Artem Kägi

Supercomputing Systems AG

Technoparkstrasse 1

CH-8005 Zürich

IPA 2022

"Endless Runner" Spiel für Android

Inhaltsverzeichnis

Inhaltsverzeichnis

[1 Teil 1 2](#_Toc98172396)

[1.1 Vorwort 2](#_Toc98172397)

[1.2 Aufgabenstellung 2](#_Toc98172398)

[1.2.1 App 3](#_Toc98172399)

[1.2.2 Gameplay 3](#_Toc98172400)

[1.2.3 Backend 3](#_Toc98172401)

[1.3 Projektaufbauorganisation 5](#_Toc98172402)

[1.4 Vorkenntnisse 6](#_Toc98172403)

[1.5 Vorarbeit 6](#_Toc98172404)

[1.5.1 In den letzten 6 Monaten 6](#_Toc98172405)

[1.6 Arbeitsprotokoll 7](#_Toc98172406)

[1.6.1 Mo 28.02.2022 7](#_Toc98172407)

[1.6.2 Di 01.03.2022 7](#_Toc98172408)

[1.6.3 Mi 02.03.2022 8](#_Toc98172409)

[1.6.4 Do 03.03.2022 8](#_Toc98172410)

[2 Projekt 10](#_Toc98172411)

[2.1 Zusammenfassung 10](#_Toc98172412)

[2.1.1 Ausgangslage 10](#_Toc98172413)

[2.1.2 Umsetzung 10](#_Toc98172414)

[2.1.3 Ergebnis 10](#_Toc98172415)

[2.2 Informieren 11](#_Toc98172416)

[2.2.1 Ziele 11](#_Toc98172417)

[2.2.2 Vorgaben 11](#_Toc98172418)

[2.3 Planen 12](#_Toc98172419)

[2.3.1 Realisierungskonzept 12](#_Toc98172420)

[2.3.2 Datenmodell 13](#_Toc98172421)

[2.4 Entscheiden 14](#_Toc98172422)

[2.4.1 Varianten 14](#_Toc98172423)

[2.4.2 Entscheid 14](#_Toc98172424)

[2.5 Realisierung 15](#_Toc98172425)

[2.5.1 Projektumgebung 15](#_Toc98172426)

[2.5.2 Bewegung 15](#_Toc98172427)

[3 Glossar 17](#_Toc98172428)

# Teil 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 wird die deteillierte 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.

Durch die ganze Arbeit hinweg wurde die Projektplanungsmethode IPERKA eingesetzt.

## 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.

### 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.

### 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.

### 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.

## Projektaufbauorganisation

**Lehrbetrieb und Durchführungsort:**

Supercomputing Systems AG

Technoparkstrasse 1

8005 Zürich

043 456 16 00

[Info@scs.ch](mailto:Info@scs.ch)

**Kandidat:**

Artem Kägi

Agnesstrasse 8

078 705 64 27

[artem.kaegi@scs.ch](mailto:artem.kaegi@scs.ch)

**Berufsbildner:**

Christian Walder

Benedict Schule

Militärstrasse 106

8004 Zürich

044 298 17 83

[christian.walder@benedict.ch](mailto:christian.walder@benedict.ch)

**Verantwortliche Fachkraft:**

Pascal Iselin

Supercomputing Systems AG

Technoparkstrasse 1

8005 Zürich

079 215 62 37

[pascal.iselin@scs.ch](mailto:pascal.iselin@scs.ch)

**Nebenexpert:**

Rinoy Manavalan

[rinoy.manavalan@gmail.com](mailto:rinoy.manavalan@gmail.com)

078 226 80 19

**Hauptexpert:**

Marko Tepeluk

[martepeluk@gmail.com](mailto:martepeluk@gmail.com)

076 497 58 88

## 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.

## Vorarbeit

Für die IPA wurden folgende Massnahmen vorgenommen:

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

### 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

## Arbeitsprotokoll

Starttermin: 28.02.2022

Abgabedatum: 15.03.2022

|  |  |  |  |
| --- | --- | --- | --- |
| 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. |
| 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. |
| 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. | |
| 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. | | |

# Projekt

## 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.

### 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.

### 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.

### Ergebnis

## 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.

### 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.

### 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.

## 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.

### 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.

1. 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 durch zu kommen.

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.

1. 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.

1. 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.

1. 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.

1. 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.

1. Datenbank

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

1. Backend Schnittstelle

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

1. Unittest backend

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

1. 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.

1. Assets

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

### Datenmodell

Die Datenbank läuft lokal auf dem Arbeitslaptop des Kandidaten.

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

Graphical user interface, text, application

Description automatically generated

Abbildung : Datenbankmodell in MySql Workbench

## 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.

### Varianten

Wie man im Realisierungskonzept gesehen hat, gibt es für einige Ziele, welche über mehrere Lösungsmöglichkeiten verfügen. Zum Beispiel könnte man beim Sterben die ganze Szene neu laden. Man könnte aber auch eine Funktion haben, die alles auf die Anfangsposition zurücksetzen.

Eine weitere Entscheidung währe bei der Bewegung des Spielers. Man könnte den Spieler nur mit einem Rigidbody und Physik bewegen. Man könnte aber auch die Position des Spielers direkt Manipulieren ohne Physik oder sogar eine Hybridlösung anwenden, wobei man die Recht- & Linksbewegung Statisch mit direkter Manipulation durchführt, die Vorwärtsbewegung hingegen mit Physik durchführt.

Als Letztes währe da noch das Menu, welches Viele varianten haben könnte, für unser Projekt aber sehe ich zwei grosse. Entweder wir haben eine eigene Szene für das Menu und das Spiel oder wir haben das Menu mit dem Spiel in einer Szene.

### Entscheid

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

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. Das mach diese Lösung die schwerste zum Umsetzen, aber sie sollte sich durch ein gutes Spielgefühl auszeichnen.

Zu guter Letzt währe 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.

## Realisierung

Das Realisieren ist die vierte Phase der IPERKA-Methode.

In dieser Phase wird vor allem die Programmierarbeit dokumentiert.

### 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.

### 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

# 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. |