Höhere Technische Bundeslehranstalt und Bundesfachschule
im Hermann Fuchs Bundesschulzentrum

Virtual Running Course

Diplomarbeit

Schulautonomer Schwerpunkt

Mobile Computing

ausgeführt im Schuljahr 2021/2022 von:

Tobias Pieber, 5AHELS

Thomas Schinwald, 5AHELS

Betreuer/Betreuerin:

Dipl-Ing. Gerhard Waser

Braunau, am 2. März 2021

Eidesstattliche Erklärung

Wir erklären an Eides statt, dass ich/wir die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als angegebene Quellen und Hilfsmittel nicht direkt benutzt und die benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche erkenntlich gemacht habe/n.

……..…….....
Braunau am Inn, xx.xx.2022

……..…….
Thomas Schinwald

……..…….....
Braunau am Inn, xx.xx.2022

……..…….
Tobias Pieber

Abstract TS

Athletes often perceive indoor sports as lacking compared to their real counterparts. Without goals and variety, there is sometimes not enough motivation for extensive training. There are already many indoor trainers that specialise in this kind of sector. However, they offer only a limited selection of courses. So, the users can only choose from a limited selection. The pricing for such system is usually very high. Our work aims to present a novel solution to these problems.

We propose the indoor training system TreadRun. TreadRun is an indoor trainer for runners. The user can run through selected courses in a virtual world. These courses can be created in locations all over the world. The length as well as the course can be precisely configured. While running, our hardware collects the necessary data from the treadmill. TreadRun makes use of our algorithm to dynamically generate the running courses. This makes a near unlimited selection of courses possible. Due to the dynamic structure, only selected parts of the world need to be calculated at once. This enables the usage of TreadRun on a wide range of end devices.

We describe our process of development in three sections. The first section explains the use of hardware. We present our technique of reading data from the treadmill. The second section shows the process of dynamically generating a virtual world. Our way of map data collection as well as world building is presented in detail. The third section describes the general game design of TreadRun. This includes user experiences in visuals as well as interactivity. TreadRun consists of a multitude of components, which we describe in the following work.

Kurzfassung TS

Indoor-Sport wird von Sportlern oft als monoton empfunden. Ohne Zielsetzung und Abwechslung fehlt oft die nötige Motivation für ein ausgiebiges Training. Es gibt bereits eine Vielzahl von Indoor-Trainern, die sich auf das Lauftraining spezialisieren. Jedoch bieten diese nur eine begrenzte Kursauswahl. Somit sind die Auswahlmöglichkeiten der Nutzer stark beschränkt. Der Preis von Systemen für das gezielte Indoor-Training ist oft sehr hoch. Ziel unserer Arbeit ist, diese Probleme zu lösen.

Wir stellen das Indoor-Training System TreadRun vor. TreadRun ist ein Lauftrainer, der den Nutzer durch gewählte Kurse in einer virtuellen Welt laufen lässt. Der Nutzer kann dabei Kurse auf der ganzen Welt erstellen. Die Länge sowie der genaue Verlauf dieser unterliegt vollständig der Auswahl des Nutzers. Die nötigen Laufdaten werden dabei mit unserer Hardware vom Laufband abgelesen. TreadRun nutzt einen Algorithmus, der eine völlig dynamische Generation von Laufkursen ermöglicht. Somit gibt es für den Nutzer unbegrenzte Auswahlmöglichkeiten. Durch den dynamischen Aufbau muss immer nur ein kleiner Teil der generierten Welt berechnet werden, was den Betrieb auf einer Vielzahl von Endgeräten ermöglicht.

Wir beschreiben unseren Entwicklungsprozess in drei Abschnitten. Der erste Abschnitt erklärt die Verwendung von Hardware. Wir präsentieren unsere Technik zur Datenablesung vom Laufband. Der zweite Abschnitt zeigt den Prozess zur dynamischen Generation einer virtuellen Welt. Die Art zur Gewinnung der nötigen Daten sowie der Ablauf des Aufbaus wird im Detail vorgestellt. Der dritte Abschnitt beschreibt das allgemeine Gamedesign von TreadRun. Dieser beinhaltet Nutzererfahrungen in Optik sowie Interaktivität. TreadRun ist ein Aufbau aus einer Vielzahl von Komponenten, welche wir in der folgenden Arbeit beschreiben.

Inhaltsverzeichnis

1 Einleitung TS

Eine Software, die dynamisch Trainingskurse aus der echten Welt nachbaut in Kombination mit Hardware, die Nutzerdaten ausliest – diese Elemente bilden das Grundgerüst des Indoor-Trainers TreadRun. Von der Beschaffung der nötigen Daten bis hin zur Nutzerfreundlichkeit ergeben sich einige Herausforderungen. Wir untersuchen Möglichkeiten zur Umsetzung dieses Vorhabens, erläutern Probleme und präsentieren schlussendlich die Voraussetzungen für eine innovative und neuartige Lösung. Dieser Abschnitt gibt auch einen grundsätzlichen Einblick in die von uns gewählte Entwicklungsumgebung sowie unseren Entwicklungsablauf.

1.1 Problemstellung TS

Viele Sportlerinnen und Sportler betreiben regelmäßig Indoor-Sport. Dabei wird meist auf verschiedenste Unterhaltungsmöglichkeiten zurückgegriffen, da Indoor-Sport nicht nur ansatzweise dasselbe Erlebnis wie Outdoor-Sport bietet. Genau hier liegt das Einsatzgebiet von TreadRun. Ein Indoor-Trainer, der den Nutzerinnen und Nutzer ein interaktives und motivierendes Sporterlebnis bietet.

In einem typischen Anwendungsfall benötigt der Nutzer ein Laufband, einen Bildschirm sowie einen Computer, der die von uns angegebenen Mindestanforderungen erfüllt. Beim Start von TreadRun verbindet sich die Software mit der Hardware, die der Nutzer zuvor am Laufband installiert hat. Der Nutzer wählt über eine Karte seinen gewünschten Kurs aus. Kurz darauf wird dieser Kurs in einer virtuellen Welt nachgebaut. In dieser virtuellen Welt erscheint daraufhin ein Charakter, der den Nutzer repräsentiert. Der Nutzer kann diesen Charakter nun mit dem Laufband steuern und so den nachgebildeten Kurs ablaufen. Wissenswerte Daten wie die aktuelle Geschwindigkeit sind dabei stets sichtbar.

Unser Ziel ist, diesen Indoor-Trainer für so viele Sportlerinnen und Sportler wie möglich zugänglich zu machen. Daher liegt ein besonderer Fokus auf Benutzerfreundlichkeit und Kompatibilität.

1.2 Problembeschreibung *TS*

Da TreadRun sowohl aus einem Hardwareteil als auch eine Softwareteil besteht, ergibt sich eine weite Spanne von Problemen. Die Hardware misst mittels eines Sensors die Geschwindigkeit des Sportlers und reicht die aktuellen Daten an die Software weiter. Um eine virtuelle Welt nachzubauen, werden entsprechende Daten benötigt. Wir beziehen hierbei unsere Daten aus der OpenStreetMap. Mithilfe dieser Kartendaten kann TreadRun Kurse auf der ganzen Welt nachbauen.

Unser Ziel ist Kurse so zu generieren, dass diese möglichst immersiv und gleichzeitig leistungseffizient sind. Um eine Kompatibilität mit möglichst vielen Endgeräten zu ermöglichen, müssen viele Faktoren beachtet werden. Darunter sind Ladezeiten, Wettersituationen, Charakter-Animationen und Landschaftskomplexität. Viele grundlegende Eigenschaften einer virtuellen Welt müssen auch bedacht werden.

1.3 Lösung *TS*

Wir zogen für die Entwicklung von TreadRun verschiedenste Lösungen in Betracht. Die Entwicklungsumgebung Unity bietet uns hierbei die nötige Grundlage. Hier zeigen wir unsere grundlegenden Lösungsansätze, welche in den späteren Kapiteln genauer beschrieben werden.

1.3.1 Kartendaten *TP*

Die benötigten Kartendaten werden von OpenStreetMap bezogen. OpenStreetMap ist ein Open Source Projekt, das von Contributern geführt wird. Diese werden über die Overpass-API als JSON Objekt übergeben und können ausgewertet werden. Overpass benutzt eine eigene Query-Sprache, um Daten effizient auslesen zu können.

1.3.2 Dynamische Generierung TP

Die Generierung basiert auf der Strecke, die vom Benutzer eingegeben wurde und den Features die Unterstützt werden. Die Laufstrecke wird in kleinere Abschnitte aufgeteilt, und zur Laufzeit werden Kartendaten heruntergeladen. Aus diesen Daten wird eine 3-dimensionale Welt erstellt.

1.3.3 Gamedesign TS

GUI

Wir benötigen mehrere User-Interfaces, um dem Benutzer eine intuitive Bedienung von TreadRun zu ermöglichen. Uns ist wichtig, dem Nutzer ein Hauptmenü, einen Ladescreen, eine Streckenauswahl, ein Pausenmenü und einen Endscreen zu bieten. Zum Aufbau dieser Benutzeroberflächen nutzen wir zum einem die in Unity enthaltenen Vorlagen und zum anderen die von uns entworfenen UI-Elementen. Eigene UI-Elemente erstellen wir im Bildbearbeitungsprogramm Gimp sowie Photoshop. Für Abwechslung in den verschiedenen Menüs sorgen wechselnde Hintergrundbilder. Diese werden in Clip Studio Paint gezeichnet. Auch das Feedback von Button und anderen Elementen wird dabei beachtet.

Kamera

Während der Nutzer den gewählten Kurs abläuft, muss dieser stets seinen Charakter sehen. Dazu muss dem Character ständig eine sogenannte virtuelle Kamera folgen. Dazu nehmen wir uns die Cinemachine zur Hilfe. Cinemachine ist eine Unity Komponente, die uns erlaubt, Kameraeinstellungen vorzunehmen, ohne komplexe Programme dazu schreiben zu müssen. Wir konfigurieren die Höhe und Blickwinkel der Kamera und lassen sie dem Charakter des Nutzers als Ziel verfolgen. Dabei dürfen jedoch einige Dinge nicht außer Acht gelassen werden. Zum Beispiel muss bedacht werden, wie schnell die Kamera rotiert, wenn der Charakter die Richtung wechselt. Außerdem muss die Kamerareaktion, wenn sie mit einem Gebäude kollidiert, geregelt werden.

Shader

Da es uns nicht möglich ist, die Kurse in TreadRun in einem hochrealistischen Stil nachzubauen, verwenden wir einen Cartoon ähnlichen Stil. Dadurch können wir fehlende Gebäudedaten besser verschleiern und eine klare Performance Verbesserung erzielen. Zum Erreichen dieser grafischen Effekte verwenden wir sogenannte Cel-Shader. Diese programmieren wir mithilfe von Shadergraph und HLSL. Dabei werden verwenden wir zwei Version des Shaders. Einen Shader mit Textur-Input und einen mit Farb-Input.

Animation

Der Charakter des Nutzers muss sich beim Laufen bewegen, dazu brauchen wir ein Charaktermodell sowie entsprechende Animationen. Wir entscheiden uns für die Nutzung von Modellen von Mixamo. Wir verwenden eine Laufanimation sowie mehrere Idle-Animationen, falls der Nutzer nicht mehr läuft. Die verschiedenen Animationen werden dabei als States gesetzt und programmiert. Die Geschwindigkeit der Laufanimation muss hierbei ständig angepasst werden, um mit der Laufgeschwindigkeit synchron zu sein.

Wetter

Wir verwenden ein dynamisches Wettersystem, um dem Nutzer ein immersives Lauferlebnis zu bieten. Hierzu verwenden wir das Unistorm Weather System. Dies bietet uns die Möglichkeit, viele Wettereigenschaften einfach zu bestimmen. Zum Beispiel können Regen, Hagel und Nebeldichte festgelegt werden. Auch die Änderung der Tageszeit ist möglich.

1.4 Struktur TS

Diese Arbeit wird folgendermaßen strukturiert: Kapitel 2 beschreibt die wichtigsten Grundlagen der verwendeten Technologie. Darauf folgen die vier Hauptkapitel, die sich mit Umsetzung von TreadRun im Detail beschäftigen.

Kapitel 3 zeigt unsere Verwendung der Hardware. Wir zeigen, wie die Messung von Daten, Kalibration der Hardware sowie Kommunikation mit dem Programm abläuft. Auch die Verarbeitung bzw. Umrechnung der Daten wird beschrieben.

Kapitel 4 führt unseren Vorgang der Kursgeneration sowie die dabei verwendeten APIs im Detail an. Detaillierte Beschreibungen der APIs sollen einen Überblick über dessen Funktionen geben. Der Vorgang zur Generierung der virtuellen Welt wird Schritt für Schritt erklärt.

Kapitel 5 beschreibt das Game Design im Detail. Es wird der grundlegende Aufbau von TreadRun bis hin zu speziellen Anpassungen der Nutzererfahrung gezeigt. Charakter Animationen, User-Interfaces, Shader und Kamera-Ansichten sind essenzielle Teile des Game Designs von TreadRun.

Schlussendlich präsentieren wir einen vollständigen Anwendungsfall und veranschaulichen wie unsere Lösung funktioniert. Darauf folgt eine Evaluierung unserer Projektplanung.

Den Schluss unserer Arbeit bildet eine Auseinandersetzung mit potenziellen, zukünftigen Erweiterungen dieser Arbeit.

2 Verwendete Technologien

Wir geben in diesem Abschnitt einen Einblick in die verwendete Technologie in Software als auch Hardware.

2.1 Software

2.1.1 Entwicklungsumgebung TS

Eine Game-Engine bildet das Grundgerüst für die Entwicklung von Videospielen. Sie machen es möglich, verschiedenste Komponenten wie Animationen bis hin zu User-Interfaces einfach einzubinden und sofort in Echtzeit zu testen. Rendern von Grafiken, Speicherverwaltung, Kollisionserkennung und vieles mehr fallen in den Zuständigkeitsbereich der Engine. Grundsätzlich besteht eine Engine aus einer Rendering-Engine, Audio-Engine, Physik-Engine und dem Programm, das die Programmlogik enthält.

Wir entscheiden uns bei der Entwicklung von TreadRun für die Entwicklungsumgebung Unity. Durch die große Anzahl an Online-Ressourcen und Benutzerfreundlichkeit ist Unity eine sehr einsteigerfreundliche Engine. Unity bietet die Möglichkeit, für verschiedene Plattformen zu entwickeln und besitzt außerdem eine große Anzahl von kostenlosen Online-Inhalten, die sich für die Nutzung im Projekt anbieten. Sowohl wegen der genannten Eigenschaften als auch wegen vorheriger Erfahrungen ist Unity die Entwicklungsumgebung unserer Wahl für TreadRun.

2.1.2 Shader TS

3D-Grafiken verwenden eine Vielzahl von Informationen, um das Bild korrekt darzustellen: Texturen, Lichter usw. Diese Informationen werden von der Grafikhardware verarbeitet und dann an ein Wiedergabegerät weitergeleitet.

Der Ablauf der Bilderzeugung wird Rendering genannt. Dabei sind Shader Programme, die die gegebenen Daten modifizieren. Somit lassen sich mit Shadern Bildmanipulationen aller Art durchführen, was uns bei der Entwicklung von TreadRun hilft.

2.1.2.1 Shadergraph

Der Shadergraph ist der visuelle Shader-Editor von Unity. Anstatt Code in HLSL zu schreiben, besteht hier die Möglichkeit, mit vorgefertigten Codeblöcken zu arbeiten. Dies bietet einige Vorteile. Zum einen ist der Shadergraph einsteigerfreundlicher, zum anderen ist es übersichtlicher, da das Schreiben von Standardcode wegfällt. Für uns ist jedoch der größte Vorteil, dass es im Shadergraph Vorschaufenster gibt, die aktuelle Änderungen anzeigen. Dadurch wird für uns eine schnellere Entwicklung der benötigten Shader möglich.

2.1.2.2 HLSL

Doch auch der Shadergraph kommt nicht ganz ohne Code aus. In manchen Situationen ist es Vorteilhaft mit Code zu arbeiten. Hierbei kommt in Unity die Shader-Programmiersprache HLSL zum Einsatz.

```
1   void MainLight_float(float3 WorldPos, out float3 Direction, out float3 Color,
2       out float DistanceAtten, out float ShadowAtten)
3   {
4   #ifdef SHADERGRAPH_PREVIEW
5       Direction = normalize(float3(0.5f, 0.5f, 0.25f));
6       Color = float3(1.0f, 1.0f, 1.0f);
7       DistanceAtten = 1.0f;
8       ShadowAtten = 1.0f;
```

Code 1: HLSL Syntax

HLSL ist die Abkürzung für High Level Shader Language. Diese Sprache wird verwendet, um Shader zu schreiben. Ihre Syntax hat Ähnlichkeiten zu C, jedoch benutzt sie eine eigene Programmstruktur und Datentypen.

2.1.2.3 Grafik API

In Verbindung mit dem Shader sollte auch kurz geklärt werden, was eine Grafik API ist. Die Grafik API ist grundsätzlich für die Kommunikation zwischen GPU und Programmen verantwortlich. Sie ist eine Pipeline von mehreren Schritten, die zur Erzeugung des finalen Bilds dienen. Mit Shader können einige dieser Schritte manipuliert werden und somit die gewollten grafischen Effekte erzeugt werden. Mittlerweile gibt es verschiedene moderne Grafik

APIs. Dabei unterstützt Unity DirectX, Metal, OpenGL und Vulkan. Hier gibt es die Möglichkeit, je nach Plattform, Unity selbst die ideale Grafik API wählen zu lassen.

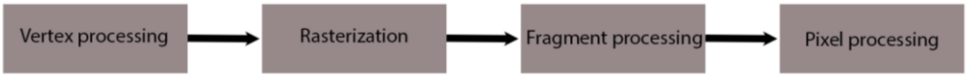

Abbildung 1: Grafik API Pipeline

2.1.3 Bildbearbeitung TS

Für die Gestaltung der UI entscheiden wir uns, Teile dieser selbst zu erstellen. Dazu verwenden wir hauptsächlich das Bildbearbeitungsprogramm Gimp. Außerdem verwenden wir Clip Art Studio, um UI-Hintergründe sowie Menü Details zu zeichnen.

2.1.4 Scripting in Unity TS

Uns stehen in Unity für viele Standardaufgaben schon fertige Lösungen zur Verfügung, doch die eigentliche Logik des Programms muss von uns programmiert werden. Dazu schreiben wir in Unity Skripte mit der Programmiersprache C#. Skripte verhalten sich in Unity für gewöhnlich wie Komponenten und werden daher an Gameobjects angehängt.

Unity nützt die Programmiersprache C# in Kombination mit dem Mono-Framework. Dies ist eine Open-Source-Variante des .NET Frameworks, die es ermöglicht C#-Anwendungen auch auf anderen Systemen zu betreiben.

2.1.5 OpenStreetMap TP

OpenStreetMap ist eine Weltkarte, welche unter einer offenen Lizenz kostenlos verwendet werden kann. Alle Daten, die diese Karte ausmachen, werden von sogenannten Contributern bereitgestellt, die in vielen Fällen ihre eigene Nachbarschaft kartografieren. Dadurch werden die Datensätze immer genauer, da lokales Wissen eingesetzt wird.

Jedes Element der Karte besteht aus sogenannten Features. Features bestehen aus mehreren Nodes, die ihren Längen- und Breitengrad gespeichert haben.

Overpass API

Overpass ist die Schnittstelle zwischen OpenStreetMap Daten und einem Programm. Mit einer eigenen Query-Sprache kann jedes Element aus den Datensätzen abgefragt werden.

2.1.6 Strava TP

Strava ist ein Social Media Netzwerk für Athleten, auf welches man Aktivitäten hochladen kann. Mit Stand 2021 hatte 76 Millionen Benutzer. Der Feed wird durch die Freunde eines jeden Benutzer erstellt und man kann sich untereinander austauschen.

Strava bietet eine öffentliche API für das Hochladen von Aktivitäten, Abrufen der Nutzerdaten und mehr.

2.1.7 Discord TP

Discord ist eine App für Nachrichten, Chats, und Videoanrufe. Es ist vor allem bei Spielern sehr beliebt, da Discord auf dem PC installiert werden kann und für fast jedes Spiel eine Integrierung besitzt. Die Discord Rich Presence zeigt anderen Leuten an, was der Benutzer gerade spielt. Diese Rich Presence ist für jedes Spiel vom Entwickler frei einstellbar.

Discord bietet eine Integrierung ihrer SKD für Unity an.

2.2 Hardware *TP*

2.2.1 Raspberry Pi

Der Raspberry Pi ist ein Einplatinencomputer, der dazu entwickelt wurde, um den Einstieg in die Computerprogrammierung zu erleichtern.

Das gängigste Betriebssystem ist Debian, ein auf Linux basierendes Betriebssystem.

Der Raspberry Pi besitzt GPIOs die die Schnittstelle zwischen CPU und externer Beschaltung dienen. Mit Python kann man diese sehr einfach Ansteuern und Auslesen. In den neuesten Modellen ist auch Bluetooth und Wifi verbaut.

Der Raspberry Pi Zero W ist eine kleinere Version des Raspberry Pi, steht diesem aber in nichts nach.

Abbildung 2: Raspberry Pi Zero W

3 Hardware TP

Die Hardware besteht aus einem Raspberry Pi Zero W, einem Fotowiderstand und einem Gyrosensor.

Der Raspberry Pi Zero W wurde ausgewählt, da dieser neben WiFi auch ein Bluetooth-Modul besitzt und zudem kleiner als ein herkömmlicher Raspberry Pi ist.

Bei der Programmiersprache hat man sich auf C# geeinigt, da .NET Core mit Linux kompatibel ist. Allerdings wird .NET Core auf dem Raspberry Pi Zero W nicht unterstützt deshalb wurde auf das Mono-Framework umgestellt. Das Mono-Framework ist eine Open-Source-Variante des .NET Frameworks, die es ermöglicht, C#-Anwendungen auch auf anderen Systemen zu betreiben. Unteranderem auch auf Linux.

Die Hardware wird zum Auslesen der Geschwindigkeit eines Laufbands genutzt. Diese Geschwindigkeit wird dann an das Programm gesendet, wo es weiterverarbeitet wird.

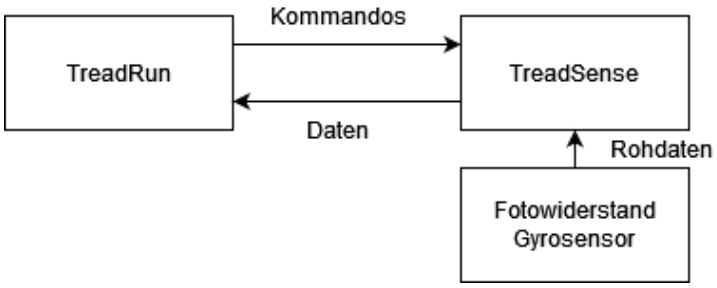

Abbildung 3: Relation TreadRun - TreadSense

3.1 Verbindungsprotokoll

Um Daten zwischen dem Raspberry Pi und der Software auszutauschen, wird das TCP-Protokoll verwendet.

Der Raspberry Pi wird hierzu als Server genutzt, um mehrere Computer an dasselbe Gerät zu verbinden. Es wurde ein Protokoll zum Anmelden des Geräts erstellt. Dieses besteht aus vier Actions.

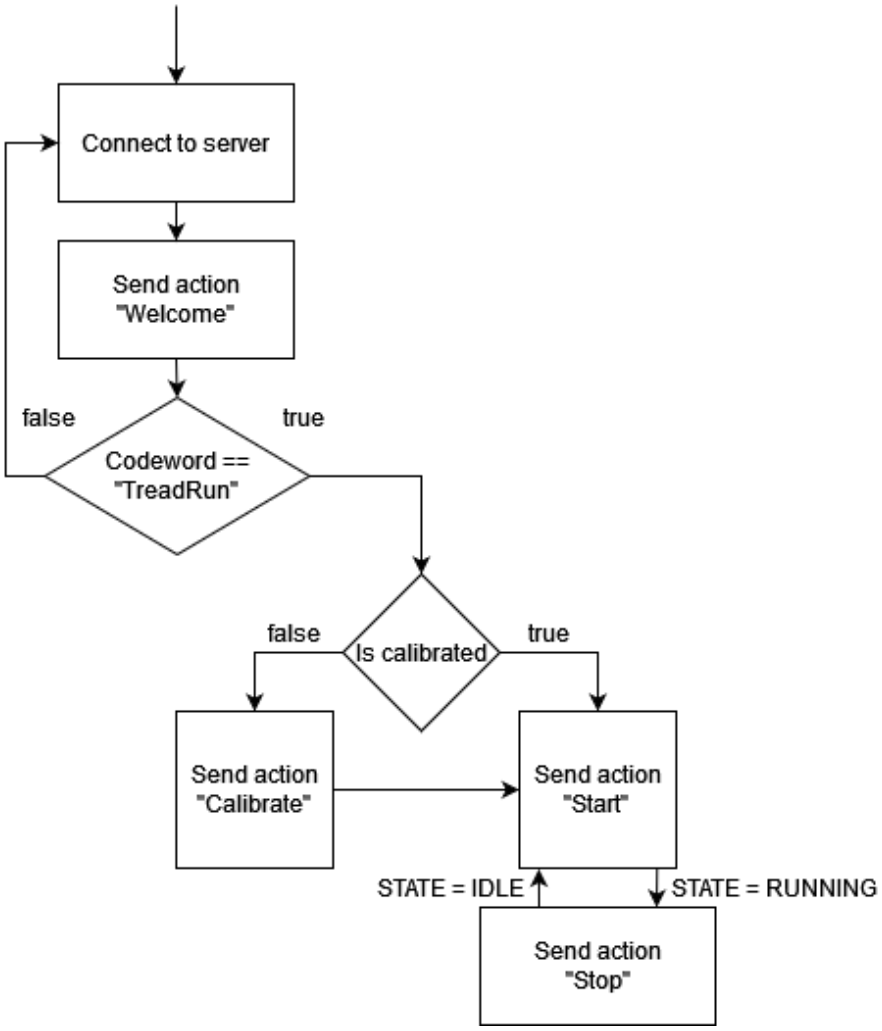

Abbildung 4: Connection flow chart

3.1.1 Welcome

Das Welcome-Protokoll ist die erste Aktion, die ausgeführt wird, sobald sich das Programm mit der Hardware verbindet. Es wird eine Beschreibung des Geräts mit einem bestimmten Codewort zurückgeschickt. Dies stellt sicher, dass kein anderes Gerät fälschlicherweise als ein valides Gerät erkannt wird, was später zu Laufzeitfehlern führen würde.

3.1.2 Calibrate

Das Gerät wird für das aktuelle Laufband kalibriert. Mehr dazu unter 3.2. Zurückgegeben wird der Status der Operation und, wenn vorhanden, jeglicher aufgetretener Fehler.

3.1.3 Start

Sobald diese Aktion ausgeführt wird, wird kontinuierlich die Geschwindigkeit sowie die aktuelle Steigung des Laufbands übergeben. Das Gerät befindet sich nun im Zustand RUNNING. Es kann keine andere Aktion ausgeführt werden außer Stop.

3.1.4 Stop

Diese Aktion kann nur ausgeführt werden, wenn das Gerät in dem RUNNING Zustand ist. Es beendet jegliche Aktivität und stellt das Gerät wieder auf den IDLE Zustand zurück. Es wird der Status der Operation sowie Fehlernachrichten zurückgegeben.

3.2 Geschwindigkeit

Um die Geschwindigkeit des Laufbandes auszulesen, müssen auf dem Laufband weiße Sticker angebracht werden. Diese werden durch den Fotowiderstand erfasst und durch die Programmierung verarbeite.

3.2.1 Kalibration

Bevor man die Geschwindigkeit auslesen kann, muss man das Gerät kalibrieren. Da das Gerät keine Vorstellung hat, wie viele und wie weit die Sticker auf dem Band verteilt sind, wird das Laufband auf 5 km/h beschleunigt und es wird die Zeit zwischen den Anschlägen des Fotowiderstands gemessen.

Durch die Formel

$$s = 5 * 3.6 * t$$

wird die Distanz zwischen jedem Sticker ermittelt und in eine Liste gespeichert. Die Distanz wird in mm berechnet, um leichter ermitteln zu können, wie viel Sticker auf dem Band sind.

Nachdem die einzelnen Distanzen zwischen den Stickern ermittelt sind, werden diese gespeichert.

3.2.2 Auslesen

Die Zeit zwischen jedem Ausschlag des Fotowiderstands wird gemessen.

Durch die Formel

$$v = \frac{s}{t}$$

kann die aktuelle Geschwindigkeit berechnet werden.

Die Distanz s erhält man aus der Liste, die beim Kalibrieren entstand.

```
1  public double CalculateVelocity(double time)
2  {
3      try
4      {
5          currentMps = distances[index].ToFixed(2) / time.ToFixed(0);
6          if (++index >= distances.Count) index = 0;
7          return currentMps;
8      }
9      catch (Exception)
10     {
11         return 0;
12     }
13 }
```

Code 2: Calculate velocity

Da man nicht unterscheiden kann, ob der erste Sticker, der erkannt wird, auch der erste in der Liste ist, werden die letzten 5 ermittelten Geschwindigkeiten addiert und der Mittelwert übergeben. Dadurch wird auch sichergestellt, dass Messfehler so gering wie möglich gehalten werden.

3.3 Steigung

Die Steigung ist insofern wichtig, um eine akkurate Auswertung des Laufes liefern zu können, sowie den Benutzer auf mögliche Steigungsunterschiede aufmerksam zu machen.

3.3.1 Kalibration

Die Kalibrierung der Steigung erfolgt durch das Auslesen des Gyrosensors. Der aktuelle Wert wird gespeichert.

3.3.2 Auslesen

Der Wert des Gyrosensors wird ausgelesen und mit dem gespeicherten Wert subtrahiert, wenn der Wert ein Offset über- beziehungsweise unterschreitet. Somit ergibt sich die aktuelle reale Steigung des Laufbands.

Diese Methode bietet die Möglichkeit, genauer die Steigung zu bestimmen, da der Sensor auch leicht schief angebracht werden könnte.

```
1  public int CalculateIncline(double currentActualIncline, double offset)
2  {
3      try
4      {
5          if (
6              currentActualIncline >= defaultIncline - offset &&
7              currentActualIncline <= defaultIncline + offset
8          )
9          {
10             return (int)currentActualIncline - (int)defaultIncline;
11         }
12
13         return 0;
14     }
15     catch (Exception)
16     {
17         return 0;
18     }
19 }
```

Code 3: Calculate incline

4 Dynamische Generation/API TP

In diesem Kapitel veranschaulichen wir die Technologien, welche in TreadRun verwendet werden, sowie einen Einblick in die Generierung der Spielwelt.

4.1 API

4.1.1 Overpass API

Die Overpass API dient zum Download von Kartendaten direkt von OpenStreetMap. Um die Kartendaten abzufragen, wird eine eigen Query-Sprache namens Overpass QL verwendet. Es kann alles abgefragt werden, was ein Mapper hochladen kann, wie Keys, Tags, Objekttypen und auch Kombinationen. Diese abgefragten Daten können als XML, CSV oder JSON zurückgegeben werden. Das Timeout ist standardmäßig auf 180 Sekunden eingestellt.

Overpass Turbo ist eine Website, um die Overpass QL zu testen. Mit verschiedenen Hilfsmitteln kann man eine Query ganz einfach zusammenstellen.

Es werden mehrere öffentliche Server zur Verfügung gestellt. Der Server von Kumi Systems [1] hat keine Ratelimits und Zugriffe werden nicht gespeichert.

```
1  [out:json][timeout:25]
2  [bbox:48.239971875,13.026151875,48.245588125,13.031768125];
3  (
4      node;way;
5      readonly[typeof=multipolygon](if:count_mebers() == 2);
6  );
7  out body;
8  (._;>;);
9  out body;
```

Code 4: Overpass Query

4.1.2 Open Routing Service

Open Routing Service ist eine Open Source Routing Instanz, die auf OpenStreetMap Daten basiert. Man kann verschiedenste Profile übergeben, inwiefern sich das Routing verhalten soll.

```
1  public class ORSProfile
2  {
3      public static readonly string DrivingCar = "driving-car";
4      public static readonly string DrivingHGV  = "driving-hgv";
5      public static readonly string CyclingRegular  = "cycling-regular";
6      public static readonly string CyclingRoad = "cycling-road";
7      public static readonly string CyclingMountain  = "cycling-mountain";
8      public static readonly string CyclingElectric  = "cycling-electric";
9      public static readonly string FootWalking  = "foot-walking";
10     public static readonly string FootHiking  = "foot-hiking";
11     public static readonly string WheelChair  = "wheelchair";
12 }
```

Code 5: OpenRoutingService Profiles

Übergeben werden die geografischen Koordinaten, welche in Betracht gezogen werden sollen. Zurückgegeben werden je nach Einstellung eine Polyline und die Distanz als JSON Objekt.

Neben der Directions API bietet der Service auch eine Geocode API.

4.1.2.1 Geocode Search

Dieser API-Endpunkt gibt eine Reihe an geografischen Koordinaten, auf Basis des übergeben Strings, zurück.

Wir verwenden ihn, um auf der Karte den Benutzer leichter zwischen Orten wählen zu lassen.

4.1.2.2 Geocode Reverse

Der Reverse Geocode API-Endpunkt liefert eine genaue Aufschlüsselung über geografische Koordinaten. So kann man auslesen, in welchem Land und in welcher Stadt sich der Benutzer gerade virtuell aufhält.

4.1.3 Strava API

Um, wie alle größeren Sportapps eine Integrierung mit Strava zu haben, benutzen wir die Strava API.

Dazu gibt es einen eigenen Anmeldevorgang im Spiel selbst. Strava bietet die Möglichkeit über OAuth2 einen Athleten anzumelden und Berechtigungen für diesen zu erhalten.

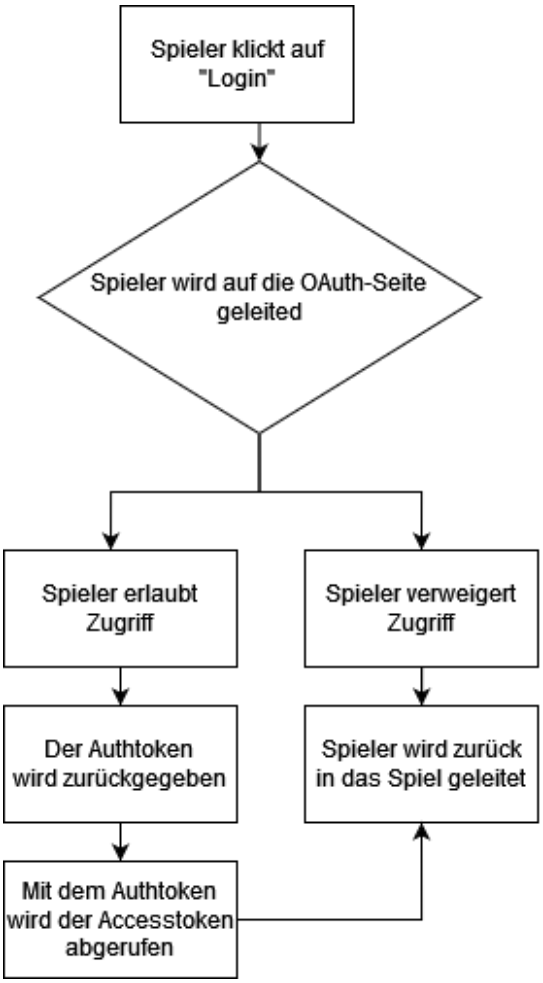

Abbildung 5: Strava OAuth Flow

Jeder API-Endpunkt benötigt einen Access-Key des jeweiligen Benutzers mit der erforderlichen Berechtigung. Diesen bekommt man nach dem Anmeldevorgang sowie einen Refresh-Key. Der Access-Key muss immer wieder einmal mit dem Refresh-Key neu angefordert werden.

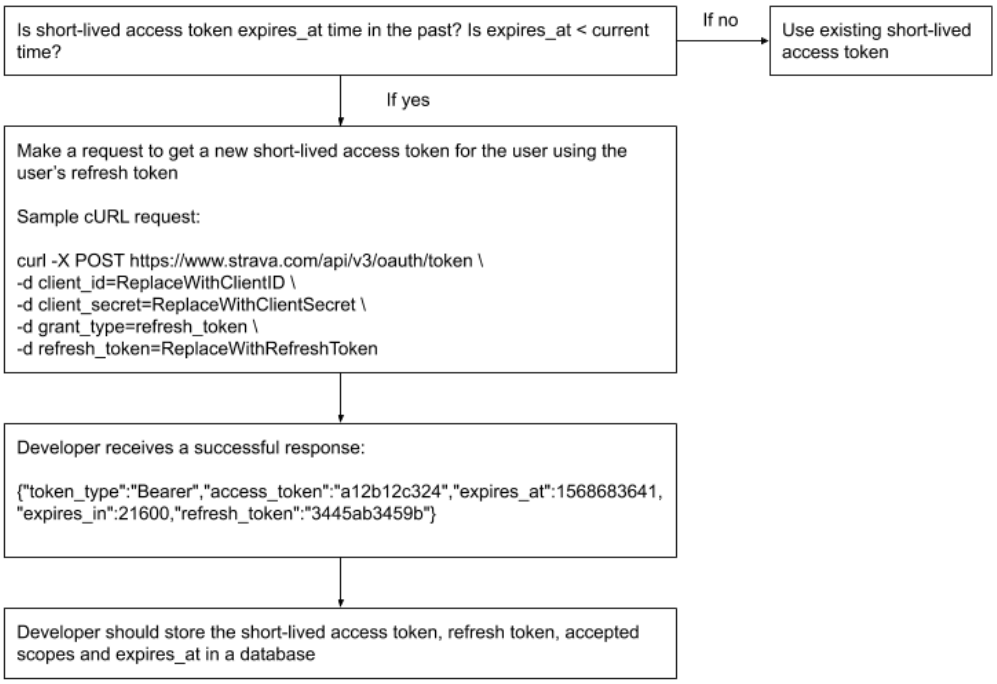

Abbildung 6: Strava Access Flow [2]

Um eine Aktivität hochladen zu können, wird eine GPX-Datei verwendet. Das GPX-Format ist ein erweitertes XML-Format, welches ein Array aus geografischen Koordinaten und einem Zeitstempel beinhaltet.

Diese Datei wird zur Laufzeit erstellt und am Ende einer Strecke über den API-Endpunkt hochgeladen. Somit hat der Benutzer die Aktivität in seinem Strava Account gespeichert.

4.1.4 Discord Rich Presence

Die Discord Rich Presence setzt sich aus 3 wichtigen Dingen zusammen. Der Zustand des Programmes, ob der Benutzer überhaupt Discord besitzt und das Development Kit von Discord selbst.

Die SDK wird für den Zugriff auf Discord benötigt. Dieses beinhaltet alle Funktionen, die benötigt werden, die Rich Presence zu erneuern.

Nur wenn Discord auf dem PC des Benutzers läuft, wird die Rich Presence eingeschalten.

Der Zustand des Programmes gibt vor, welche Daten die Rich Presence ausgeben soll. Wenn der Benutzer in der Karte oder im Menü ist, wird nur eine Statusmeldung ausgegeben. Falls der Benutzer aber gerade läuft, wird die Pace, die Zeit und die aktuelle Distanz angezeigt.

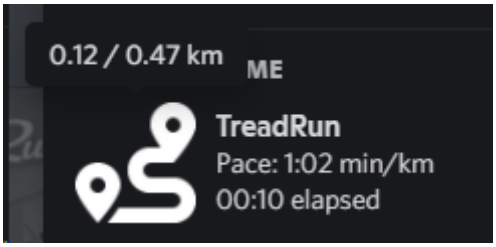

Abbildung 7: Discord Rich Presence

4.2 Dynamische Generierung

In diesem Teil geht es um die dynamische Generierung der Spielwelt. Wir erklären, wie die Welt aufgebaut wird und die richtige Gegend geladen wird.

Der Ablauf von der Benutzereingabe bis zur begehbaren Welt erfolgt in 4 Schritten.

4.2.1 OpenStreetMap Features

OpenStreetMap Features sind die Dinge, die die OpenStreetMap ausmachen. Jedes Element auf OpenStreetMap kann als Node, Way oder Relation angegeben werden.

Jedes Element besitzt mindestens einen Tag, welcher die Art des Elements angibt. Eine Straße ist zum Beispiel vom Typ Line und besitzt den Tag „highway=primary“, und ist somit eindeutig als Straße definiert.

Alle Flächen bestehen aus einer Line mit derselben Start- und Endnode. Somit können auch Features wie Gebäude oder Nutzland erstellt werden.

Das Feature-System wurde als Abstraktion implementiert, um neue Elemente so einfach wie möglich hinzuzufügen. Jedes Feature benötigt eine Build-Methode und eine IsFeature-Methode. IsFeature überprüft, ob das Element jenes Feature ist. In der Build-Funktion ist der Aufbau der Features implementiert.

```
1  public abstract class OSMFeature
2  {
3      public abstract IEnumerator Build(Transform parent = null);
4      public abstract OSMFeature IsFeature(IOSM osm);
5  }
```

Code 6: OSMFeature definition

Es gibt sieben Standardfeatures, welche alle 3 Elementtypen abdecken. Jedes davon wurde mit einer Standard Build-Funktion implementiert.

```
public override IEnumerator Build(Transform parent)
{
    var pos = MercatorProjection.LatLon2XY(Coordinates);
    var go = Resource.LoadModel("TreeFab");
    go.transform.position = pos;
    go.transform.rotation = Quaternion.Euler(0, UnityEngine.Random.Range(0, 360), 0);
    go.transform.SetParent(parent);
    yield return null;
}
```

Code 7: Build-Method of a tree

Gebäude

Gebäude werden als Area angesehen. Ein Gebäude kann aus verschieden Eigenschaften bestehen. Dazu gehören unter anderem: Höhe, Stockwerke und die Dachform.

Gebäude müssen im Uhrzeigersinn erstellt werden, da ansonsten das Mesh in die entgegengesetzte Richtung generiert wird. Daher müssen wir überprüfen, ob die Punkte im Uhrzeigersinn sind, bevor das Gebäude generiert werden kann.

Für die Generation der Gebäude benutzen wir das ProceduralToolkit von Daniil Basmanov [3]. Ein Gebäude kann beliebig viele Seiten besitzen, denn die Fassaden werden von Punkt zu Punkt erstellt. Das Dach kann einen Giebel besitzen oder flach sein.

Straßen

Für eine Straße wird ein Mesh entlang den Nodes generiert und mit einem Material versehen. Jede Oberfläche kann ein eigenes Material besitzen. Eine Straße kann jede beliebige Breite annehmen. Der Standardwert dafür ist 1 Meter.

Kreuzungen werden erkannt, aber es wird kein spezielles Mesh dafür generiert.

Bäume

Es gibt ein Baummodell, das instanziiert wird. Nachdem es nur ein Modell gibt, wird es nach Zufallsprinzip gedreht. Dieses Modell ist austauschbar und es ist leicht auch mehrere Modelle zu benutzen.

Gewässer

Gewässer sind ein Spezialfall, da das Terrain dafür deformiert werden muss. Da dies eine sehr leistungsfressende Operation ist, muss das Terrain Stück für Stück abgesetzt werden. Alles was Gewässer betrifft, wird in einem Thread ausgeführt.

4.2.2 Mercator Projektion

Um die Längen- und Breitengrade der Kartendaten für das Unity Positionssystem nutzbar zu machen, müssen diese umgewandelt werden. Dies geschieht mit der Mercator Projektion, sie wandelt geografische Koordinaten in Koordinaten um. Die Koordinaten werden in Metern angegeben. Da das Unity Positionssystem auf die echte Welt übersetzbar ist mit einem Meter gleich einer Unit, müssen wir keine weiteren Berechnungen aufstellen.

```
1  private static Units TranslateCoords(Coordinates coords, Coordinates origin)
2  {
3      //Degree always 0
4      double angle = ToRadians(0);
5
6      double xx = (coords.Longitude - origin.Longitude) * MetersDegLon(origin.Latitude);
7      double zz = (coords.Latitude - origin.Latitude) * MetersDegLat(origin.Latitude);
8
9      double r = Math.Sqrt(xx * xx + zz * zz);
10
11     if(r != 0)
12     {
13         double ct = xx / r;
14         double st = zz / r;
15
16         xx = r * ((ct * Math.Cos(angle)) + (st * Math.Sin(angle)));
17         zz = r * ((st * Math.Cos(angle)) - (ct * Math.Sin(angle)));
18     }
19
20     return new Units(xx, zz);
21 }
```

Code 8: Mercator Projection

4.2.3 Polyline

Eine Polyline speichert eine Reihe geografischer Koordinaten auf bis auf die fünfte Nachkommastelle als String enkodiert.

```
48.24395, 13.02844
"usmeHwronA"
```

Code 9: Encoded polyline

4.2.4 Bounding Box

Mit Hilfe einer Bounding Box kann eine Fläche auf der Oberfläche der Erde angegeben werden. Dazu werden zwei geografische Koordinaten benötigt. Eine für links unten und eine weitere für rechts oben.

4.2.5 Aufbau der Welt

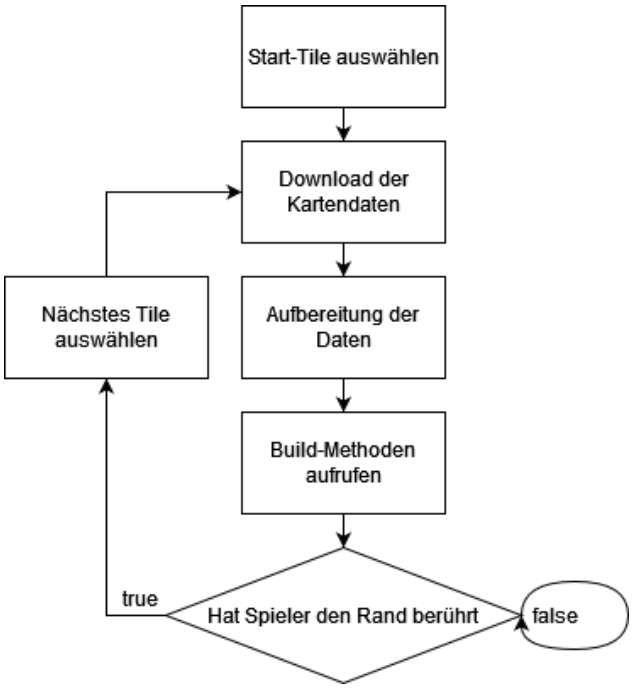

Abbildung 8: Flowchart - Generierung

4.2.5.1 Tile-Generierung

Im ersten Schritt wird die benutzerdefinierte Laufstrecke in ca. 1km²-Tiles aufgeteilt. Dies hat den Vorteil, dass selbst bei einer langsameren Internetverbindung ein nahtloser Übergang möglich ist, da der Download der Daten auf ein Minimum reduziert wird.

Um die Laufstrecke aufzuteilen, wird zunächst die kleinstmögliche Bounding Box um die Polyline der Laufstrecke gesucht. Im nächsten Schritt wird diese um einen halben Kilometer auf jeder Seite vergrößert, um auch die Randbereiche abzudecken.

Aus der Bounding Box wird nun ein 2D-Raster erstellt, das den 1 km²-Tiles entspricht.

Um die Tiles später genau zuordnen zu können, wird eine ID vergeben, die ihre Position in dem Raster beinhaltet. Jedes Tile kennt seine Bounding Box und Position im Raster.

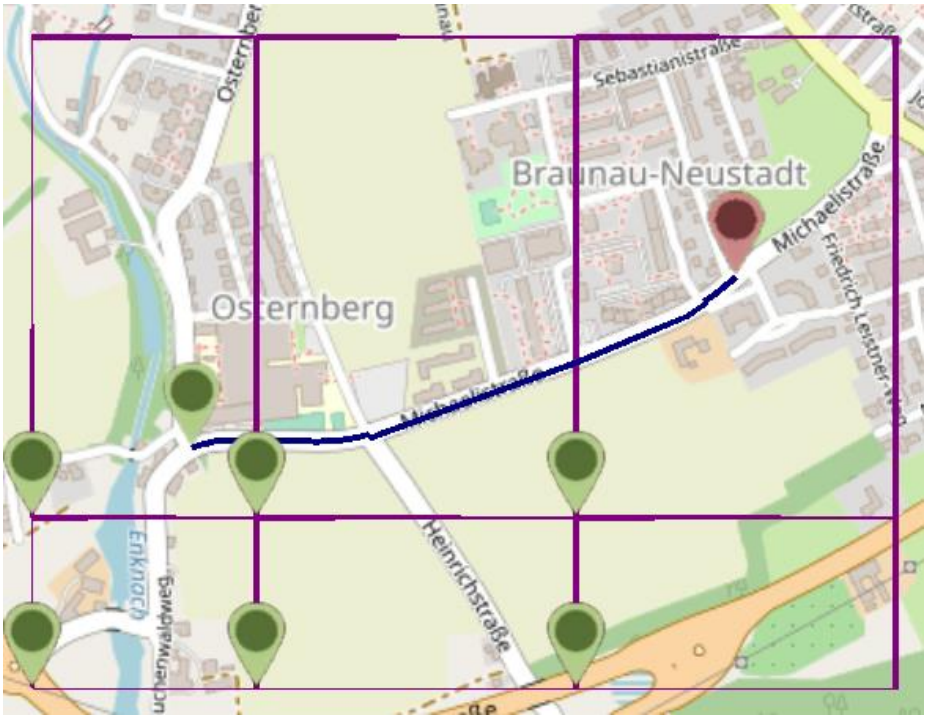

Abbildung 9: Tileraster

4.2.5.2 Download der Kartendaten

Wenn ein Tile generiert werden soll, müssen zuerst die Kartendaten abgerufen werden. Mithilfe der Overpass API können wir die Kartendaten für eine Bounding Box abrufen. Diese

Daten werden als JSON-Objekt übergeben und beinhalten jede Node, jeden Way und jede Relation, die in der Bounding Box gefunden wurden.

Um die Daten verarbeiten zu können, müssen diese zuerst von einem JSON-Objekt auf Nodes, Ways und Relations umgewandelt werden. Denn jede Node die in einem Way verbaut ist, wird nicht mehr benötigt. Diese Operation benötigt im Durchschnitt 2,5 Sekunden.

Nun wird aus der Liste der implementierten OpenStreetMap Features jeder Way, jede Node und Relation überprüft, ob die Tags dem Feature entsprechen. Am Ende wird eine Liste aller Features mit entsprechenden Eigenschaften in das Tile gespeichert.

Diese Operation benötigt im Durchschnitt 16 Millisekunden.

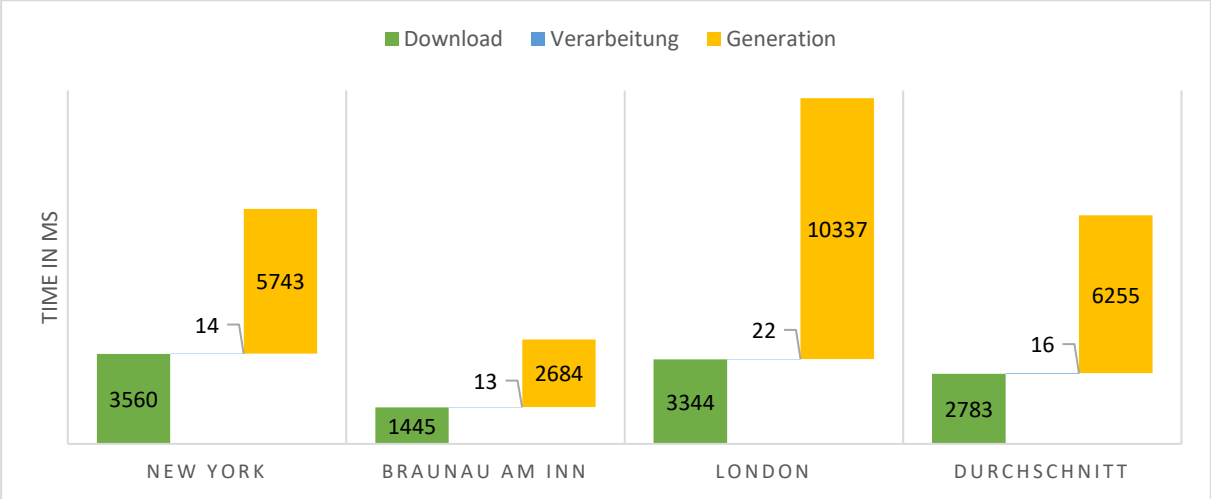

Abbildung 10:Generationszeiten eines Tiles

1

4.2.5.3 Generierung

Nach dem Download erfolgt die Generierung. Zuerst wird das Terrain für das Tile erstellt. Auf dem Terrain werden acht Trigger erstellt, die 100 Meter in das Tile ragen. Die Trigger geben den benachbarten Tiles bescheid, wenn der Spieler diese berührt. Die benachbarten Tiles werden danach generiert. Die Trigger werden auch dafür verwendet um den Spieler zu Tracken.

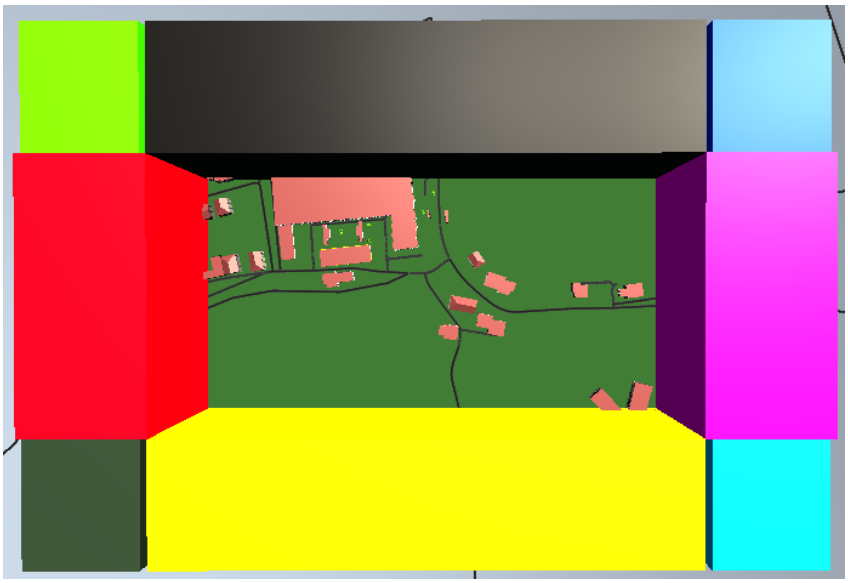

Abbildung 11: Terrain with trigger

Nun wird für jedes Feature die Build-Methode aufgerufen. Siehe OpenStreetMap Features.

Viele Objekte zu generieren, birgt die Gefahr, dass Unity zu ruckeln beginnt. Darum benutzen wir einen Object Pool, der zu Beginn der Scene geladen wird. Wenn also ein Objekt benötigt wird, gibt der Pool eines seiner gespeicherten Objekte frei. Sobald das Objekt wieder zerstört werden würde, nimmt der Object Pool das Objekt wieder auf. Dadurch wird zwar mehr Arbeitsspeicherplatz benötigt, dafür ist es Performanter.

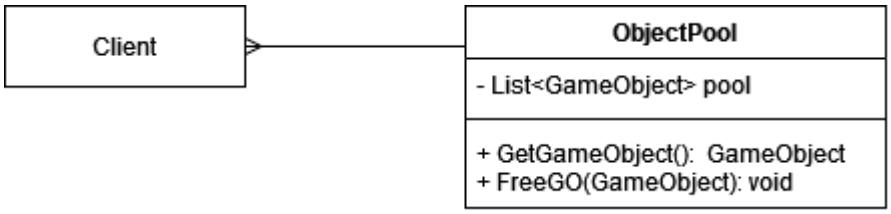

Abbildung 12: Object Pooling

4.2.5.4 Dekonstruktion

Die Dekonstruktion des Tiles erfolgt, wenn sich der Charakter mindestens zwei Tiles weiter befindet.

Wenn ein Tile den Startpunkt sowie den Endpunkt der Strecke besitzt, wird es nicht zerstört da das Tile später wieder geladen werden müsste. Stattdessen wird es deaktiviert und verbraucht somit keine CPU - Leistung und GPU - Leistung mehr.

5 Game Design TS

Dieses Kapitel veranschaulicht unser Konzept von TreadRun. Wir erklären unseren Entwicklungsprozess hinter den Funktionen der Spielwelt.

5.1 Gameplay

Im diesem Unterkapitel beschreiben wir alle Spielmechaniken, mit denen der Nutzer in Kontakt kommen kann. Wie zum Beispiel die Navigation durch das Programm und die verschiedenen Abläufe in TreadRun. Wir befassen uns mit dem gesamten Ablauf des Programms und veranschaulichen verschiedene Situationen mit Beispielen.

5.1.1 Steuerung

TreadRun wird vom Nutzer mithilfe eines Laufbands gesteuert. Dazu wird die zuvor beschriebene Hardware benötigt. Dabei wird die Geschwindigkeit des Nutzers zur Laufzeit übertragen und im Programm selbst dargestellt. Im HUD wird dem Nutzer ein übersichtlicher Einblick in seine Leistung bzw. aktuelle Zeit gegeben. Die Navigation der verschiedenen Menüs erfolgt hingegen über den Computer, auf dem TreadRun läuft.

5.1.2 Character Controller

Der Charakter des Nutzers wird von einem sogenannten Character Controller gesteuert. Darunter ist ein Skript zu verstehen, welches verschiedene Eigenschaften des Charakters steuert. Die wichtigste Funktion ist hierbei die Routenverfolgung. Diese sorgt dafür, dass der Charakter den vom Nutzer gewählten Kurs reibungslos abläuft. Wenn der Charakter seine Laufrichtung ändert, muss sich natürlich auch seine Rotation anpassen. Mithilfe von Winkelfunktionen wird die neue Ausrichtung des Charakters berechnet.

5.1.3 Ablauf

Der Nutzer findet sich beim Start von TreadRun, in dessen Hauptmenü wieder. Von dort gelangt dieser auf das Kartenmenü, wo der Laufkurs gewählt wird. Daraufhin wird der Kurs generiert.
Diese Abläufe werden durch sogenannte Szenen ermöglicht. Grundsätzlich ist unter einer Szene ein Abschnitt des Programms zu verstehen, der nur aktiv ist, wenn der Nutzer diesen verwendet. Sobald eine Szene verlassen wird, werden jegliche verwendeten Grafikelemente entladen und laufende Skripte deaktiviert.

Dies kann zum Beispiel anschaulich gemacht werden, wenn der Nutzer das Hauptmenü verlässt, um sich das Kartenauswahl anzusehen. Sofort werden sämtliche Skripte für Menü Events deaktiviert und die zugehörigen UI-Elemente entladen.

Es ist jedoch oft nötig, einige Daten einer Szene auch in anderen Szenen zur Verfügung zu haben. Um zu wissen, welche Szene als Nächstes geladen werden muss, verwenden wir zum Beispiel eine statische Klasse, die einen Index der nötigen Szene beinhaltet. Szenen helfen uns dabei, TreadRun so übersichtlich wie möglich zu halten sowie flexibler zu entwickeln. Außerdem macht dies die Optimierung der Performance weitaus einfacher.

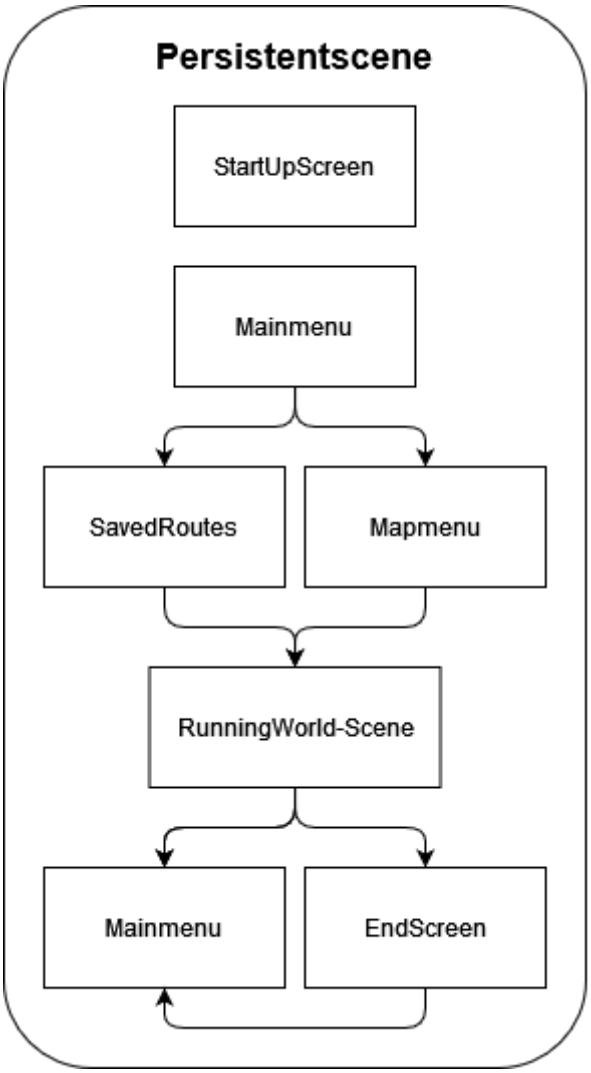

Abbildung 13: Scenen Ablauf

5.1.4 Ziel

Das Grundkonzept von TreadRun ist, einen Kurs zu wählen und diesen einem Laufband abzulaufen. Durch unsere Integration von Strava, ist es möglich, seine sportlichen Aktivitäten zu tracken.

TreadRun misst sowohl die Geschwindigkeit als auch die Zeiten des Nutzers. Sobald dieser das Ende seines Laufs erreicht, erscheint ein Endscreen, der die Daten des Laufs übersichtlich darstellt. Ziel dabei ist, den Nutzer stets zu motivieren, indem diesem oft seine aktuellen Daten sowie seine Entfernung zum selbst gesetzten Ziel gezeigt werden.

5.2 User Interface

In diesem Kapitel beschreiben wir sämtliche Verwendungen von UI-Elementen in TreadRun. Diese sind ein wichtiges Element, um eine angenehme Nutzererfahrung zu bieten.

Menüs müssen grafisch ansprechend sein und ein zufriedenstellendes Feedback geben. Verschiedenste Szenen müssen eingebunden werden und dabei die Ladezeiten dieser beachtet werden.

5.2.1 Hauptmenü

Das Hauptmenü ist eines der wichtigsten Menüs, da der Nutzer hier die meiste Zeit verbringt und erste Eindrücke vom Programm gewinnt. Das Hauptmenü muss viele Aufgaben erfüllen und dabei übersichtlich sein.

Hauptfunktionen

Die linke Button-Leiste beinhaltet die wichtigsten Grundfunktionen. Mit dem Start-Button wird der Nutzer, sofern dieser einen Sensor verbunden hat, auf die Kartensektion weitergeleitet. Der Save Routes-Button lädt ein weiteres Menü, das sämtliche zuvor gespeicherten Laufstrecken enthält. Zum Beenden des Programms wird der Quit-Button verwendet.

Abbildung 14: Menü Hauptfunktionen

Button Feedback

Gutes Button Feedback wird mit sogenanntem Sprite Swapping erreicht. Das bedeutet, dass auf dem Button ein Bild liegt, welches je nach Nutzerinteraktion getauscht wird. Wird der Button beispielsweise mit dem Cursor berührt, wird die Buttongrafik durch eine modifizierte Version ersetzt. Wir unterscheiden in dieser Buttonleiste zwischen drei Zuständen, was bedeutet, dass drei Buttongrafiken für einen Button benötigt werden. Diese erstellen wir in dem Bildbearbeitungsprogramm Gimp.

Da diese Methode zur Erzeugung von Button-Feedback sehr aufwendig ist, verwenden wir sie nur in Teilen des Haupt- und Saved Route Menüs. Unity bietet die Möglichkeit, Farbe sowie Sichtbarkeit von Buttons je nach Interaktion schnell anzupassen.

Hintergründe

Um die Menüs weniger monoton zu gestalten, verwenden wir verschiedene Hintergrundbilder. Diese Bilder zeichnen wir in Clip Art Studio, wobei wir Vorlagen aus Unity verwenden. Die Hintergründe werden in regelmäßigen Zeitabständen ausgewechselt. Um einen fließenden Bildübergang zu ermöglichen, schreiben wir ein Skript, dass die Sichtbarkeit des Bilds langsam verringert. Sobald es nicht mehr sichtbar ist, wir das nächste Bild langsam eingeblendet. So wird ein dynamisches Menü ermöglicht, dass stets Abwechslung bietet.

Abbildung 15: Menü Hintergrund

Verbindung mit Hardware

Zur Nutzung von TreadRun wird entsprechende Hardware benötigt. Daher befindet sich in der unteren Leiste ein Button für den Verbindungsaufbau zur Hardware. Dieser wechselt seine Farbe, sobald eine Verbindung aufgebaut wurde. Ohne diese Verbindung kann der Nutzer das Hauptmenü nicht verlassen. Da dies bei der Entwicklung stören würde, implementieren wir einen Developer-Modus, mit dem diese Sperre umgangen wird. Der Modus wird mit der Tastenabkürzung Shift + P umgeschaltet.

Verbindung mit Strava

Wir ermöglichen einen Log-in bei Strava über das UI-Element am rechten Rand des Hauptmenüs. Diese erscheint bei einem Klick auf das User-Icon. Hierbei wird eine Menüanimation verwendet, um das Element zur Anmeldung ein- und auszublenden.

Startup-Screen

Beim Start von TreadRun wird vor dem Laden vom Hauptmenü ein Start-up-Screen gezeigt. Hier verwenden wir Logos und eine Animation. Der Screen bleibt für wenige Sekunden sichtbar, bis das Hauptmenü geladen wird.

Abbildung 16: Startup-Screen

5.2.2 Gespeicherte Kurse

Es muss möglich sein, seine gespeicherten Kurse einzusehen. Dazu haben wir ein eigenes Menü erstellt. Hier werden alle gespeicherten Kurse aufgelistet. Diese können von dort auch geladen werden. Die Möglichkeit, Kurse zu speichern, besteht bei der Auswahl einer neuen Route im Kartenmenü. Dort wird dem Nutzer angeboten, seine Kursauswahl zu speichern, um sie später nochmals zu laden.

Die grafischen Elemente in diesem Menü erstellen wir in Gimp. Es stehen acht Speicherslots für Kurse zur Verfügung. Wie im Hauptmenü werden auch hier die zuvor beschriebenen Hintergründe verwendet.

5.2.3 Ladebildschirm

TreadRun benötigt beim Generieren der virtuellen Welt eine gewisse Ladezeit. Um dem Nutzer zu zeigen, dass ein Ladevorgang stattfindet, verwenden wir einen Ladebildschirm. Mit einer kleinen Animation eines Läufers wird gezeigt, dass das Programm weiterhin läuft und nicht abgestürzt ist. Außerdem werden während des Ladevorgangs Zitate zum Sport abwechselnd eingeblendet.

Der Ladebildschirm muss immer wissen, welche Szene im Hintergrund geladen werden muss. Wie zuvor beschreiben, werden Szenendaten aus einer statischen Klasse bezogen. Sobald

die Zielszene vollständig geladen wurde, wird der Ladebildschirm deaktiviert und die Szene gewechselt.

5.2.4 HUD

Mit dem Head-up-Display werden dem Nutzer während des Trainings seine aktuellen Daten angezeigt. In der HUD wird sowohl die zurückgelegte Strecke als auch die Geschwindigkeit des Nutzers dargestellt.

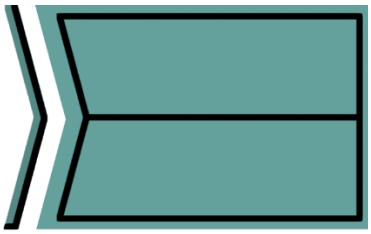

Abbildung 17: HUD Layout

Um weniger zu stören, ist die HUD etwas durchsichtig. So entsteht ein besserer Blick auf den Laufkurs.

5.2.5 Pausenmenü

Pause

Natürlich muss das Lauftraining jederzeit pausiert werden können. Dazu kommt ein Pausenmenü zum Einsatz. Um Ladezeiten zu vermeiden, liegt dieses in derselben Szene wie die generierte Welt. Es wird sichtbar, sobald der Nutzer die ESC-Taste drückt. In diesem Menü ist es möglich, das Lauftraining vorzeitig zu beenden oder direkt zurück zum Hauptmenü zu springen. Außerdem kann hier der Nutzer seinen virtuellen Charakter wählen.

Charakter Auswahl

Wir geben dem Nutzer die Möglichkeit, zwischen einem männlichen oder weiblichen Charakter zu wählen. Die Auswahl erfolgt im Pausenmenü durch zwei Buttons. Bei einer Auswahl wird das Charaktermodell sofort und ohne Ladezeit gewechselt. Die Auswahl des Charaktermodells wird durch ein Skript gesteuert, das die Buttonevents des Pausenmenüs enthält.

5.2.6 Endscreen

Sobald der Nutzer sein Ziel erreicht, wird der Endscreen geladen. Dort werden die Daten des Laufs übersichtlich dargestellt. Von hier gelangt der Nutzer wieder zum Hauptmenü und kann eine neue Runde starten.

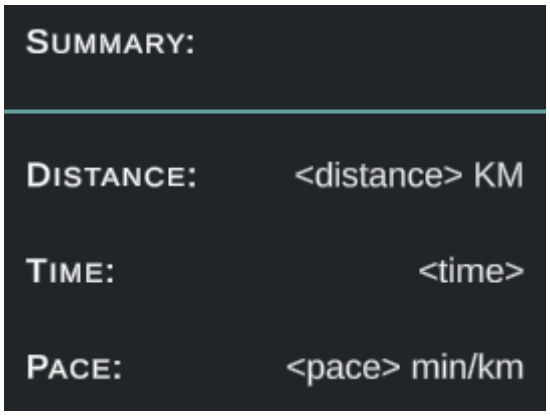

Abbildung 18: Endscreen Summary

5.3 Animation

Um die Laufsimulation realistisch zu gestalten, sind Charakteranimationen essenziell. In diesem Abschnitt beschreiben wir unsere Verwendung von Animationen. Außerdem begründen wir unsere Auswahl von Charaktermodellen und zeigen, wie diese noch erweitert werden könnten.

5.3.1 Mixamo

Über Mixamo

Mixamo ist eine amerikanische Firma, die sich auf die Entwicklung von 3D-Charakter-Animationen spezialisiert. Ihr Fokus liegt auf der schrittweisen Automatisierung von Charakter-Animationen.

Asset Bibliothek

Mixamo stellt online eine große Bibliothek von Charakteren und dazugehörigen Animationen zur Verfügung. Diese können wir ohne Einschränkungen in unserem Projekt nutzen. Mixamo Animationen lassen sich einfach in eine Game Engine importieren. Sie verwenden das sogenannte FBX-Format.

Abbildung 19: Charakter Format

Export Einstellungen

Vor dem Import können auf der Mixamo Website noch einige Details eingestellt werden. Zum Beispiel die Qualität und die Framerate der Animation. Bei Charaktermodellen können auch Einstellungen speziell für Unity gewählt werden. Wahlweise können Animation auch gekürzt werden.

Nach dem Import in Unity, können Animationen weiter modifiziert werden. Mithilfe von Mixamo, können wir dem Nutzer ein realistisches Charakterverhalten bieten und so für ein angenehmes Laufgefühl sorgen.

5.3.2 Charaktere

Spielermodell in 3D-Umgebung

Der Nutzer steuert beim Laufen einen virtuellen Charakter. Dabei gibt es die Wahl zwischen einem männlichen und weiblichen Charakter. Bei der Auswahl der Charaktermodells entscheiden wir uns für zwei sehr einfache Modelle.

Wir entscheiden uns für diese Modelle, um unseren Entwicklungsprozess zu erleichtern. Bei der Implementierung von Modellen in 3D-Umgebungen muss vieles beachtet werden.

Stil

Da wir die Laufkurse in einem Comic-Stil gestalten, muss auch der Charakter entsprechend angepasst werden. Beleuchtung und Farbe müssen weitgehend verändert werden, um ein zufriedenstellendes Ergebnis zu erzielen. Dies erreichen wir mit unserem Cel-Shader, den wir im Shader Abschnitt beschreiben.

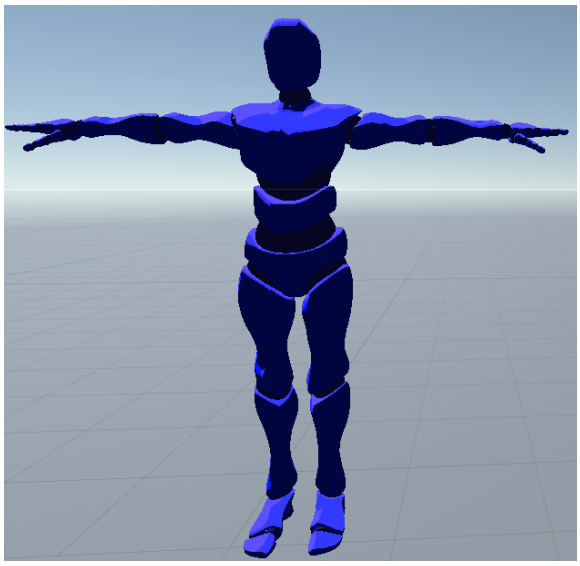

Abbildung 20: Angepasstes Charaktermodell

Kompatibilität

Auch die Animationskompatibilität darf nicht missachtet werden. Unser Charaktermodell ist mit vielen Animationen von Mixamo kompatibel. Dadurch ist es uns möglich, Animationen flexibel

zu ändern oder auszutauschen. Beispielsweise könnten jederzeit neue Idle-Animation für bestimmte Situationen hinzugefügt werden.

5.3.3 Laufanimation

Da der Kerninhalt von TreadRun im Ablaufen von Kursen besteht, ist die Laufanimation sehr wichtig. Die Animation wird gestartet, sobald sich das Laufband des Nutzers in Bewegung setzt.

Beim Laufen herrscht ein Zusammenspiel zwischen Animationsgeschwindigkeit und Laufgeschwindigkeit, was im Animation Playback Abschnitt beschrieben wird.

5.3.4 Idle Animationen

Idle Animation sind Animationen, die abgespielt werden, wenn das Programm stillsteht. In diesem Fall ist darunter ein Stillstand des Laufbands zu verstehen.

Wir verwenden hierbei eine Auswahl an Animationen. Unter anderem eine Animation, in der sich der Charakter hinsetzt und sich hinlegt. Dies sind Features, die zwar nicht unbedingt nötig sind, jedoch tragen sie zu einem besseren Eindruck von TreadRun bei.

5.3.5 Animation States

States

Die aktiven Animationen in TreadRun werden durch States gesteuert. Diese werden in Unity mit einem visuellen Editor aufgebaut. Hier wird die Reihenfolge von allen Zuständen ersichtlich.

Editor

Wie schon erwähnt, werden Animationsreihenfolgen im Unity Editor aufbaut. Gestartet wird vom Start-State. Dieser wird aufgerufen, sobald auf den Animationscontroller des Charakters zugegriffen wird. Vom Start wird der Zustand sofort auf den ersten Idle-Zustand gesetzt. Dort wird gewartet, bis der Nutzer zu laufen beginnt. Nach längerer Zeit werden andere Idle-Animationen abgespielt. Wenn der Nutzer nun zu laufen beginnt, werden diese sofort

abgebrochen und auf den Laufzustand gewechselt. Die Bedingungen für den Zustandswechsel selbst werden in einem Skript programmiert. Für jeden Zustand können Übergangsbedingungen konfiguriert werden. So können alle Abläufe richtig festgelegt werden.

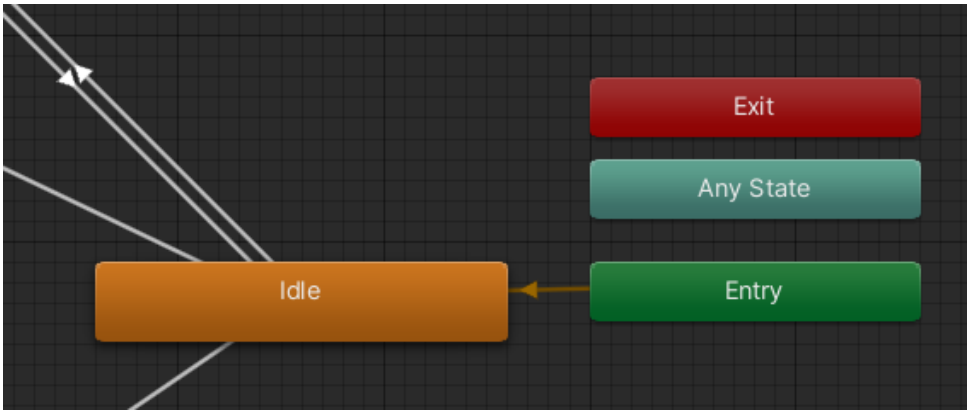

Abbildung 21: Animation Entry

5.3.6 Animation Playback

Beim Abspielen von Animationen muss die Abspielgeschwindigkeit stets beachtet werden. Bei unseren Idle-Animationen bleibt die Abspielgeschwindigkeit auf dem Standardwert. Dies liegt daran, dass bei diesen Animationen die Position des Charakters nicht verändert wird.

Bei der Laufanimation hingegen wird die Geschwindigkeit der Animation auf die Geschwindigkeit des Nutzers angepasst.

5.4 Kamera

Die Kamera bestimmt, was der Nutzer sieht. In TreadRun soll der Charakter des Nutzers aus der dritten Person sichtbar sein. Daher folgt die Kamera stets dem Charakter und passt ihren Winkel je nach Charakterdrehung an.

5.4.1 Cinemachine

Cinemachine ist eine Unity Erweiterung, die uns komplexe Kamera Controller zur Verfügung stellt. Dabei müssen wir nur wenig eigenen Code schreiben.

3rd Person Setup

Das Aufsetzen einer passenden Kamera ist sehr einfach. Zunächst wird ein Cinemachine Brain der Szene hinzugefügt. Dies ist für die Kommunikation zwischen der Main Camera von Unity und den hinzugefügten Cinemachine Cameras zuständig.

Cinemachine verwendet virtuelle Kameras. Diese sind keine Camera Objects, sondern Camera Controller. Sie steuern also die Eigenschaften eines Camera Objects von Unity. Es gibt viele Arten der virtuellen Kameras, deren Verhalten mit Code erweitert werden kann. Wir verwenden die Freelook Virtual Camera.

Freelook Virtual Camera

Wir verwenden diese Vorlage, da sie die besten Anpassungsmöglichkeiten für eine 3rd Person Camera bietet. Besonders wichtig für uns sind die Target Einstellungen. Wir bestimmen, auf welches Objekt die Kamera gerichtet ist und welchem Objekt sie folgt. Einstellungen wie der Radius, in dem sich die Kamera bewegen darf, werden auch spezifiziert.

5.4.2 Kamera Rotation

Wenn sich die Rotation des Charakters ändert, muss sich auch die Kamera mitdrehen. Dazu verwenden wir die in Cinemachine enthaltene Look At Einstellung.

Cinemachine Look At

Die Look At Einstellung informiert die Kamera, auf welches Objekt sie schauen muss. Wenn das entsprechende Objekt nun seine Position ändert, passt sich die Kamera entsprechend an. Wir ändern einige spezielle Einstellungen, damit die Rotation nicht zu abrupt ist. Zum einen erfolgt die Rotation nach einer Zeitverzögerung, zum anderen beschränken wir die Geschwindigkeit der Rotation.

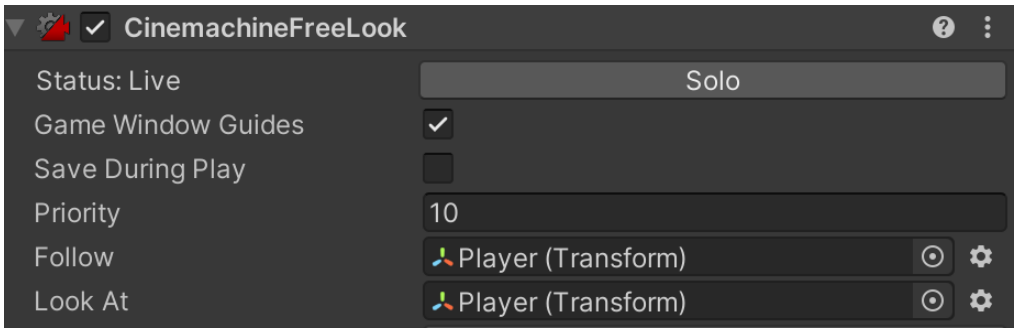

Abbildung 22: Cinemachine Look At

5.4.3 Kamera Kollision

Bei der Verfolgung des Charakters kann es immer wieder vorkommen, dass die Kamera mit einem Gebäude kollidiert. Hierzu gibt es mit der Cinemachine auch eine Lösung.

Cinemachine Collider

Mit dem Cinemachine Collider kann die Sichtbarkeit des gewählten Targets verbessert werden. Diese zusätzliche Cinemachine Erweiterung prüft, ob ein Gameobject die Sicht auf den Charakter versperrt. Daraufhin wird die Kamera entsprechend bewegt, um eine bessere Sicht zu ermöglichen.

Unsere Priorität ist, den Charakter immer auf einer angenehmen Blickdistanz zu halten. Deshalb erfolgt die Kollisionsvermeidung per Höhenverstellung der Kamera.

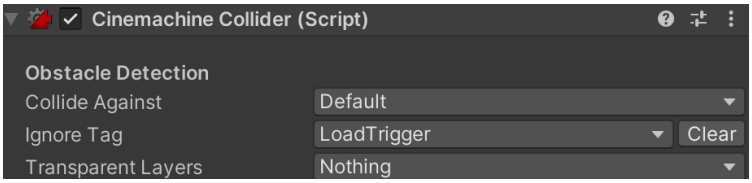

Abbildung 23: Cinemachine Collider

5.5 Wetter-System

Um dem Nutzer ein abwechslungsreiches Lauferlebnis zu bieten, verwenden wir ein dynamisches Wettersystem. Da ein solches System sehr aufwendig ist, verwenden wir das fertige Unistorm Wettersystem. In diesem Abschnitt beschreiben wir die Konfiguration dieses Systems.

5.5.1 Unistorm

Wie zuvor schon beschrieben, ist Unistorm ein Wettersystem, mit dem sich Wettersituationen einfach konfigurieren lassen. Von einfachen Wetterverläufen bis hin zu komplexen, zeitabhängigen Konfigurationen ist vieles möglich. Wir verwenden dabei nur einen Bruchteil der Funktionen.

Charakter Tracking

Um nicht unnötig Rechenleistung zu verbrauchen, wir das Wetter nur in einem bestimmten Bereich berechnet. Dieser Bereich umgibt den Charakter des Nutzers in Form einer Sphäre. Da der Charakter ständig seine Position wechselt, muss sich auch die Sphäre mitbewegen. Dazu verwenden wir das in Unistorm eingebaute Tracking-Tool. Dadurch ist es möglich, Unistorm ein Zielobjekt vorzugeben, welchem gefolgt wird.

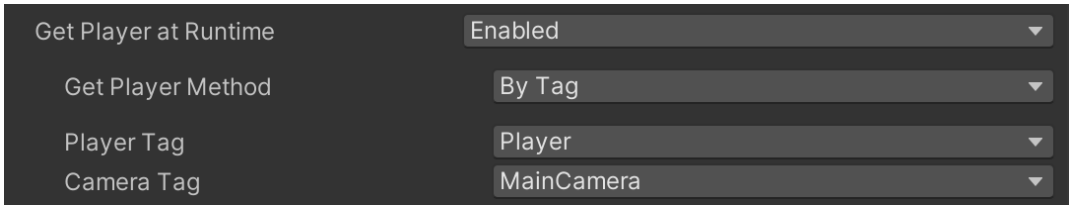

Abbildung 24: Charakter Tracking

Einstellungen

Unistorm enthält eine Vielzahl von Einstellungsmöglichkeiten. Dabei gibt es eine Voreinstellung für Desktop-PCs und Mobile Geräte. Wir verwenden aus Performance-Gründen die Mobile Konfiguration, worauf wir im Performance Abschnitt genauer eingehen. Um auch

Läufe in der Nacht zu ermöglichen, verwenden wir entsprechende Helligkeitseinstellungen. Mit unseren Einstellungen kommen wir schlussendlich zu einem zufriedenstellenden Ergebnis.

5.5.2 Skybox

Skybox Material

Die Skybox bestimmt das Aussehen des Himmels. Es ist ein Material, das überall dort zu sehen ist, wo keine Gameobjects platziert wurden. Da es ein Panorama darstellen muss, um realistisch zu wirken, sind diese sehr aufwendig zu erstellen.

Skybox Arten

Grundsätzlich wird zwischen Procedural und Cubemap Skyboxen unterschieden. Procedural Skyboxen werden zur Laufzeit generiert. Eine Cubemap hingegen verwendet eine festgelegte Textur. Wir verwenden die von Unistorm bereitgestellte Cubemap Skybox.

5.5.3 Wetter Situationen

Unistorm ermöglicht uns, einzelne Wettersituation selbst zu konfigurieren und zu speichern. Diese werden in einen Auswahlpool gelegt, aus dem Unistorm Wettersituationen wählt. Wir verwenden dabei eine zufällige Auswahl.

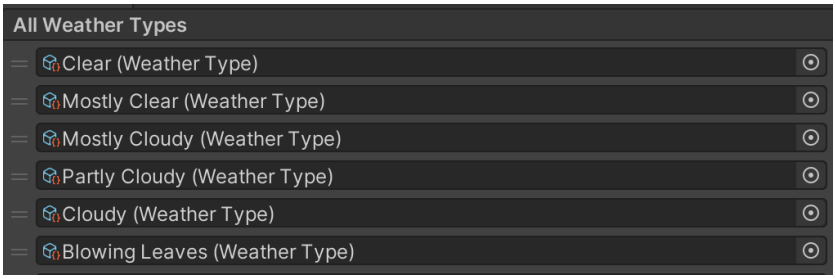

Abbildung 25: Wetter Situationen

5.5.4 Performance

Gute Performance ist eines der wichtigsten Aspekte bei der Entwicklung von TreadRun. Da realistische Wettereffekte äußerst ressourcenintensiv sind, geben wir hier besonders acht.

Unsere Performance Tests von TreadRun zeigen, dass die Desktop-Voreinstellungen von Unistorm nicht für dieses Projekt geeignet sind. Daher verwenden wir Einstellungen, die eigentlich für Mobilgeräte gedacht sind.

Da beim Laufen die Aufmerksamkeit des Nutzers hauptsächlich auf dessen Charakter liegt, fällt die niedrigere Qualität des Wetters kaum auf. Auf manche Einstellungen wie Sonnenstrahlen wollen wir jedoch nicht verzichten. Deshalb priorisieren wir einige Einstellungen, die wir höher setzen.

5.6 Shader

Wie zuvor beschrieben, sind Shader kleine Programme, die Informationen in der Grafikpipeline manipulieren. In Unity gibt es bereits Standard Shader die wir auch verwenden. Für Charakter, Gebäude und Bäume verwenden wir jedoch spezielle Cel-Shader.

5.6.1 Shader Arten

Grundsätzlich wird zwischen zwei Arten von Shadern unterschieden: Pixel Shader und Vertex Shader.

Für unseren Cel-Shading Effekt nutzen wir einen Surface Shader. Diese beinhalten sowohl Elemente eines Pixel- als auch Vertex Shaders. Vorteilhaft dabei ist, dass Unity selbst den Code für die Beleuchtung des Materials generiert.

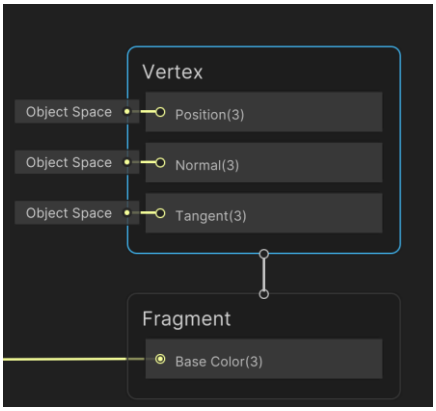

Abbildung 26: Shader Template

Pixel Shader

Da nur ein Fragment des Bilds bearbeitet wird, wird dieser auch oft als Fragment Shader bezeichnet. Es wird die Farbe jedes einzelnen Pixels ausgegeben. Die richtige Farbe wird mithilfe von Daten der Texturen, Beleuchtung und Schatten berechnet.

Vertex Shader

Ein Vertex Shader hat vielseitige Einsatzmöglichkeiten, wir verwenden ihn jedoch nur für die Manipulation von Beleuchtungs- und Farbzusammensetzungen.

Mit Vertex Shader werden die Eigenschaften von sogenannten Vertices manipuliert. Für bessere Effekte werden die Ausgaben des Vertex Shaders im Pixel Shader weiterverarbeitet.

5.6.2 Cel-Shader

Eigenschaften

Cel-Shading ist eine nicht realistische Art von Rendering, die dem Objekt einen speziellen Comic-Look verleiht. Technisch wird dies durch eine Änderung Farb- und Schattendarstellung erzielt.

In einem realistischen Stil gehen Farben sowie Schatten in einem Verlauf ineinander über. Dies ist nicht der Fall bei Cel-Shading. Licht wird in klaren Blöcken gerendert, was zu einem Comic-Look führt.

Wir manipulieren auch die Farbübergänge. Statt glatten Übergängen gehen die Farben abrupt ineinander über. So erreichen wir Übergänge, die unsere Objekte grober aussehen lassen.

Vorteile

Wir verwenden einen Cel-Shading Stil, da uns nicht die Daten zur Generierung einer fotorealistischen Welt zur Verfügung stehen. Der Stil gibt uns die Möglichkeit, einfachere Texturen und Modelle zu nutzen.

Der Grafikstil bringt außerdem eine enorme Performance Verbesserung mit sich. Uns ist es dadurch möglich, das Programm auf Zielsystemen mit niedrigeren Spezifikationen zu betreiben.

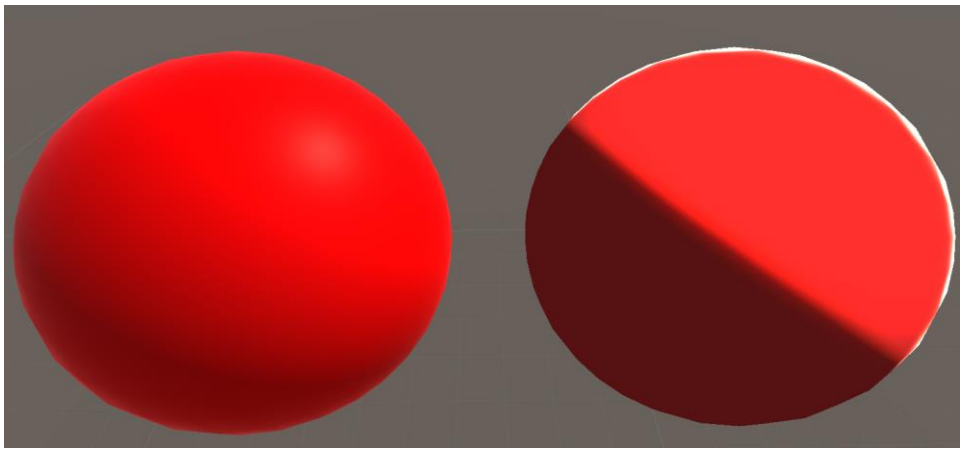

Abbildung 27: Vergleich: ohne- und mit Cel-Shader

5.6.3 Beleuchtung

Beim Aufbau eines Shaders müssen alle möglichen Lichtquellen beachtet werden. Wir müssen dabei für jede Lichtquelle einen eigenen Codeabschnitt schreiben. In diesem Abschnitt beschreiben wir die verwendeten Lichtquellen. Die entsprechenden Programme werden im nachfolgenden Abschnitt beschrieben.

Ambient Light

Das Ambient Light ist die indirekte Lichtquelle, die auf alle Gameobjects in einer Szene wirkt. Es ist also ein globaler, konstanter Wert. Über diesem Wert werden andere Lichtquellen überlagert.

Diffuse Light

Das Diffuse Light wird nur durch den Winkel zwischen einer Lichtquelle und Oberfläche beeinflusst. Daher können wir aus dem Skalarprodukt vom Normalvektor einer Stelle und dem Richtungsvektor der Lichtquelle, das Diffuse Light berechnen.

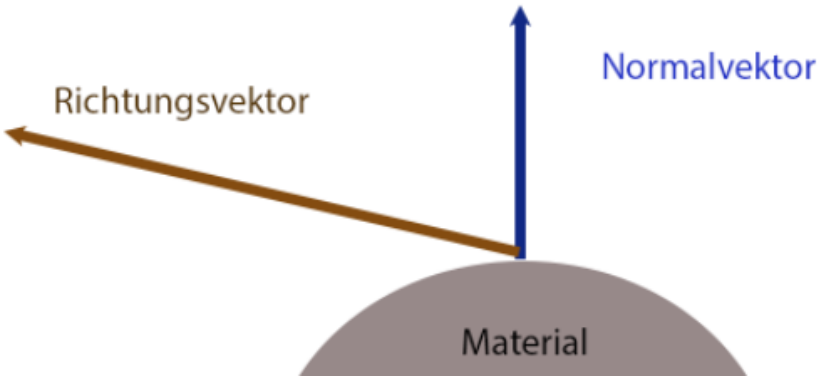

Abbildung 28: Diffuse Light Elemente

Specular Light

Das Specular Light ist abhängig von der Ansichtsseite und der Lichtquelle. Es ist ein besonders helles Licht, das nur von einer bestimmten Ansicht bemerkbar ist. Berechnet wird dies mit dem Normalvektor der Oberfläche, Richtungsvektor der Ansicht und dem Richtungsvektor der Lichtquelle.

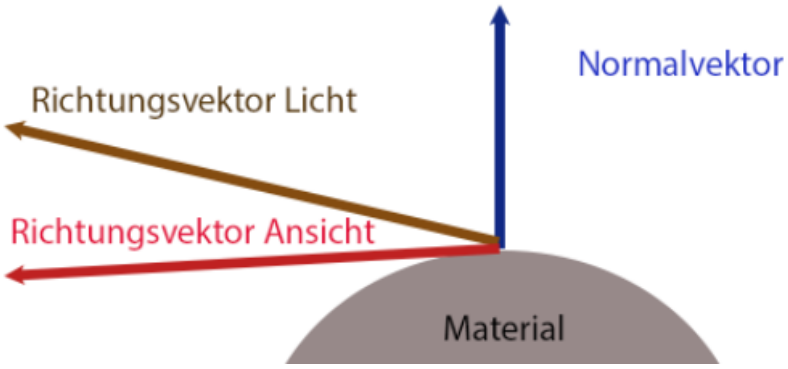

Abbildung 29: Specular Light Elemente

Fresnel Light

Das Fresnel Light erzeugt einen sogenannten „Bleeding“ Effekt. Also eine hellere Beleuchtung an den Kanten von Objekten. Der Effekt sieht besonders bei Cel-Shadern gut aus. Diesen Effekt berechnen wir aus dem Richtungsvektor der Ansicht und einem Normalvektor.

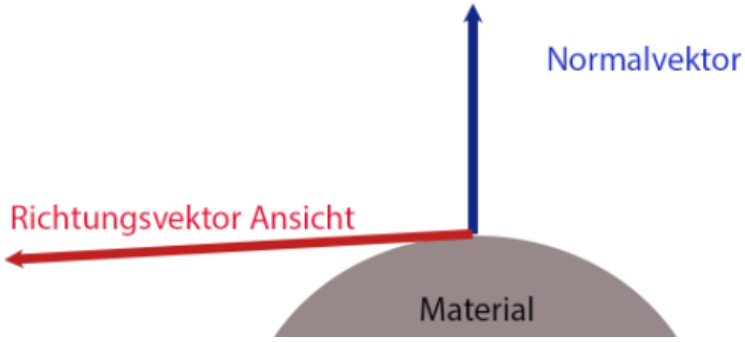

Abbildung 30: Fresnel Light Elemente

5.6.4 Programmierung der Shader

5.6.4.1 Ambient Light Implementierung

Zur Berechnung des Ambient Light kommen wir nur mit Codeblöcken aus. Im Shadergraph gibt es dazu bereits einen fertigen Ambient Codeblock. Wir legen mit einem Albedo Block die Grundfarbe des Ambient Light fest. Die Ausgabe dieser Codeblöcke liefert dem Shader die Daten für das Ambient Light.

5.6.4.2 Diffuse Light Implementierung

Richtungsvektor der Lichtquelle

Für diese Implementation benötigen wir zunächst den Richtungsvektor der Lichtquelle. Dabei kommen wir nicht daran vorbei, HLSL Code zu schreiben.

Wir verwenden eine Funktion, die die Position in der Welt übergeben bekommt. Von dieser bekommen wir einen Richtungsvektor sowie einen Farbwert. In der Funktion werden Schattenwerte im Bezug zur Position geprüft. Dann werden die Variablen auf die entsprechenden Werte gesetzt und ausgegeben.

Diffuse Light Berechnung

Das Diffuse Light berechnen wir aus dem Skalarprodukt des Richtungsvektors der Lichtquelle und eines Normalvektors. Zur Berechnung des Skalarprodukts gibt es im Shadergraph einen eigenen Codeblock. Diese werden entsprechend verbunden, um den richtigen Wert zu liefern.

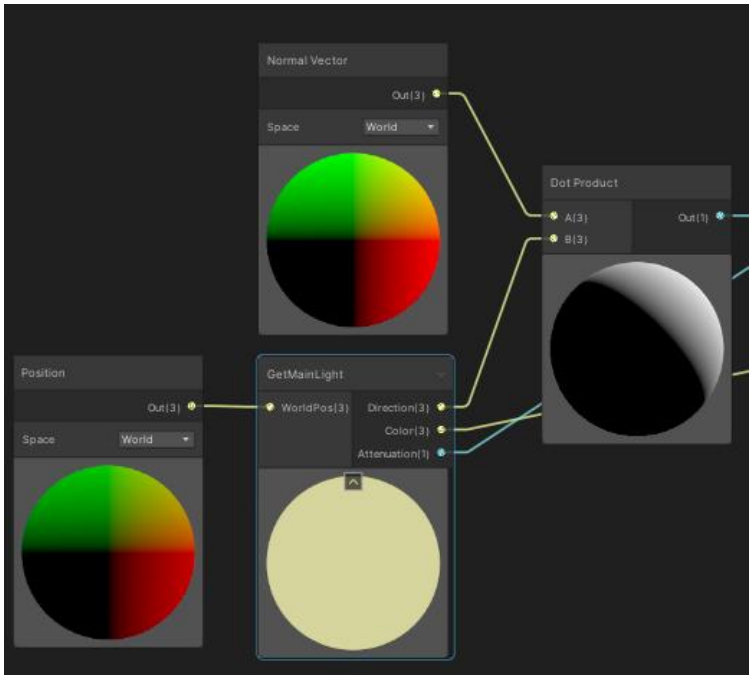

Abbildung 31: Diffuse Light im Shadergraph

Die neu geschriebene Funktion wird in den zuvor beschriebenen Ambient Light Code integriert. Da das Ambient Light überlagert werden muss, wird das Diffuse Light mit dem Eingang des Ambient Light multipliziert.

5.6.4.3 Specular Light Implementierung

Blickrichtungsvektor

Zur Berechnung des Specular Light brauchen wir die Blickrichtung. Diese bekommen wir durch einen bereits vorhandenen Codeblock. Da dies ein Richtungsvektor ist, normalisieren wir ihn. Wir setzen ihn also auf die Länge eins.

Specular Light Berechnung

Wir addieren den neuen Richtungsvektor mit dem Vektor der Lichtquelle. Der Vektor der Lichtquelle ist derselbe, der bei der Berechnung des Diffuse Light verwendet wird. Das Specular Light entsteht nun aus dem Skalarprodukt eines Normalvektors und der zuvor berechneten Summe.

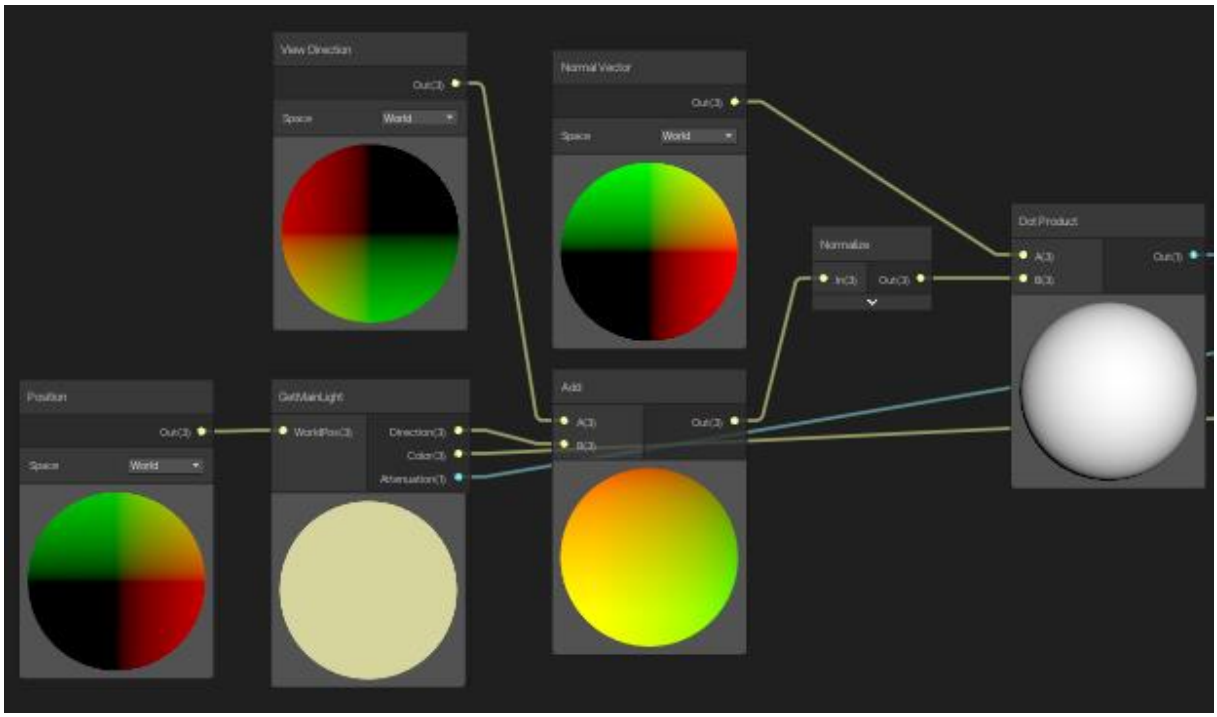

Abbildung 32: Specular Light im Shadergraph

Um das Specular Light besser sichtbar zu machen, verstärken wir es. Dies erreichen wir mit einem Power Codeblock, der als Verstärker dient. Dabei legen wir einen maximalen Verstärkungswert fest. Zur Integration in den Shader addieren wir den Ausgangswert von Specular Light zum Diffuse Light.

5.6.4.4 Fresnel Light Implementierung

Der Effekt des Fresnel Light ist nur aus einem flachen Winkel sichtbar. Die Implementation von Fresnel Light ist im Gegensatz zum Diffuse und Specular Light sehr einfach. Im Shadergraph gibt es für Fresnel Light bereits einen fertigen Codeblock. Wir müssen diesem nur einen Normalvektor vorgeben und Unity erledigt den Rest.

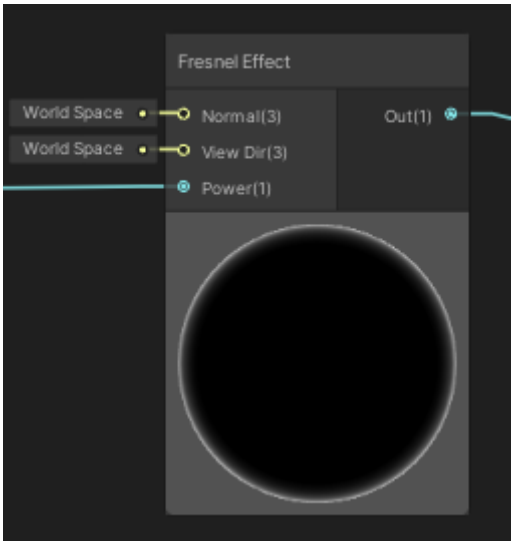

Abbildung 33: Fresnel Light im Shadergraph

6 Anwenderfall TP

In diesem Abschnitt beschreiben wir einen vollständigen Anwendungsfall.

Wir gehen von einer neuen Installation auf Windows aus. Man benötigt das installierte Programm und den Sensor mit 5 weißen Stickern.

Vorbereitung des Laufbands

Zuerst müssen die Sticker auf dem Laufband in regelmäßigen Abständen platziert werden. Danach muss das Lesegerät so platziert werden, dass dieses mit dem Fotosensor die Sticker erkennen kann. Dies geht an jedem Ort des Laufbands, empfohlen wird aber am Ende des Bands.

Damit das Programm den Sensor erkennt, muss am Computer ein Hotspot gestartet werden. Dieser sollte den Namen „TreadRun“ und als Passwort „TreadSense“ verwenden, da ansonsten der Sensor nicht erkannt werden könnte.

Danach wird der Sensor mit Strom versorgt und die Einrichtung des Laufbands ist damit abgeschlossen.

Programmstart

Wenn das Programm das erste Mal gestartet wird, ist die Empfehlung, sich mit Strava anzumelden, um alle Trainings zu speichern. Dies kann über die Usereinstellungen oben rechts erfolgen. Wenn man sich dazu entscheidet, sich über Strava anzumelden, wird der Browser geöffnet und man kann sich bei Strava einloggen. Die Benutzerdaten werden gespeichert, bis man sich abmeldet.

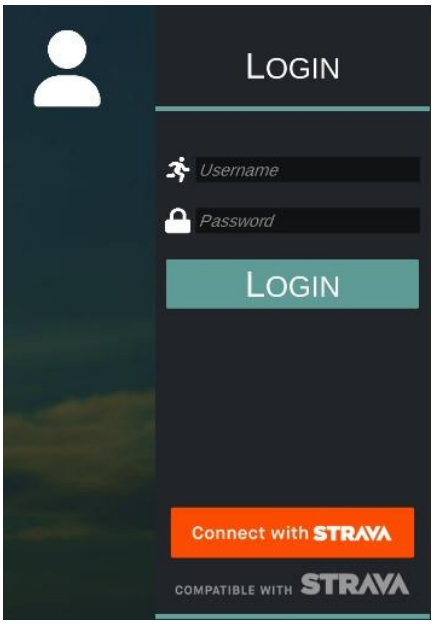

Abbildung 34: Connect with Strava

Sensor verbinden

Um den Sensor zu verbinden, klickt man auf die Schaltfläche unten rechts. Dadurch wird der Sensor im Hotspot gesucht und sich mit ihm verbunden. Wenn ein Fehler beim Verbinden auftreten sollte, wird eine Fehlernachricht angezeigt.

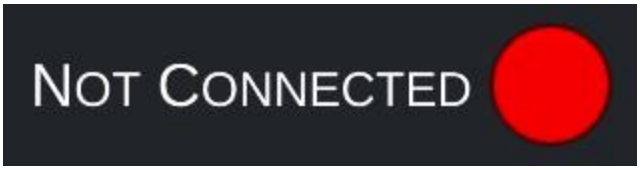

Abbildung 35: Connect sensor

Kalibration

Wenn der Sensor erfolgreich verbunden wurde, muss er kalibriert werden. Dazu muss das Laufband auf 5 km/h beschleunigt und auf 0° Steigung eingestellt werden. Nun wird mit einem weiteren Klick auf die untere rechte Schaltfläche wird die Kalibration gestartet. Falls Fehler auftreten sollten, wird eine Fehlermeldung angezeigt. Diese beinhaltet den genauen Grund, warum die Kalibration fehlerhaft ausgeführt wurde.

Streckenauswahl

Erst wenn ein Sensor verbunden und kalibriert ist, kann man das Training beginnen. Wenn kein Sensor verbunden ist, wird eine Fehlermeldung eingeblendet.

Mit einem Klick auf „Saved Routes“ hat man eine Auswahl aus allen gespeicherten Strecken. Ein Klick auf eine dieser Strecken lädt die Strecke und man befindet sich im Spiel.

Oder mit einem Klick auf „Play“ wird das Kartemenü geladen. Auf der Karte kann mit Doppelklick der Startpunkt gesetzt werden. Mit jedem weiteren Doppelklick wird das Ende der Strecke gesetzt. Jeder bereits vorhandene Endpunkt wird zu einem Zwischenpunkt.

Auf der linken Seite gibt es Schaltflächen, die verschiedene Funktionen besitzen.

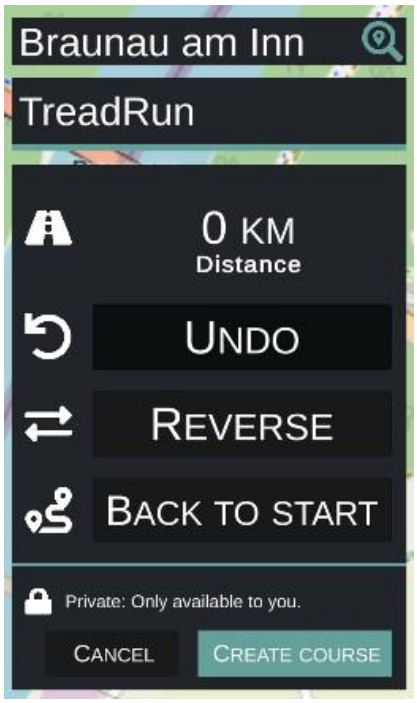

Abbildung 36: Map GUI

Suchfeld

Das Suchfeld kann dazu benutzt werden, um schnell zwischen verschiedenen Orten der Welt zu wechseln. Man gibt zum Beispiel „Salzburg“ ein, drückt Enter, und die Karte zentriert sich in Salzburg.

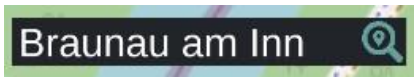

Abbildung 37: Map GUI - Suchfeld

Name

Eine Strecke, welche man gerne speichern möchte, muss man benennen. Der Name wird in das Namensfeld eingetragen. ACHTUNG, dieser Name ist nicht mehr änderbar.

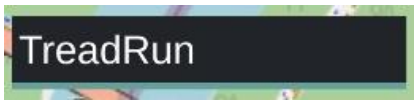

Abbildung 38: Map GUI - Name

Undo

Der Undo-Knopf setzt alles auf die vorherige Änderung zurück.

Abbildung 39: Map Gui - Undo

Reverse

Der Reverse-Knopf sorgt dafür, dass die Strecke in die entgegengesetzte Richtung verläuft.

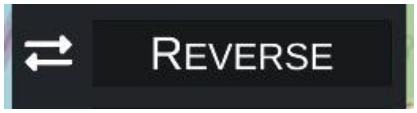

Abbildung 40: Map Gui - Reverse

Back to Start

Damit wird das Ende auf die Startposition gesetzt. Es ist nicht gegeben, dass dies eine Schleife erzeugt.

Abbildung 41: Map Gui - Back to Start

Mit einem Klick auf „Create Course“ wird der Start der Strecke generiert. Falls der Benutzer keinen Namen eingegeben hat, erscheint eine Warnung, ob der Benutzer, ohne zu speichern fortfahren möchte.

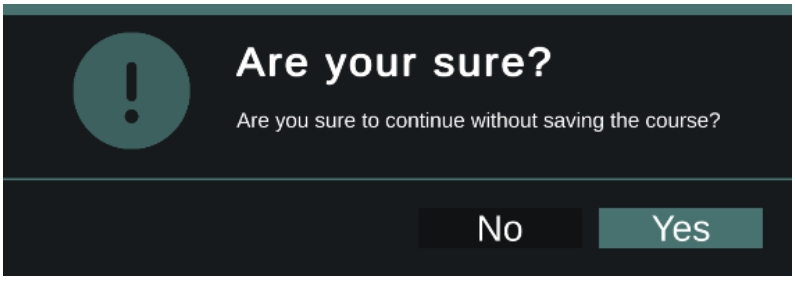

Abbildung 42: Warnung

Laufen

Sobald der Start des Kurses fertig generiert ist, wird der Sensor aktiv und überträgt die momentane Geschwindigkeit und Steigung. Außerdem startet die Zeitnehmung.

Nun wird der Spieler mit der Geschwindigkeit des Laufbands fortbewegt und Häuser, Straßen und Bäume werden fortlaufend generiert. Außerdem wird laufend die aktuelle Geschwindigkeit und die zurückgelegte Strecke aktualisiert und angezeigt.

Das Ende der Strecke wird klar sichtbar gemacht, um den Spieler zu motivieren, auch noch die letzten Meter durchzuhalten.

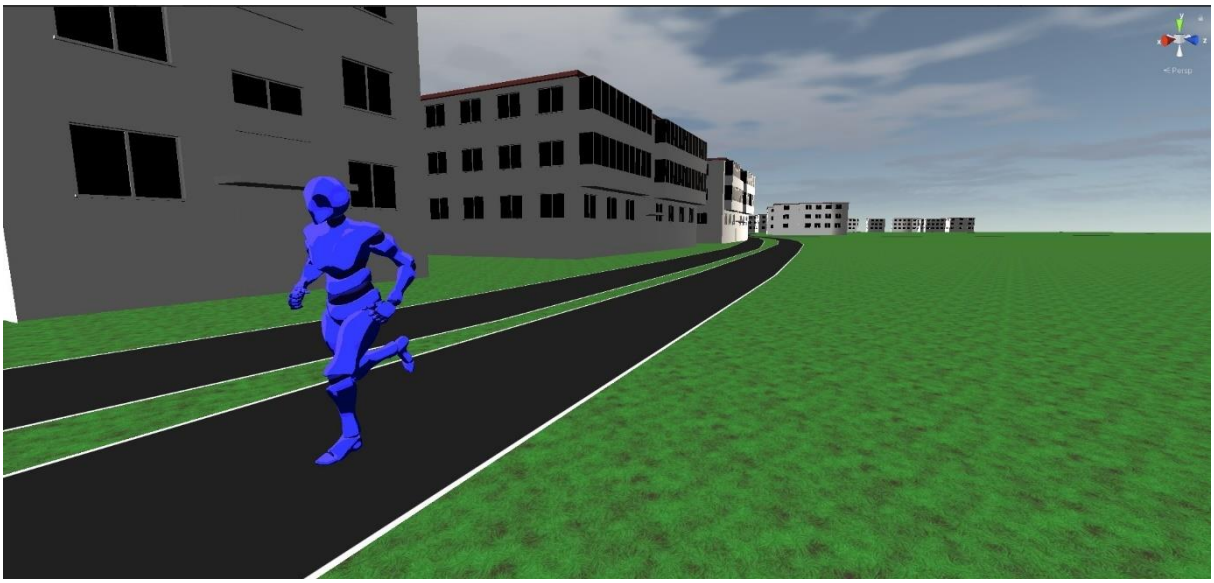

Abbildung 43: In-game running

Ende

Am Ende eines Laufes sieht man seine Statistiken wie Dauer, Distanz und Pace. Wenn der Benutzer mit Strava verbunden ist, wird die Aktivität auf Strava hochgeladen. Mit einem Klick auf „Done“ kommt man in das Hauptmenü und kann noch eine Runde laufen oder das Programm beenden.

Mögliche Fehlerquellen

Verbindungsaufbau:

- Der Sensor ist möglicherweise nicht mit dem Hotspot verbunden. Wenn dieser Fehler auftritt, den Sensor vom Stromnetz nehmen und 30 Sekunden warten, bevor man ihn wieder anschließt.
- Der Hotspot wurde falsch benannt oder das Passwort falsch geschrieben.

Kalibration:

- Wenn das Laufband nicht eingeschaltet ist oder der Sensor die Sticker nicht erkennt, wird nach 30 Sekunden der Kalibrationsversuch abgebrochen.

- Der Sensor erkennt die Sticker nicht korrekt. An einer LED am Sensor kann man sehen, ob der Sensor die Sticker erkennt. Der Sensor und die Sticker sollten so platziert werden, dass jeder Sticker erkannt wird.

Generierung:

- Eine schlechte oder keine Internetverbindung kann dazu führen, dass die Welt nicht mehr generiert wird.
- Wenn die Region der Strecke auf OpenStreetMap nicht gut gemappt wurde, wird nichts generiert.

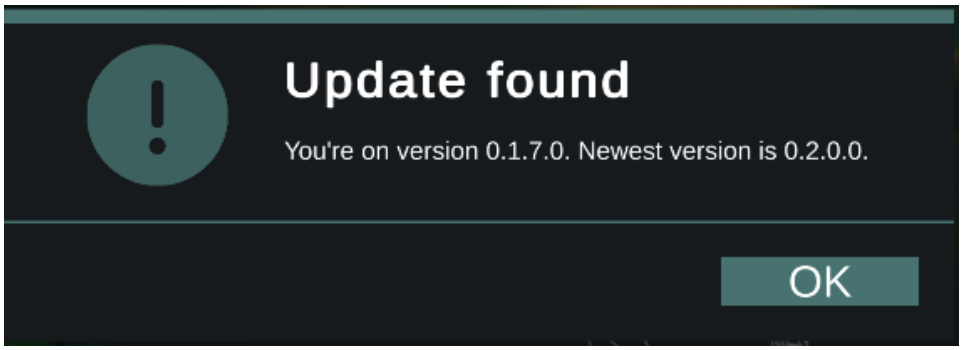

Abbildung 44: Update Benachrichtigung

7 Projekt Management TP

Dieses Kapitel veranschaulicht das Projektmanagement unserer Arbeit. Wir befassten uns mit verschiedenen Seiten des Projektmanagements.

Wir arbeiteten nach der SCRUM Methode. Jeder von uns hatte seine fest zugeteilten Aufgaben und sollte diese in einer Woche lösen. Nach jeder Woche wurden die neuen Features kurz angesehen und besprochen, ob und wie man diese verbessern könnte.

Um einen Überblick über alle Aufgaben zu erlangen, wurde mit dem Ticketsystem Jira gearbeitet. Jira ermöglichte uns einen Einblick auf unseren Fortschritt und zeigte auf wo Zeit liegenblieb.

7.1 Planung

13.09.2021	Einarbeitung in die verwendeten Technologien
31.10.2021	Erster Prototyp mit grundsätzlicher Funktionalität
03.12.2021	Erste Demoversion für den Tag der offenen Tür
31.01.2022	Fertigstellung der Gesamtfunktionalität
28.02.2022	Abschließende Tests sowie Fehlerbehebungen

Die Meilensteine wurden so gewählt, um schulische Termine zu berücksichtigen.

Jedes Teammitglied wurde ein eigener Aufgabenbereich zugeteilt, an dass dieses gearbeitet hat. Das ermöglichte eine fast ungehinderte Produktion des Projekts.

7.2 Evaluation

Dank der SCRUM-Methode lagen wir gut in der Zeit und erreichten jeden Meilenstein knapp eine Woche früher als geplant. Die Extrazeit wurde genutzt, um eventuelle Fehler oder Designentscheidungen auszubessern und zu überarbeiten.

Da wir jede Woche neue Aufgaben zugeteilt bekommen haben war es auch immer abwechslungsreich, was die Motivation stärkte am Projekt weiterzuarbeiten.

Wir bemerkten ebenfalls das Ausmaß dieses Projekts und mussten an manchen Stellen, Abstriche tätigen, um in der Zeit fertig zu werden.

Nicht so gut lief die strikte Einhaltung der Aufgaben da es manchmal dazu kam das wir an etwas anderem gearbeitet haben als wir zu diesem Zeitpunkt sollten. Somit wurden kritische Punkte viel zu spät bearbeitet und führten zu mehr Problemen als nötig gewesen wären.

7.3 Zeiterfassung

7.3.1 Thomas Schinwald

Datum	Uhrzeit von/bis	Arbeits-Stunden	Arbeitsprozess Thema, Tätigkeit, Arbeitsschritte, Arbeitsablauf, evtl. Koordination externer Partner	Anmerkungen, Hilfsmittel und Hilfestellungen durch Betreuungslehrer/in bzw. externe Berater
	IS EGAL EINFACH FREI LASSEN			

…….……..... …….……

Ort und Datum Vorname Nachname

7.3.2 Tobias Pieber

Datum	Uhrzeit von/bis	Arbeits-Stunden	Arbeitsprozess Thema, Tätigkeit, Arbeitsschritte, Arbeitsablauf, evtl. Koordination externer Partner	Anmerkungen, Hilfsmittel und Hilfestellungen durch Betreuungslehrer/in bzw. externe Berater
	IS EGAL EINFACH FREI LASSEN			

…….……..... …….…….

Ort und Datum Vorname Nachname

8 Zukünftige Arbeit TP

In diesem Kapitel behandeln wir die weiterführenden Arbeiten an dem Projekt.

8.1 Performance Update

Um TreadRun auf jedem Computer mit guter Performance zum Laufen zu bringen, muss noch einiges an der Performance des Programms gearbeitet werden. Zu diesem Zeitpunkt werden noch alle Features geladen, sobald diese auf einem Tile platziert sind. Das muss sich ändern zu Streaming, das heißt die Features erst in den Arbeitsspeicher laden und erst wenn diese benötigt werden, sie zu aktivieren. Auch müssen LOD Modelle der Gebäude, welche weiter weg sind, generiert werden, um das Mesh so klein wie möglich zu halten.

Außerdem sollten Nodes, welche sich nicht auf einem Tile befinden, von der Overpass QL-Query abgeschnitten werden. Dies erspart die doppelte Generierung von Features.

Jedes Feature, welches nicht dynamisch generiert wird, sondern ein Prefab benutzt, bekommt einen eigenen Object Pool, um auch hier zu verhindern, viele Objekte zu erstellen.

Generierung:

Das Terrain soll auch in 3D sein, um Hügel und Berge zu generieren und zu simulieren.

8.2 Mehr Features

Es gibt eine Unmenge an Elementen auf der OpenStreetMap. Wir haben davon vier implementiert. Hier wollen wir mehr Features einbauen. Als nächstes Feature ist der Wald geplant.

8.2.1 Mod Support

Da wir nicht jedes Element der OpenStreetMap implementieren können, dürfen Spieler, die ihre eigenen Features in das Spiel einbinden möchten, das tun, indem sie Modifikationen, kurz

Mods, entwickeln. Diese können von jedem Benutzer heruntergeladen werden und selbst in ihr Spiel eingefügt werden.

Dazu entwickeln wir eine SDK, mit der man ein Feature programmieren kann und es als DLL-Datei exportiert. Diese Datei wird ein einen speziellen Ordner abgelegt, sodass unser Modsystem dieses erkennt und einbinden kann.

8.3 Optionen

Um dem Benutzer eine möglichst große Freiheit bei der Gestaltung der Spielweise, zu ermöglichen, werden wir Optionen implementieren. Es wird die allgemeinen Einstellungen, wie Grafik- und Audioeinstellungen geben, sowie spezielle Einstellungen.

Kamera:

Will der Benutzer seinen Charakter nicht sehen, sondern eine Firstperson-Ansicht nutzen, kann er die Ansicht umschalten.

Features:

Nicht jeder hat einen leistungsstarken Computer, der viele Features noch ordentlich Rendern kann, ohne Probleme zu bekommen. Deshalb sollte der Benutzer jedes der nicht essenziellen Features deaktivieren können.

Heads Up Display:

Nicht jeder möchte Statistiken sehen und andere möchten so viele wie möglich sehen. Aus diesem Grund haben wir beschlossen, auch dies einstellbar zu machen. Das HUD wird modular gestaltet, um jeden Benutzer sein individuelles HUD zu ermöglichen.

Strava:

Wenn man mit Strava angemeldet ist, aber nicht möchte, dass die Aktivität hochgeladen wird, wird man es ausschalten können.

8.3.1 Charakterbuilder

Jeder Spieler sollte sich individuell über ihren Charakter identifizieren können. Deshalb integrieren wir einen Charakterbuilder. Der Charakterbuilder wird viele Kleidungsmöglichkeiten bieten. Mithilfe von Punkten kann der Spieler auch Ausrüstungsgegenstände kaufen. Wie man Punkte erhält, wird in der „Multiplayer“ Sektion näher erläutert.

8.1 Multiplayer

Beim Laufen ist es wichtig, auch hin und wieder in einer Gruppe zu sein, um sich mehr zu pushen. Aus diesem Grund wollen wir einen Multiplayermodus einfügen.

Der Multiplayermodus wird aus verschieden Räumen mit bis zu 16 Spielern bestehen. Die Verbindung wird auf einer Peer-To-Peer Verbindung bestehen, wobei ein Spieler den Raum hostet.

Ein öffentlicher Server, der dauerhaft Online ist, wird von uns gehostet und verwaltet. Dieser dient für Bewerbe, Veranstaltungen und als Treffpunkt für die Benutzer.

Jeder Spieler benötigt die Mods des Hosts. Wenn dies nicht der Fall ist, werden nur essenzielle Features generiert.

Vereine:

Jeder Spieler soll einen eigenen virtuellen Verein gründen können. Jeder Verein wird Punkte bekommen für jeden Spieler, der in einem Verein an Bewerben teilnimmt. Mit diesen Punkten soll man virtuelle Trikots und andere Dinge für alle Vereinsmitglieder kaufen können.

Wettbewerbe:

Wettbewerbe sind virtuelle Läufe auf offiziellen Marathonkursen, aber auch auf erdachten Kursen. Es wird keine Altersklasseneinteilung geben, sondern nach einem Level, welcher angibt wie schnell ein Spieler ist. Dadurch soll das Feld gut ausgeglichen sein. Jeder Spieler erhält für die Teilnahme Punkte.

Achievements:

Für verschiedene Aktivitäten bekommt ein Spieler Punkte. Darunter fallen auch Aufgaben, zum Beispiel der erste Aufenthalt in einer größeren Stadt oder bestimmte Kilometermeilensteine.

8.2 Website

Auf der Website sollten alle Events, Spieler, und Pace Partner in einer Karte angezeigt werden.

Pace Partner:

Pace Partner sind vom Computer gesteuerte Spieler, welche mit einer konstanten Geschwindigkeit laufen. Pace Partner werden auf der ganzen Welt verteilt und man ihnen folgen. Sie werden durch die schönsten Regionen laufen und auch im Singleplayer-Modus erscheinen.

8.3 Grafische Überarbeitung

Eine grafische Überarbeitung ist wichtig, um auch schwächere PCs zu unterstützen.

Auch soll die Qualität der Texturen sich verbessern, da alles eher plastisch aussieht. Mit stilistischen Texturen würde es besser aussehen. Auch die Beleuchtung muss dazu geändert werden.

8.4 Fahrräder

Es ist denkbar, nachdem das Programm funktioniert, neben Läufer auch Fahrradfahrer einzubinden. Dadurch wird eine größere Zielgruppe angesprochen und man benötigt nicht zwei verschiedene Programme, wenn man zum Beispiel Triathlon betreibt und sowohl Laufen als auch Fahrradfahren trainiert.

Um Fahrräder unterstützen zu können, müsste ein neuer Sensor entwickelt werden, beziehungsweise Drittanbietergeräte benutzt werden.

9 Verwandte Arbeiten TP

Dieses Kapitel befasst sich mit Projekten die dieselben Technologien wie unser Projekt benutzen oder sich ein einer Weise ähnlich sind.

9.1 Zwift

Zwift ist ein Indoor-Trainings-Programm, das neben Radfahren auch Laufen unterstützt. Zwift hat nur vorgefertigte virtuelle Welten. Diese werden von Hand entworfen und gebaut. Zwift erlaubt in diesen vorgefertigten Welten freie Bewegung auf den Wegen und Sportzentren. Mit über 2 Millionen Nutzer wird auch niemanden langweilig da es in Zwift nur einen Onlinemodus gibt.

Der Unterschied liegt darin das Zwift zwar ihre Welten nicht dynamisch generiert, aber dafür grafisch viel mehr zu bieten hat, da die Welten von Hand gebaut werden.

9.2 RGT - Magic Roads

RGT Cycling ist ein Indoor-Trainings-Programm für Radfahrer und hat virtuelle aber auch Videostrecken im Angebot. Eine Besonderheit ist aber Magic Roads. Magic Roads bieten die Möglich auf jeder Straße der Erde zu trainieren. Dazu muss man eine GPX-Datei mit der eingefahrenen Strecke an eine E-Mail-Adresse senden. Die Strecke wird danach erstellt und man kann sie in der Streckenliste auswählen und abfahren. Steigungen werden berücksichtigt und eine kleine Landschaft um die Straße herum wird erzeugt

Der große Unterschied zu unserem Projekt liegt darin das Magic Roads nicht dynamisch generiert wird. Ebenso werden keine Häuser erzeugt, sondern eine fiktive Landschaft, die aber alle Höhenunterschiede berücksichtigt.

9.3 MapBox Game SDK

Die Mapbox Game SDK bietet eine Vielzahl an Anwendungen. Wir beschäftigen uns nur mit dem „Location based games“ Feature. Dieses Feature kann dynamisch Gebäude, Straßen

und andere Dinge generieren. Höhenunterschiede werden berücksichtigt und man kann die Stile der generierten Gebäude und allem anderen einstellen. Echtzeit Verkehrsdaten können auch berücksichtigt werden.

Der einzige Unterschied liegt darin, dass Mapbox die Generierung nicht in Tiles unterteilt, sondern vom Spieler aus in jede Richtung eine gewisse Weite generiert.

9.4 Runn™

Der Runn™ Smart Laufbandsensor von North Pole Engineering ist ein Geschwindigkeitssensor für ein Laufband.

Der Runn™ lässt sich ganz einfach an den seitlichen Schienen der meisten Laufbänder installieren. Indem ein optischer Geschwindigkeitsmesser zur Messung der Bandlaufgeschwindigkeit zum Einsatz kommt, übermittelt dieses Gerät die Daten über BLE und ANT+. [4]

Die Unterschiede zwischen Runn™ und TreadSense zeigen sich in der Kalibrierung und der Übertragungsmethode. Während Runn™ direkt am Gerät einen Knopf zum Kalibrieren besitzt, kann TreadSense auch über den PC kalibriert werden. TreadSense überträgt die gesammelten Daten über TCP und nicht Bluetooth LE.

10 Fazit TS

Unsere Lösung zu einem Indoor-Trainingssystem wird durch mehrere Komponenten ermöglicht. Ziel ist, dem Nutzer eine dynamische Laufkurswahl zu ermöglichen. Diesen Kurs kann der Nutzer durch die Steuerung mit seinem Laufband ablaufen. Wir haben unser Ziel erreicht und unsere Lösung in dieser Arbeit dokumentiert.

Aus unserer Arbeit lässt sich schließen, dass ein virtueller Nachbau der Welt möglich ist, sofern die nötigen Daten vorliegen. Wir beziehen unsere Daten aus der OpenStreetMap, die eine riesige Menge von Daten liefert. Diese sind jedoch oft unvollständig, was uns zu Improvisationen zwingt. Die offensichtliche Lösung, die wir auch verwenden, ist eine Vereinfachung der virtuellen Welt. Dadurch ist es uns möglich, sonst offensichtliche Probleme zu verbergen.

Diese Arbeit hat uns die Wichtigkeit von weitgehender Planung gezeigt, besonders im Bereich der Optimierung. Das Projekt soll auf einer weiten Spanne von Geräten betrieben werden können, was sich als eine große Hürde herausstellt. Bei der Generation einer virtuellen Welt müssen unzählige Elemente berechnet und geladen werden. Die nötige Anpassung des Projekts erreichen wir mit speziellen Lade- und Entladetechniken sowie einem vereinfachten Kursdesign. Mit unserer Lösung ermöglichen wir somit einen Betrieb auf einer Vielzahl von Geräten.

Diese Arbeit gibt uns einen praktischen Einblick in den Entwicklungsprozess mit Unity, darunter die vielen Möglichkeiten und die technischen Limitationen. Unsere Lösung bietet viel Potenzial für zukünftige Erweiterungen und könnte die Basis für neuartige Projekte darstellen.

Acknowledgements

11 Abbildungsverzeichnis

12 Codeverzeichnis

13 Referenzen

[1] Kumi Systems, „Overpass by Kumi Systems,“ Kumi Systems, [Online]. Available: https://overpass.kumi.systems/. [Zugriff am 23 03 2022].

[2] Strava, „Strava Developers,“ [Online]. Available: https://developers.strava.com/docs/authentication/. [Zugriff am 23 03 2022].

[3] D. Basmanov, „GitHub,“ [Online]. Available: https://github.com/Syomus/ProceduralToolkit. [Zugriff am 23 03 2022].

[4] Zwift, „Runn™,“ [Online]. Available: https://www.zwift.com/eu-de/shop/product/runn. [Zugriff am 16 03 2022].

14 Team

14.1 Thomas Schinwald

Geburtsdatum, Geburtsort:	28.08.2002, Salzburg
Schulbildung	Volksschule Lochen Hauptschule Lochen HTL Braunau
Praktika	MIC
Anschrift	Fichtenweg 4 5221 Lochen am See Austria
E-Mail	thomas.schinwald@htl-braunau.at

14.2 Tobias Pieber

Geburtsdatum, Geburtsort:	23.07.2003, Salzburg
Schulbildung	Volksschule Schleedorf Sport-Mittelschule Seekirchen HTL Braunau
Praktika	Sofa 1
Anschrift	Oberharlochen 6 5231 Schalchen Austria
E-Mail	tobias.pieber@htl-braunau.at

REMOVE THIS PAGE BEFORE PRINTING!

Kapitel	**Name**	**Bearbeiter/in ✓ = Added names (short)**
	Abstract	Thomas Schinwald ✓
	Kurzfassung	Thomas Schinwald ✓
1.1	Problem Setting	Thomas Schinwald ✓
1.2	Problem Statement	Thomas Schinwald ✓
1.3	Inventions	Thomas Schinwald / Tobias Pieber ✓
1.4	Structure	Thomas Schinwald ✓
2.1	Software	Thomas Schinwald / Tobias Pieber ✓
2.2	Hardware	Tobias Pieber ✓
3	Hardware	Tobias Pieber ✓
4	Dynamische Generation/API	Tobias Pieber ✓
5	Gamedesign	Thomas Schinwald ✓
6	Case Study/Evaluation	Tobias Pieber ✓

7	Project Management	Tobias Pieber ✓
8	Future Work	Tobias Pieber ✓
9	Related Work	Tobias Pieber ✓
10	Conclusion	Thomas Schinwald ✓