daran das ich keine Zeit mehr hatte diese Funktionalität einzubauen. Dies sollte zu keinen grossen Problemen führen da diese Funktionalität nur zu Dokumentationszwecken geplant war. Im ersten Test gibt es ein kleines Problem mit dem Namen. Wenn dieser leer bleibt so wird nach ihn erneut gefragt beim nächsten Aufstarten der App. Das liegt daran das die PlayerPrefs immer einen String zurückgeben, auch wenn unter dem gegebenen Key nichts gesetzt wurde. Deswegen wird geschaut, ob dieser String leer ist oder nicht.

2.7 Reflexion

Die Informieren-Phase verlief eigentlich gut. Die Aufgabenstellung war klar und ich konnte mir einen guten Überblick verschaffen und musste kaum nachfragen.

Die Planen-Phase verlief auch gut, da ich schon einen generellen Plan hatte wie ich alles umsetzten wollte. Ich habe hier das erste Mal bemerkt, wie viel Zeit die Dokumentation verbrauchen würde.

Dank der Planung konnte ich beim Realisieren gut starten und wusste genau was ich zu tun hatte. Das grösste Problem war hier ganz klar die Dokumentation, welche mir viel Zeit gekostet hat, welche ich eigentlich nicht eingeplant hatte. Neben dem hatte ich auch Probleme mit der Datenbank welche ab und zu nicht funktionieren wollte.

Beim Realisieren fing es gut an, wurde zum Ende hin aber ziemlich stressig. Ich hatte nicht genug Zeit eingeplant, um alles umzusetzen. Ich musste Kompromisse eingehen und eine bestimmte Funktionalität nicht implementieren.

Beim Kontrollieren war ich zufrieden, alles was implementiert wurde hat funktioniert mit wenigen kleinen Fehlern.

Rückblickend habe ich die Dokumentation eindeutig unterschätzt, es musste doch mehr Zeit in die Dokumentation investiert werden als erwartet, wie man es am Zeitplan auch erkennen kann. Leider hat das auch zu Zeitproblemen geführt.

Für das nächste Projekt nimm ich mit, dass ich mehr Zeit in das Dokumentieren einplanen muss, da ich diesen Punkt doch recht unterschätzt habe. Dazu würde ich mehr acht auf die Planung geben den das hat auch dazu geführt, dass ich ein Feature auslassen musste.

Insgesamt bin ich zufrieden mit der IPA und mit dem Resultat. Das Spiel spielt sich gut und macht Spass, das Backend funktioniert zum grössten Teil und ich hoffe es bald mal auf den Google Playstore hochzuladen.

3 Glossar

Begriff	Erklärung
Unity	Ein Tool zum Entwickeln von Spielen oder auch eine Game Engine genannt. In diesem Projekt wird vor allem in Unity gearbeitet, um das Spiel zu entwickeln.
Blender	Blender ist ein 3D-Computergrafik-Software-Toolset, dass in unserem Fall zum Erstellen von 3D Modellen verwendet wird.
Szene	Ein 3D Raum welcher GameObjects enthält.
GameObject	Basisklasse für alle GameObjects (3D Objekte in der Szene) in Unity-Szenen.
Komponente	Basisklasse für alles, was an GameObjects angehängt ist und angehängt wird.
Rigidbody	Rigidbody ist eine Komponente. Durch das Hinzufügen einer Rigidbody-Komponente zu einem Objekt wird seine Bewegung unter die Kontrolle der Physik-Engine von Unity gestellt. Auch ohne Hinzufügen von Code wird ein Rigidbody-Objekt durch die Schwerkraft nach unten gezogen und reagiert auf Kollisionen mit ankommenden Objekten, wenn auch die richtige Collider-Komponente vorhanden ist.
Collider	Collider-Komponenten definieren die Form und Grösse eines Objekts für physikalische Kollisionen. Man benutzt verschiedene Collider basierend auf Form (Box Collider, Sphere Collider…). Dabei ist wichtig zu erwähnen das ein Mesh Collider die genaue Form des Meshes vom GameObjekt annimmt an dem es angeschlossen ist.
Mesh	Die Grafische Darstellung des Objektes. Ein 3D Model.
OnCollisionEnter	Eine Unity funktion die aufgerufen wird wenn das Objekt auf dem dieses
Tag	Ein Tag ist ein Referenzwort, das man einem oder mehreren GameObjects zuweisen kann.
Unity Button	Ein UI Objekt von Unity welches beim drücken einen Event ausführen kann.
PlayerPrefs	„PlayerPrefs“ ist eine Klasse, die Spielereinstellungen zwischen Spielsitzungen speichert. Es kann String-, Float- und Integer-Werte in der Plattformregistrierung des Benutzers speichern.
IEnumerator	IEnumerator startet eine Coroutine. Eine Coroutine ist eine Funktion, die ihre Ausführung aussetzen kann, bis die angegebene

	YieldInstruction beendet ist. Ähnlich zu einer Funktion mit Wait in anderen Sprachen.
Lerp	Die grundlegende Operation der linearen Interpolation zwischen zwei Werten wird üblicherweise in der Computergrafik verwendet.
Vektor 3	Diese Struktur wird in Unity verwendet, um 3D-Positionen und Richtungen herumzureichen. Es enthält auch Funktionen zum Ausführen allgemeiner Vektoroperationen.
Vektor3.Lerp()	Wird benutzt, um auf allen 3 Vektoren eine Lineare Interpolation zu erreichen.

4 Abbildungsverzeichnis

5 Quellenverzeichnis

https://www.pkorg.ch/

https://github.com/ArtemKaegi/IPA-2022-Endless-Runner

https://de.wikipedia.org/wiki/Modell_der_vollst%C3%A4ndigen_Handlung

https://www.youtube.com/c/Brackeys

https://www.youtube.com/watch?v=FlE8e1JwVzs

https://www.youtube.com/watch?v=zc8ac_qUXQY

https://www.youtube.com/watch?v=LgMO7hRQRQs

https://docs.unity3d.com/Manual/index.html

https://docs.unity3d.com/ScriptReference/MonoBehaviour.StartCoroutine.html

https://docs.unity3d.com/ScriptReference/Networking.UnityWebRequest.html

https://answers.unity.com/questions/1519382/textures-and-materials-turn-pink-after-installing.html

https://flask.palletsprojects.com/en/2.0.x/

https://github.com/Bekwnn/UnityJsonHelper/blob/master/JsonHelper.cs

6 Anhang

6.1 Code C#

6.1.1 GameManager.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using TMPro;
using UnityEditor;
using UnityEngine;
using Random = UnityEngine.Random;

public class GameManager : MonoBehaviour
{
    #region misc

    private TextMeshProUGUI coinText;
    private Transform lastExit;
    private GameObject player;
    private TextMeshProUGUI scoreText;
    private float defaultMaxSpeed = 20;
    private bool GameStarted = false;
    private int score;
    private BackendConnector connector;

    #endregion

    #region Menu

    private GameObject menu;
    private GameObject skinSelector;
    private GameObject gameOverUi;
    private GameObject gameUi;
    private GameObject newPlayerUi;
    private GameObject highscoreUi;
    private GameObject highscoreUiContent;
    [SerializeField] private TextMeshProUGUI playerNameText;

    #endregion

    #region Camera

    private CameraController camera;
    private GameObject cameraDeathPosition;
    [SerializeField] private float cameraMovementSpeed;
    [SerializeField] private float cameraRotationSpeed;
    [SerializeField] private GameObject cameraDefaultPosition;
    [SerializeField] private GameObject cameraPlayerPosition;
    [SerializeField] private GameObject cameraSkinsPosition;

    #endregion

    #region Tiles

    private float currentTileGenerationZ;

    [Serializable]
    public struct Tile
    {
        public float length;
        public GameObject tileObject;
    }
```

```
    [SerializeField] private Tile[] tiles;

    private ArrayList generatedTiles = new ArrayList();

    #endregion

    #region Obstacles

    private float currentObstacleGenerationZ;

    [Serializable]
    public struct Obstacle
    {
        public float length;
        public GameObject obstacleObject;
        public Transform[] exits;
    }

    [SerializeField] private Obstacle[] obstacles;
    [SerializeField] private float firstObstacleDistance;
    [SerializeField] private int lastObstacle;

    private ArrayList generatedObstacles = new ArrayList();

    #endregion

    #region Coins

    private ArrayList generatedCoins = new ArrayList();
    [SerializeField] private GameObject coin;
    [SerializeField] private float coinDistance;
    [SerializeField] private int currentCoins;
    private int allCoins;

    #endregion

    #region Skins

    [SerializeField] private SkinsController skinsController;
    [SerializeField] private GameObject defaultSkin;
    public GameObject skin;

    #endregion

    private void Start()
    {
        allCoins = PlayerPrefs.GetInt("Coins");
        player = GameObject.FindWithTag("Player");
        menu = GameObject.FindWithTag("Menu");
        skinSelector = GameObject.FindWithTag("SkinSelector");
        gameOverUi = GameObject.FindWithTag("GameOver");
        gameUi = GameObject.FindWithTag("GameUI");
        newPlayerUi = GameObject.FindWithTag("NewPlayer");
        highscoreUi = GameObject.FindWithTag("Highscore");
        highscoreUiContent = GameObject.FindWithTag("HighscoreContent");
        coinText = GameObject.FindWithTag("coinText").GetComponent<TextMeshProUGUI>();
        camera =
GameObject.FindGameObjectWithTag("MainCamera").GetComponent<CameraController>();
        camera.currentCameraPosition = cameraDefaultPosition;
        camera.cameraMovementSpeed = cameraMovementSpeed;
        camera.cameraRotationSpeed = cameraRotationSpeed;
        player.SetActive(false);
        skinSelector.SetActive(false);
        gameOverUi.SetActive(false);
        gameUi.SetActive(false);
        highscoreUi.SetActive(false);
        scoreText = gameUi.GetComponentInChildren<TextMeshProUGUI>();
        connector =
GameObject.FindWithTag("BackendConnector").GetComponent<BackendConnector>();
        if (PlayerPrefs.GetString("PlayerName") == "")
        {
            menu.SetActive(false);
        }
        else
        {
```

```
            newPlayerUi.SetActive(false);
        }
    }

    #region Backend

    public void CreateNewPlayerBackend()
    {
        string playerName = playerNameText.text;
        string deviceId = SystemInfo.deviceUniqueIdentifier;

        PlayerPrefs.SetString("PlayerName", playerName);

        connector.AddPlayer(playerName, deviceId);
    }

    public void SetNewHighscore(int highscore)
    {
        string deviceId = SystemInfo.deviceUniqueIdentifier;

        connector.setHighscore(deviceId, highscore.ToString());
    }

    #endregion

    #region Game Control

    public void StartGame()
    {
        currentObstacleGenerationZ = firstObstacleDistance;
        menu.SetActive(false);
        player.GetComponent<PlayerController>().selectedSkin = skin != null ? skin :
defaultSkin;
        player.SetActive(true);
        camera.currentCameraPosition = cameraPlayerPosition;
        GenerateStart();
        GameStarted = true;
        gameUi.SetActive(true);
    }

    private void GenerateStart()
    {
        GenerateRandomTile();
        GenerateRandomTile();
        GenerateRandomTile();
        GenerateRandomTile();
        GenerateRandomTile();
        GenerateRandomTile();
        GenerateRandomTile();
        GenerateRandomTile();
        GenerateRandomTile();
        GenerateRandomTile();
        GenerateRandomTile();
        GenerateRandomTile();
        GenerateRandomTile();
        GenerateRandomTile();
        GenerateRandomTile();
        GenerateRandomTile();
        GenerateRandomTile();
        GenerateRandomTile();
        GenerateRandomTile();
        GenerateRandomTile();
        GenerateRandomTile();
        GenerateRandomTile();
        GenerateRandomTile();
        GenerateRandomObstacle();
        GenerateRandomObstacle();
        GenerateRandomObstacle();
        GenerateRandomObstacle();
        GenerateRandomObstacle();
        GenerateRandomObstacle();
        GenerateRandomObstacle();
        GenerateRandomObstacle();
        GenerateRandomObstacle();
        GenerateRandomObstacle();
        GenerateRandomObstacle();
        GenerateRandomObstacle();
        GenerateRandomObstacle();
        GenerateRandomObstacle();
        GenerateRandomObstacle();
        GenerateRandomObstacle();
        GenerateRandomObstacle();
```

```
        GenerateRandomObstacle();
        GenerateRandomObstacle();
        GenerateRandomObstacle();
    }

    public void GameEnd()
    {
        allCoins += currentCoins;
        PlayerPrefs.SetInt("Coins", allCoins);
        Destroy(cameraDeathPosition);
        cameraDeathPosition = new GameObject();
        var dathPos = cameraDeathPosition.transform.position;
        dathPos = camera.currentCameraPosition.transform.position;
        cameraDeathPosition.transform.rotation =
camera.currentCameraPosition.transform.rotation;
        dathPos = new Vector3(dathPos.x, dathPos.y, dathPos.z - 5);
        cameraDeathPosition.transform.position = dathPos;
        camera.currentCameraPosition = cameraDeathPosition;
        gameOverUi.SetActive(true);
        coinText.text = currentCoins.ToString();
        if (GameStarted)
        {
            camera.Shake(-1, 0.3f);
        }
        SetNewHighscore(score);
    }

    private void ResetGame()
    {
        currentCoins = 0;
        player.SetActive(true);
        player.transform.position = player.GetComponent<PlayerController>().startPosition;
        player.GetComponent<PlayerController>().maxSpeed = defaultMaxSpeed;
        player.SetActive(false);
        gameOverUi.SetActive(false);
        gameUi.SetActive(false);
        foreach (GameObject x in generatedTiles)
        {
            Destroy(x);
        }

        foreach (GameObject x in generatedObstacles)
        {
            Destroy(x);
        }

        foreach (GameObject x in generatedCoins)
        {
            Destroy(x);
        }

        generatedTiles = new ArrayList();
        generatedObstacles = new ArrayList();
        generatedCoins = new ArrayList();
        currentObstacleGenerationZ = 0;
        currentTileGenerationZ = 0;
        score = 0;
        scoreText.text = score.ToString();
        GameStarted = false;
    }

    public void Restart()
    {
        ResetGame();
        StartGame();
    }

    #endregion

    #region Level Generation

    public void GenerateRandomTile()
    {
        int i = Random.Range(0, tiles.Length);
        GenerateTile(i);
    }
```

```
    private void GenerateTile(int i)
    {
        GameObject x = Instantiate(tiles[i].tileObject,
            transform.position + Vector3.forward * currentTileGenerationZ, transform.rotation,
            null);
        generatedTiles.Add(x);
        currentTileGenerationZ += tiles[i].length;
    }

    public void GenerateRandomObstacle()
    {
        int i = Random.Range(0, obstacles.Length);
        GenerateObstacle(i);
    }

    public void GenerateObstacle(int i)
    {
        GameObject x = Instantiate(obstacles[i].obstacleObject,
            transform.position + Vector3.forward * currentObstacleGenerationZ,
transform.rotation,
            null);
        generatedObstacles.Add(x);

        GenerateCoins(i);

        currentObstacleGenerationZ += obstacles[i].length;
        if (GameStarted)
        {
            player.GetComponent<PlayerController>().maxSpeed += 0.1f;

            score += 1;
            scoreText.text = score.ToString();
        }
    }

    private void GenerateCoins(int currentObstacle)
    {
        int amountOfCoins = (int) (obstacles[currentObstacle].length / coinDistance);
        if (generatedObstacles.Count > 2)
        {
            if (lastExit == null)
            {
                lastExit = obstacles[lastObstacle].exits[Random.Range(0,
obstacles[lastObstacle].exits.Length)];
            }

            Transform currentExit = obstacles[currentObstacle]
                .exits[Random.Range(0, obstacles[currentObstacle].exits.Length)];
            if (1 <= Mathf.Abs(lastExit.position.y - currentExit.position.y))
            {
                for (int currentCoin = 0; currentCoin < amountOfCoins / 2; currentCoin++)
                {
                    generatedCoins.Add(Instantiate(coin,
                        new Vector3(lastExit.position.x, lastExit.position.y,
                            (currentObstacleGenerationZ - obstacles[lastObstacle].length) +
                            coinDistance * currentCoin), transform.rotation, null));
                }

                for (int currentCoin = 0; currentCoin < amountOfCoins / 2; currentCoin++)
                {
                    generatedCoins.Add(Instantiate(coin,
                        new Vector3(currentExit.position.x, currentExit.position.y,
                            (currentObstacleGenerationZ - obstacles[lastObstacle].length) +
                            coinDistance * (currentCoin + amountOfCoins / 2)),
                        transform.rotation, null));
                }
            }
            else
            {
                for (int currentCoin = 0; currentCoin < amountOfCoins / 3; currentCoin++)
                {
                    generatedCoins.Add(Instantiate(coin,
```

```
                        new Vector3(lastExit.position.x, lastExit.position.y,
                            (currentObstacleGenerationZ - obstacles[lastObstacle].length) +
coinDistance * currentCoin),
                        transform.rotation, null));
                }

                if (1 < Mathf.Abs(lastExit.position.x - currentExit.position.x))
                {
                    for (int currentCoin = 0; currentCoin < amountOfCoins / 3; currentCoin++)
                    {
                        generatedCoins.Add(Instantiate(coin,
                            new Vector3(lastExit.position.x, lastExit.position.y,
                                (currentObstacleGenerationZ - obstacles[lastObstacle].length)
+
                                coinDistance * (currentCoin + amountOfCoins / 3)),
transform.rotation, null));
                    }
                }
                else
                {
                    for (int currentCoin = 0; currentCoin < amountOfCoins / 3; currentCoin++)
                    {
                        generatedCoins.Add(Instantiate(coin,
                            new Vector3(lastExit.position.x,
                                player.GetComponent<PlayerController>().startPosition.y +
1.5f,
                                (currentObstacleGenerationZ - obstacles[lastObstacle].length)
+
                                coinDistance * (currentCoin + amountOfCoins / 3)),
transform.rotation, null));
                    }
                }

                for (int currentCoin = 0; currentCoin < amountOfCoins / 3 + 1; currentCoin++)
                {
                    generatedCoins.Add(Instantiate(coin,
                        new Vector3(currentExit.position.x, currentExit.position.y,
                            (currentObstacleGenerationZ - obstacles[lastObstacle].length) +
                            coinDistance * (currentCoin + (amountOfCoins / 3) * 2)),
transform.rotation, null));
                }
            }

            lastExit = currentExit;
        }

        lastObstacle = currentObstacle;
    }

    #endregion

    #region Menu Navigation

    public void ToSkinsFromMenu()
    {
        camera.currentCameraPosition = cameraSkinsPosition;
        menu.SetActive(false);
        skinSelector.SetActive(true);
        skinsController.UpdateCoinsText();
        skinsController.UpdateBuyEquipText();
    }

    public void ToMenuFromSkins()
    {
        camera.currentCameraPosition = cameraDefaultPosition;
        menu.SetActive(true);
        skinSelector.SetActive(false);
    }

    public void ToMenuFromGameOver()
    {
        ResetGame();
        gameOverUi.SetActive(false);
        menu.SetActive(true);
        camera.currentCameraPosition = cameraDefaultPosition;
    }
```

```
    public void ToHighscoreFromMenu()
    {
        highscoreUi.SetActive(true);
        menu.SetActive(false);
        connector.getHighscores(highscoreUiContent);
    }

    public void ToMenuFromHighscore()
    {
        highscoreUi.SetActive(false);
        menu.SetActive(true);
    }
    #endregion

    #region Coin Controll

    public void AddCoins(int amount)
    {
        currentCoins += amount;
    }

    public int GetCoins()
    {
        return allCoins;
    }

    public void RemoveCoins(int coins)
    {
        allCoins -= coins;
        PlayerPrefs.SetInt("Coins", allCoins);
    }

    #endregion
}
```

6.1.2 PlayerController.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEditor;
using UnityEngine;

public class PlayerController : MonoBehaviour
{
    [Serializable]
    public struct positionCluster
    {
        public GameObject[] verticalPositions;
    }

    [SerializeField] private GameManager gm;
    [SerializeField] private GameObject explosionObject;
    [SerializeField] private positionCluster[] horizontalPositions;
    [SerializeField] private Transform playerTransform;
    [SerializeField] private float laneSwitchingSpeed;
    [SerializeField] private float jumpTime;
    [SerializeField] private float duckTime;
    [SerializeField] public float maxSpeed;
    [SerializeField] private float acceleration;
    public Vector3 startPosition;
    private Rigidbody rb;
    public Transform currentTransform;
    public int currentHorizontalPosition = 1;
    public int currentVerticalPosition = 1;
    public GameObject selectedSkin;
    private GameObject currentSkin;
    private Vector3 firstPosition;
    private Vector3 lastPosition;
    private float dragDistance;
    private bool isDucking = false;
    private bool isJumping = false;
    private bool moved = false;
    private float countdown = 0;
```

```
    void Start()
    {
        currentTransform =
horizontalPositions[currentHorizontalPosition].verticalPositions[currentVerticalPosition]
            .transform;
        dragDistance = Screen.height * 3 / 100;
        rb = GetComponent<Rigidbody>();
        startPosition = transform.position;
    }

    private void OnEnable()
    {
        Destroy(currentSkin);
        currentSkin = Instantiate(selectedSkin, playerTransform.position,
playerTransform.rotation, playerTransform);
    }

    void Update()
    {
        if (rb.velocity.magnitude < maxSpeed)
        {
            rb.AddForce(0, 0, acceleration);
        }

        CheckInput();

        playerTransform.position = Vector3.Lerp(playerTransform.position,
currentTransform.position,
            laneSwitchingSpeed * Time.deltaTime);

        if (isJumping || isDucking)
        {
            countdown -= Time.deltaTime;
            if (countdown <= 0)
            {
                isDucking = false;
                isJumping = false;
                currentVerticalPosition = 1;
                currentTransform = horizontalPositions[currentHorizontalPosition]
                    .verticalPositions[currentVerticalPosition].transform;
            }
        }
    }

    void CheckInput()
    {
        if (Input.touchCount >= 1)
        {
            CheckTouch(Input.GetTouch(0));
        }
    }

    void CheckTouch(Touch touch)
    {
        switch (touch.phase)
        {
            case TouchPhase.Began:
                firstPosition = touch.position;
                lastPosition = touch.position;
                break;
            case TouchPhase.Moved:

                if (!moved)
                {
                    lastPosition = touch.position;

                    if (Mathf.Abs(lastPosition.x - firstPosition.x) + 20 > dragDistance ||
                        Mathf.Abs(lastPosition.y - firstPosition.y) > dragDistance)
                    {
                        moved = true;
                        if (Mathf.Abs(lastPosition.x - firstPosition.x) >
Mathf.Abs(lastPosition.y - firstPosition.y))
                        {
                            if ((lastPosition.x > firstPosition.x))
                            {
                                if (currentHorizontalPosition < horizontalPositions.Length -
```

```
1)
                                {
                                    currentHorizontalPosition++;
                                }
                            }
                            else
                            {
                                if (currentHorizontalPosition > 0)
                                {
                                    currentHorizontalPosition--;
                                }
                            }
                        }
                        else
                        {
                            if (lastPosition.y > firstPosition.y)
                            {
                                if (!isJumping && !isDucking)
                                {
                                    StartJump();
                                }
                            }
                            else
                            {
                                if (!isJumping && !isDucking)
                                {
                                    StartDucking();
                                }
                            }
                        }

                        currentTransform = horizontalPositions[currentHorizontalPosition]
                            .verticalPositions[currentVerticalPosition].transform;
                    }
                }

                break;
            case TouchPhase.Ended:
            {
                moved = false;
                break;
            }
        }
    }

    void StartJump()
    {
        isJumping = true;
        countdown = jumpTime;
        currentVerticalPosition = 2;
    }

    void StartDucking()
    {
        isDucking = true;
        countdown = duckTime;
        currentVerticalPosition = 0;
    }

    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("ObstacleTrigger"))
        {
            gm.GenerateRandomObstacle();
        }
        else if (other.CompareTag("TileTrigger"))
        {
            gm.GenerateRandomTile();
        }
        else if (other.gameObject.CompareTag("Coin"))
        {
            other.gameObject.GetComponent<CoinController>().Pop();
            Debug.Log("CoinsAdd");
        }
    }

    private void OnCollisionEnter(Collision collision)
```

```
    {
        if (collision.gameObject.CompareTag("Obstacle"))
        {
            Die();
            Debug.Log(collision.gameObject.transform.position);
        }
    }

    private void Die()
    {
        gm.GameEnd();
        GameObject currentExplosion =
            Instantiate(explosionObject, playerTransform.position, playerTransform.rotation,
null);
        rb.velocity = new Vector3(0, 0, 0);
        Destroy(currentExplosion, 10);
        playerTransform.position =
horizontalPositions[1].verticalPositions[1].transform.position;
        currentTransform = horizontalPositions[1].verticalPositions[1].transform;
        transform.position = startPosition;
        currentHorizontalPosition = 1;
        currentVerticalPosition = 1;
        gameObject.SetActive(false);
    }

}
```

6.1.3 BackendConnector.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEngine;
using UnityEngine.Networking;


public class BackendConnector : MonoBehaviour
{
    private string URI = "http://172.20.128.71:5000";

    public GameObject highscorePanel;

    public void AddPlayer(string playerName, string deviceId)
    {
        StartCoroutine(CreateNewPlayer(playerName, deviceId));
    }

    public void setHighscore(string deviceId, string highscore)
    {
        StartCoroutine(setNewHighscore(deviceId, highscore));
    }

    public void getHighscores(GameObject parentObject)
    {
        StartCoroutine(getHighscoresIE(parentObject));
    }

    private IEnumerator CreateNewPlayer(string playerName, string deviceId)
    {
        using (UnityWebRequest webRequest =
               UnityWebRequest.Get(URI + "/createNewPlayer/" + playerName + "/" + deviceId))
        {
            yield return webRequest.SendWebRequest();


            switch (webRequest.result)
            {
                case UnityWebRequest.Result.ConnectionError:
                case UnityWebRequest.Result.DataProcessingError:
                    Debug.LogError("Error: " + webRequest.error);
                    break;
                case UnityWebRequest.Result.ProtocolError:
                    Debug.LogError("HTTP Error: " + webRequest.error);
                    break;
                case UnityWebRequest.Result.Success:
                    Debug.Log("Received: " + webRequest.downloadHandler.text);
                    break;
            }
        }
    }

    private IEnumerator setNewHighscore(string deviceId, string highscore)
    {
        using (UnityWebRequest webRequest =
               UnityWebRequest.Get(URI + "/setNewHighscore/" + deviceId + "/" + highscore))
        {
            yield return webRequest.SendWebRequest();


            switch (webRequest.result)
            {
                case UnityWebRequest.Result.ConnectionError:
                case UnityWebRequest.Result.DataProcessingError:
                    Debug.LogError("Error: " + webRequest.error);
                    break;
                case UnityWebRequest.Result.ProtocolError:
                    Debug.LogError("HTTP Error: " + webRequest.error);
                    break;
                case UnityWebRequest.Result.Success:
                    Debug.Log("Received: " + webRequest.downloadHandler.text);
                    break;
            }
```

```
        }
    }

    private IEnumerator getHighscoresIE(GameObject parentObject)
    {
        using (UnityWebRequest webRequest =
               UnityWebRequest.Get(URI + "/getHighscores"))
        {
            yield return webRequest.SendWebRequest();


            switch (webRequest.result)
            {
                case UnityWebRequest.Result.ConnectionError:
                case UnityWebRequest.Result.DataProcessingError:
                    Debug.LogError("Error: " + webRequest.error);
                    break;
                case UnityWebRequest.Result.ProtocolError:
                    Debug.LogError("HTTP Error: " + webRequest.error);
                    break;
                case UnityWebRequest.Result.Success:
                    String[] highscoreInfo = webRequest.downloadHandler.text.Split(',');
                    Highscore[] highscores = new Highscore[highscoreInfo.Length / 2];
                    int currentHighscore = 0;
                    for (int i = 0; i < highscoreInfo.Length; i++)
                    {
                        if (i % 2 == 0)
                        {
                            highscoreInfo[i] = highscoreInfo[i].Replace("[", "");
                            highscoreInfo[i] = highscoreInfo[i].Replace("\\u200b", "");
                            highscores[currentHighscore] = new Highscore();
                            highscores[currentHighscore].playerName = highscoreInfo[i];
                        }
                        else
                        {
                            highscoreInfo[i] = highscoreInfo[i].Replace("]", "");
                            Debug.Log(highscoreInfo[i]);
                            highscores[currentHighscore].highscore =
int.Parse(highscoreInfo[i]);
                            currentHighscore++;
                        }
                    }

                    Debug.Log(highscores[0].highscore);
                    foreach (var x in highscores)
                    {
                        GameObject currentPanel = Instantiate(highscorePanel,
parentObject.transform.position,
                            parentObject.transform.rotation, parentObject.transform);
                        GameObject text = currentPanel.transform.GetChild(0).gameObject;
                        text.GetComponent<TextMeshProUGUI>().text  = " " + x.playerName + "
highscore: " + x.highscore;
                    }

                    break;
            }
        }
    }
}
```

6.1.4 SkinsController.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEngine;
using UnityEngine.PlayerLoop;

public class SkinsController : MonoBehaviour
{
    [SerializeField] private SkinObjectController[] skins;
    private int equipedSkin;
    private int currentlyDisplayedSkin = 0;
    private GameManager gm;
    private bool[] bought;
    [SerializeField] private TextMeshProUGUI BuyEquipText;
    [SerializeField] private TextMeshProUGUI CoinsText;

    private void Start()
    {
        gm = GameObject.FindWithTag("GameController").GetComponent<GameManager>();
        equipedSkin = PlayerPrefs.GetInt("EquipedSkin");
        gm.skin = skins[equipedSkin].skin;
        bought = new bool[skins.Length];
        for (int i = 0; i < bought.Length; i++)
        {
            bought[i] = PlayerPrefs.GetString("Skin" + i) == "Bought";
        }

        bought[0] = true;
        Debug.Log(bought[1]);
        UpdateBuyEquipText();
    }

    public void NextSkin()
    {
        if (currentlyDisplayedSkin < skins.Length - 1)
        {
            skins[currentlyDisplayedSkin].gameObject.SetActive(false);
            currentlyDisplayedSkin += 1;
            skins[currentlyDisplayedSkin].gameObject.SetActive(true);
            UpdateBuyEquipText();
        }
    }

    public void PreviousSkin()
    {
        if (currentlyDisplayedSkin > 0)
        {
            skins[currentlyDisplayedSkin].gameObject.SetActive(false);
            currentlyDisplayedSkin -= 1;
            skins[currentlyDisplayedSkin].gameObject.SetActive(true);
            UpdateBuyEquipText();
        }
    }

    public void CheckButton()
    {
        Debug.Log(bought[currentlyDisplayedSkin]);
        if (bought[currentlyDisplayedSkin])
        {
            if (currentlyDisplayedSkin != equipedSkin)
            {
                EquipSkin();
            }
        }
        else
        {
            if (gm.GetCoins() > skins[currentlyDisplayedSkin].price)
            {
                gm.RemoveCoins(skins[currentlyDisplayedSkin].price);
                PlayerPrefs.SetString("Skin" + currentlyDisplayedSkin, "Bought");
                bought[currentlyDisplayedSkin] = true;
                UpdateBuyEquipText();
                UpdateCoinsText();
```

```
            }
        }
    }

    private void EquipSkin()
    {
        equipedSkin = currentlyDisplayedSkin;
        gm.skin = skins[currentlyDisplayedSkin].skin;
        PlayerPrefs.SetInt("EquipedSkin", equipedSkin);
        UpdateBuyEquipText();
    }

    public void UpdateBuyEquipText()
    {
        if (bought[currentlyDisplayedSkin])
        {
            if (currentlyDisplayedSkin == equipedSkin)
            {
                BuyEquipText.text = "Equiped";
            }
            else
            {
                BuyEquipText.text = "Equip";
            }
        }
        else
        {
            BuyEquipText.text = "Buy " + skins[currentlyDisplayedSkin].price;
        }
    }

    public void UpdateCoinsText()
    {
        CoinsText.text = gm.GetCoins().ToString();
    }
}
```

6.1.5 CameraController.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Random = UnityEngine.Random;

public class CameraController : MonoBehaviour
{
    public GameObject currentCameraPosition;
    public float cameraMovementSpeed;
    public float cameraRotationSpeed;
    public float elapsed = 0;
    public float duration = 0.05f;
    public float magnitude = 0;
    private void FixedUpdate()
    {

        transform.position = Vector3.Lerp(transform.position,
currentCameraPosition.transform.position, cameraMovementSpeed * Time.deltaTime);
        transform.rotation = Quaternion.Lerp(transform.rotation,
currentCameraPosition.transform.rotation,
            cameraRotationSpeed * Time.deltaTime);
    }

    private void Update()
    {
        if (elapsed < duration)
        {
            float x = Random.Range(-1f, 1f) * magnitude;
            float y = Random.Range(-1f, 1f) * magnitude;

            transform.position += new Vector3(x, y);

            elapsed += Time.deltaTime;
            Debug.Log("Shaking");

        }
    }

    public void Shake(float elapsed, float magnitude)
    {
        this.elapsed = elapsed;
        this.magnitude = magnitude;
    }

}
```

6.1.6 CoinController.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CoinController : MonoBehaviour
{
    private GameObject coin;
    [SerializeField] private GameObject death;
    private GameManager gm;
    [SerializeField] private int value;

    void Start()
    {
        gm = GameObject.FindWithTag("GameController").GetComponent<GameManager>();
        coin = GetComponentInChildren<Transform>().gameObject;
    }

    void Update()
    {
        coin.transform.Rotate(0, 60 * Time.deltaTime, 0);
    }

    public void Pop()
    {
        gm.AddCoins(value);
        GameObject currentDeath = Instantiate(death, transform.position, transform.rotation,
null);
        Destroy(currentDeath, 2);
        Destroy(gameObject);
    }
}
```

6.1.7 SkinObjectController.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class SkinObjectController : MonoBehaviour
{
    public GameObject skin;
    public int price;

    void Update()
    {
        skin.transform.Rotate(new Vector3(0, 30 * Time.deltaTime, 0));
    }
}
```

6.1.8 Highscore.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

[Serializable]
public class Highscore
{
    public string playerName;
    public int highscore;
}
```

6.2 Code Python

6.2.1 main.py

```
import logging
from flask import Flask
import mysql.connector
import json

app = Flask(__name__)

logging.basicConfig(level=logging.INFO, filename="backen.log", format='%(asctime)s
%(levelname)-8s %(message)s')

try:
    mydb = mysql.connector.connect(
        host="localhost",
        user="gameBackend",
        password="Artom125332?",
        database="endless_runner_backend"
    )

    logging.info("Connection Succesful")
except mysql.connector.Error as err:
    logging.error("Something went wrong: {}".format(err))


@app.route("/createNewPlayer/<playerName>/<deviceId>")
def create_new_player(playerName, deviceId):
    mycursor = mydb.cursor()
    try:
        mycursor.execute("INSERT INTO players(playerName, deviceId) VALUES('" + playerName +
"', '" + deviceId + "')")
        mycursor.close()
        mydb.commit()
        logging.info("Successfully ran: INSERT INTO players(playerName, deviceId) VALUES('" +
playerName + "', '" + deviceId + "')")
        return "Success"
    except mysql.connector.Error as err:
        logging.error("Something went wrong: {}".format(err))
        return "Failure"


@app.route("/setNewHighscore/<deviceId>/<highscore>")
def create_new_highscore(deviceId, highscore):
    try:
        mycursor = mydb.cursor()
        mycursor.execute("SELECT playerId FROM players WHERE deviceId = '" + deviceId + "'")
        x = mycursor.fetchall()
        print("INSERT INTO highscores(playerId, highscore) VALUES(" + str(x[0][0]) + ", " +
highscore + ")")
        mycursor.execute("INSERT INTO highscores(playerId, highscore) VALUES(" + str(x[0][0])
+ ", " + highscore + ")")
        mycursor.close()
        mydb.commit()
        logging.info("Successfully ran: INSERT INTO highscores(playerId, highscore) VALUES(" +
str(x[0][0]) + ", " + highscore + ")")
        return str(x)
    except mysql.connector.Error as err:
        logging.error("Something went wrong: {}".format(err))
        return "Failure"


@app.route("/getHighscores")
def get_highscores():
    try:
        mycursor = mydb.cursor()
        mycursor.execute("SELECT players.playerName, highscores.highscore FROM highscores
INNER JOIN players ON players.playerId = highscores.playerId ORDER BY highscore DESC LIMIT
100")
        result = mycursor.fetchall()
        mycursor.close()
        logging.info("Successfully ran: SELECT * FROM highscores ORDER BY highscore DESC LIMIT
100")
```

```
        return json.dumps(result)
    except mysql.connector.Error as err:
        logging.error("Something went wrong: {}".format(err))
        return "Failure"

@app.route("/skins/<deviceId>/<skins>")
def set_skins(deviceId, skins):
    try:
        mycursor = mydb.cursor()
        mycursor.execute("SELECT playerId FROM players WHERE deviceId = '" + deviceId + "'")
        x = mycursor.fetchall()
        mycursor.execute("INSERT INTO skins(playerId, highscore) VALUES(" + str(x[0][0]) + ",
" + skins + ")")
        result = mycursor.fetchall()
        mycursor.close
        return "success"
    except mysql.connector.Error as err:
        logging.error("Something went wrong: {}".format(err))
        return "Failure"


app.run(host="0.0.0.0")

# print(mycursor.fetchall())
```

6.3 Code Mysql

```
CREATE DATABASE `endless_runner_backend`
 /*!40100 DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci */ /*!80016 DEFAULT ENCRYPTION='N' */;

CREATE TABLE `highscores` (
  `highscoreId` int NOT NULL AUTO_INCREMENT,
  `playerId` int NOT NULL,
  `highscore` int DEFAULT NULL,
  PRIMARY KEY (`highscoreId`,`playerId`),
  UNIQUE KEY `highscoreId_UNIQUE` (`highscoreId`),
  KEY `playerId_idx` (`playerId`),
  CONSTRAINT `playerId` FOREIGN KEY (`playerId`) REFERENCES `players` (`playerId`)
) ENGINE=InnoDB AUTO_INCREMENT=31 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

CREATE TABLE `players` (
  `playerId` int NOT NULL AUTO_INCREMENT,
  `playerName` varchar(45) NOT NULL,
  `deviceId` varchar(45) NOT NULL,
  PRIMARY KEY (`playerId`),
  UNIQUE KEY `playerId_UNIQUE` (`playerId`)
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```