h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

Hochschule Darmstadt

– Fachbereich Informatik–

Visualisierung von Fußgängerdaten in Virtual Reality

Abschlussarbeit zur Erlangung des akademischen Grades

Bachelor of Science (B.Sc.)

vorgelegt von

Marvin Lars Weisbrod

Matrikelnummer: 744411

25. Oktober 2022

Referent	:	Prof. Dr. Elke Hergenröther
Korreferent	:	M.Sc. Björn Frömmer

ERKLÄRUNG

Ich versichere hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe.

Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht.

Die Zeichnungen oder Abbildungen in dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen.

Diese Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Griesheim, 25. Oktober 2022

Marvin Lars Weisbrod

ABSTRACT

The technical progress in virtual reality in recent years has made it user-friendly and affordable enough for companies to evaluate possible applications in their workflows. VR has found uses in areas such as modelling and design, personnel-training or psychology and there are undoubtedly many more that will benefit from the widespread adoption of and research on VR that is currently underway.

One such area may be the visualization of crowds, where it can be very difficult to grasp the crampedness on a two dimensional screen. The development of a VR application for visualizing pedestrian data could facilitate the goal of improved insight into the dataset and may enable the user to analyze the dataset quicker and more thoroughly than with a traditional display.

This thesis deals with the implementation of such an application based on an existing non-VR project using the Unity engine. Popular locomotion methods and user interface variants are examined and combined into a concept, which is then implemented along with additional changes to the original project to facilitate a good user experience. Following this thesis, the resulting project can be customized further and evaluated for actual use.

ZUSAMMENFASSUNG

Technische Fortschritte haben den Einstieg in die virtuelle Realität so benutzerfreundlich und erschwinglich gemacht, dass viele Unternehmen mögliche Anwendungsbereiche in ihren Workflows untersuchen. Von Produktmodellierung und -design über Schulungen bis hin zur Präsentation dem Benutzer gegenüber gibt es viele Stellen, an denen die Nutzung von VR eine Verbesserung ermöglichen kann.

Ein solcher Bereich ist die Darstellung von Menschenmassen zu Evaluierungs- und Präsentationszwecken. Durch die Entwicklung einer VR Applikation zur Visualisierung von Fußgänger-Simulationsdaten, soll es ermöglicht werden, diese Daten in VR zu begutachten und dadurch einen besseren Einblick zu erhalten, als es mit einer traditionellen Anwendung möglich wäre.

Diese Arbeit beschäftigt sich mit der Implementierung einer solchen Anwendung auf Basis eines existierenden Nicht-VR Projektes mit Hilfe der Unity Engine. Es werden verschiedene Bewegungsverfahren und Optionen zur Bedienung erläutert und daraus ein Konzept formuliert. Dieses wird zusätzlich zu mehreren Anpassungen der Basisanwendung bis auf wenige optische Features implementiert und kann im Anschluss an die Arbeit für die Benutzung in Betrieben angepasst und evaluiert werden.

INHALTSVERZEICHNIS

ABBILDUNGSVERZEICHNIS

ABKÜRZUNGSVERZEICHNIS

6DOF	6 Degrees of Freedom
AR	Augmented Reality
FPS	Bilder pro Sekunde
HMD	Head Mounted Display
GUI	Graphical User Interface
MR	Mixed Reality
VR	Virtual Reality

1 EINLEITUNG

Der Markt für Virtual Reality (VR) in Hardware und Software hat in den letzten Jahren ein extremes Wachstum erlebt und VR-Brillen zugänglich für kleine Unternehmen und Privatkunden gemacht. Durch Fortschritte in Tracking- und Displaytechnologie sind diese Headsets mittlerweile eine interessante Option für ein breites Spektrum an Anwendungsbereichen, wie zum Beispiel psychologische Behandlungen, Schulungen oder Design und Kunst. Nun bietet es sich für viele Unternehmen an zu evaluieren, ob VR eine konstruktive Addition in ihrem Workflow darstellt. Das Projekt KapaKrit[13] beschäftigt sich mit der Optimierung von Personenströmen in Bahnhöfen im Katastrophenfall. Hierbei wird unter anderem JuPedSim[12] (Jülich Pedestrian Simulator) für die Simulation und Visualisierung verwendet, was allerdings nur eine rudimentäre Möglichkeit zur Visualisierung bietet, die aufgrund der Darstellung in 2D nicht immer einfach zu Bewerten ist. Daher soll eine möglichst ideale Beispielanwendung für virtual reality entwickelt werden. Diese soll dem JuPedSim Team die Möglichkeit geben, zu Analysieren, ob VR eine brauchbare Alternative zur Visualisierung ihrer Projektdaten darstellt. Die Benutzung von VR könnte für diesen Anwendungsfall eine bessere Übersicht bieten und es erleichtern, die Daten schnell und intuitiv zu betrachten.

Als Grundlage hierfür dient eine für JuPedSim angepasste Open-source Anwendung[22][3], welche ursprünglich von Daniel Büchele für eine Masterarbeit[4] erstellt wurde. Diese Anwendung zeigt mit Hilfe der Unity Engine[37] Simulationsdaten in 3D auf einem Monitor an. Um diese Anwendung in VR zu benutzen, werden im Verlauf der Arbeit Möglichkeiten zur Navigation und Interaktion vorgestellt, ausgewählt und implementiert. Alle zur Veranschaulichung präsentierten Bilder benutzten im Repository[45] verfügbare Meshes, Texturen und Datensätze.

1.1 ZIELSETZUNG

Die existierende Anwendung zur Darstellung von JuPedSim Simulationsdaten soll für die Benutzung in VR angepasst werden, um Anwendern einen möglicherweise besseren Überblick über die Ergebnisse zu bieten als es eine traditionelle Anwendung es kann.

Hierfür wird im Rahmen dieser Arbeit die Anwendung um VR Funktionalität erweitert indem passende Systeme zur Bewegung und Interaktion mit den Daten ausgewählt und implementiert werden. Diese können im Nachhinein durch das JuPedSim Team evaluiert und erweitert werden.

1.2 AUFBAU DER ARBEIT

Zunächst werden die für das Projekt relevanten und für das Verständnis der Arbeit notwendigen Grundlagen erläutert. Hierzu gehört ein Überblick über VR und die Unity Engine. Darauf folgt eine Analyse von populären Bewegungsmethoden und User Interface Varianten, welche einen Pool an möglicher Funktionalität für die Anwendung bildet. Darauf folgt eine Definierung des finalen Konzepts für die Anwendung mit allen notwendigen Funktionen. Die Realisierung dieses Konzeptes wird in Kapitel 5 erläutert. Hier werden Änderungen an der ursprünglichen Anwendung sowie die Implementierung der VR Features beschrieben. Im letzten Kapitel wird das Ergebnis der Arbeit zusammengefasst und ein Ausblick auf mögliche Folgeprojekte gegeben.

1.3 VERWANDTE ARBEITEN

Das Programm, welches für diese Arbeit als Grundlage dient, wurde von Daniel Büchele für dessen Masterarbeit[4][3] entwickelt und von Fabian Plum und Mohcine Chraibi für die Benutzung mit JuPedSim Simulationsdaten angepasst[22]. Das Unternehmen accu:rate[47] hat dem Projekt von Büchele VR Funktionalität für Google Cardboard und Gear VR hinzugefügt[23], um den User in der Welt laufen oder durch die Augen eines Fußgängers zu schauen zu lassen. Die genauen Ausmaße der VR Funktionalität sind leider nicht bekannt und es gibt keinen öffentlichen Zugriff auf diese Version des Projekts.

2 EINFÜHRUNG

2.1 VIRTUAL REALITY

Der Begriff Virtual Reality (VR) bezeichnet eine "durch spezielle Hard- und Software erzeugte künstliche Wirklichkeit"[10]. Das Ziel von VR liegt darin, dem Nutzer ausschließlich die virtuelle Welt zu präsentieren, um eine möglichst hohe Immersion darin zu erreichen[9]. Aktuelle VR-Systeme bestehen meist aus einem am Kopf befestigte Bildschirm, sogenannten Head-Mounted-Displays (HMD), zur Übermittlung von Bild und Ton, sowie einem Controller-Paar für Benutzereingaben. Alternativ werden in stationären Installationen auch ganze Projektionsräume verwendet. Die Position und Rotation des HMD sowie der Controller wird in Echtzeit ermittelt und an VR-Anwendungen weitergeleitet. Hierfür gibt es unterschiedliche Ansätze, wie die Nutzung von im Raum verteilten Basis-Stationen als Referenzpunkt für im HMD integrierte Sensoren[7] oder die Integrierung von Kameras in das HMD, welche sowohl Umgebung als auch die zugehörigen Controller verfolgen[16]. Letzterer wird auch als *inside-out tracking* bezeichnet und ermöglicht kabellose, mobile VR-Headsets[17]. Durch Positions- und Rotationsbestimmung ergeben sich sechs Freiheitsgrade (Drei Translationsachsen und drei Rotationsachsen). Dies wird auch als 6 Degrees of Freedom (6DOF)[8] bezeichnet und ist für die Vermeidung von Simulatorkrankheit wichtig, da jede Kopfbewegung an die Anwendung weitergegeben werden kann (siehe Sektion 2.1.2). Neben virtual Reality gibt es auch Augmented Reality (AR) und Mixed Reality (MR), welche in unterschiedlichen Ausmaßen die reale Umgebung in die Erfahrung miteinbeziehen [9].

2.1.1 *Immersion*

Für VR beschreibt Immersion "den Zustand, in dem der Nutzer das Bewusstsein, sich in einer künstlichen Welt zu befinden, verliert"[2]. Dieses Verständnis, sich tatsächlich in der virtuellen Welt zu befinden, wird durch eine große Bandbreite an Faktoren gestärkt. Nach Jason Jerald[11] wird die erfahrene Immersion unter anderem durch folgende Punkte beeinflusst:

- Das Ansprechen von möglichst vielen Sinnen (häufig Sehen, Hören und begrenztes Tasten mittels Eingabegeräten)
- Die Qualität der präsentierten Stimuli (Auflösung des Bildes)
- Die Übereinstimmung zwischen physischen und simulierten Eindrücken (z.B. Unterschiede zwischen realer und simulierter Kopfbewegung)

- Das Gefühl, von der Simulation völlig umgeben zu sein (Großes Blickfeld)
- Die Bandbreite und Glaubhaftigkeit der möglichen Interaktionen. Kann der Benutzer sich frei bewegen und Gegenstände greifen? Verhalten sich diese Gegenstände wie erwartet?

2.1.2 *Motion Sickness*

Mit Bewegungskrankheit (engl. Motion Sickness) wird ein Krankheitszustand beschrieben, der unter anderem durch Stimulation des Gleichgewichtssinnes[18] oder widersprüchliche Informationen mehrerer Sinnesorgane ausgelöst werden kann[14]. Typische Symptome beinhalten Unwohlsein, Blässe, Schwitzen, Übelkeit und Erbrechen[18]. Im Umfeld der virtuellen Realität spricht man auch von Simulatorkrankheit oder VR-Krankheit. Es gibt viele Auslöser, jedoch kommt in VR vor allem der visuell-vestibuläre Konflikt zum Tragen. Hierbei stimmen die visuellen Informationen nicht mit den Informationen des Gleichgewichtssinnes überein[14]. Dies kann der Fall sein, wenn der Benutzer eine Kopfbewegung vollführt, die nicht über visuelle Informationen bestätigt wird. Anders herum kann auch visuell eine Bewegung angezeigt werden, die ein still stehender Benutzer nicht über seinen Gleichgewichtssinn wahrnimmt.

Für VR Anwendungen ergeben sich gewisse Regeln, die es einzuhalten gilt um Bewegungskrankheit zu vermeiden. Oculus gibt folgende Mechaniken als problematisch an[43]:

- Änderung der Kopforientierung ohne Benutzereingaben
- Veränderung des Sichtfeldes
- Plötzliche oder vermeintlich grundlose Verschiebung der Kamera
- Ignorieren oder Überschreiben von Kopfbewegungen des Benutzers

Zusätzlich dazu gibt es technische Aspekte, die es zu beachten gilt. Hohe Latenz zwischen Eingaben und visuellem Feedback sowie niedrige und instabile Bildfrequenzen (FPS) können für stärkere Symptome sorgen[43][44].

2.2 GAME ENGINES UND UNITY

Die Umsetzung des Projekts erfolgt mit der Unity Game Engine[37]. Eine Game Engine ist eine Ansammlung von Komponenten, welche es Entwicklern erleichtert, komplexe 2D oder 3D Spiele zu entwickeln. Zu diesen Komponenten gehören üblicherweise ein Rendering System, Physikberechnung, Audio System und GUIs[24]. Diese Elemente werden von einem Programm kombiniert und über einen Editor dem Entwickler zugänglich gemacht. In der Regel besteht zusätzlich die Möglichkeit, die Funktionen der Engine

über eigenen Code zu erweitern. In Unity geschieht dies durch Skripte in der Programmiersprache C#. Eine solche Engine erlaub es dem Entwickler, in anderen Programmen erstellte Objekte wie Audiodateien, 2D- und 3D Meshes in einer Umgebung zu kombinieren, mit Licht- und Physikberechnung zu versehen und über Logikcode ein funktionsfähiges Spiel zu erstellen.

In Unity ist jede Umgebung eine Szene mit zugehöriger Szenenhierachie (Ähnlich der Ordnerhierachie eines Dateisystems). Diese Szenenhierachie wird mit Objekten (GameObjects) gefüllt, die in dem Raum der Szene eine Position besitzen. Jedes Objekt kann Komponenten wie Renderer, Lichtquellen, Kollisionsboxen oder eigene Skripte besitzen, welche das Verhalten und aussehen des Objekts bestimmen. Sonstige wichtige Konzepte in Unity zum Verständnis der Arbeit sind in der folgenden Auflistung erklärt.

GameObject
GameObjects sind die grundlegenden Container in Unity[28]. Sie lassen sich in der Szenenhierachie verschachteln, bieten aber sonst kaum eigene Funktionen, sondern sind auf andere Unity-Komponenten angewiesen. In Kombination mit Geometrien und Texturen können beispielsweise Körper angezeigt werden. GameObjects können als eine Art Blaupause gespeichert werden, aus welcher sich Kopien erstellen lassen. Diese Blaupausen nennt man Prefabs[31].

Transform
Ein Transform bestimmt die Position, Rotation und Skalierung eines GameObjects[34]. Jedes GameObject besitzt immer genau eine Transform Komponente. Durch Verschachtelung von GameObjects innerhalb einer Szenenhierachie ergibt sich so ein Transformationsbaum. Für die spätere Implementierung spielt die Unterscheidung zwischen lokaler und globaler Transformation eine Rolle. Die lokale Transformation ist das Transform selbst, die globale Transformation ist das Ergebnis des Transforms und aller im Transformationsbaum darüberliegenden Transforms kombiniert.

Script
Ein Script ist ein Codefragment, welches erlaubt, direkten Einfluss auf das Spielgeschehen auszuüben[32]. Sie werden zur Verwaltung und Manipulation der GameObjects, Eingabeverwaltung, Physikberechnung und viele weitere Funktionen verwendet. In Unity wird hierfür die Programmiersprache C# verwendet.

MonoBehavior
MonoBehavior ist die Basisklasse jedes Unity Skriptes [30]. In Unity können zwar Skripte benutzt werden, die nicht von MonoBehavior erben, jedoch sind es nur diese Skripte, die als Komponente GameObjects hinzugefügt werden können. MonoBehavior Skripte enthalten Methoden zur Regelung des Programmflusses wie *OnUpdate()*, welche für jedes Bild neu ausgeführt wird.

Canvas

Ein Canvas in Unity ist ein Behälter für Elemente der Graphical User Interface (GUI) wie Text, Knöpfe und Slider und stellt die Fläche dar, auf der die GUI gerendert wird[25]. Bei einem Canvas kann es sich entweder um einen *world space* oder *screen space* Canvas handeln. Das screen space Canvas wird direkt vor der Kamera platziert und folgt dieser. Das world space Canvas hingegen steht frei im Raum und kann wie andere GameObjects im Editor oder durch Skripte verschoben, rotiert oder skaliert werden.

3 MÖGLICHE FUNKTIONALITÄT

Bevor ein Konzept für die neue VR Anwendung entwickelt werden kann, muss in Erfahrung gebracht werden, welche Funktionen momentan populäre Spiele oder Programme besitzen. Anhand dieser Untersuchung können für den Anwendungsfall passende Features ausgewählt und implementiert werden.

3.1 ANFORDERUNGEN

Zunächst werden grundlegende Anforderungen an alle für diese Anwendung auszuwählenden Features festgelegt. Dies betrifft vor allem Bewegungsarten und User Interfaces.

Unterstützung verschiedenster Hardware
Anbieterexklusive Features wie einzigartige HMD-Funktionen (z.B. Hand-Tracking) oder Controller-Layouts sollten vermieden werden.

Mobilität
Die Anwendung soll nur mit Headset und Controllern auskommen. Für kabelgebundene Headsets ist auch ein PC nötig, aber Standalone HMDs dürfen nicht wegen zu hoher benötigter Rechenleistung ausgeschlossen werden.

Intuitive Bedienung
Der Benutzer soll mit der Bedienung schnell vertraut werden können, ohne komplett neue Bedienkonzepte lernen zu müssen.

Komfort
Besonders in Bezug auf die Bewegungsmethoden müssen Features vermieden werden, die bekannt dafür sind, Simulatorkrankheit auszulösen.

3.2 FEATURES IN POPULÄREN COMPUTERSPIELEN

Einen guten Überblick über in Frage kommende Features liefert ein Blick auf aktuelle Bestseller Listen. Die digitale Verkaufsplattform Steam hat für das Jahr 2019 eine solche Liste veröffentlicht[6]. Aus dieser Liste wurden die “Platinum” und “Gold” Sellers (24 Titel) analysiert und deren Fortbewegungsmethoden und GUI Ansätze in Tabelle 3.1 gesammelt. Hierbei wird die für VR essentielle 1:1 Translation von physischer Bewegung zu ingame Bewegung (6DOF) nicht explizit aufgezählt, da sie in jedem Titel vorhanden ist. Die Funktionalität der Spiele wurde öffentlich zugänglichem Material

wie Trailern und offiziellen Videos, Homepages, Verkaufsseiten sowie Artikeln der Entwickler entnommen. Viele der Spiele wurden wegen der hohen Kosten nicht selbst getestet. Daher handelt es sich um eine subjektive und wegen Updates schnell veraltende Auflistung. Dennoch ist erkennbar, dass bestimmte Features sehr häufig auftauchen. Auf diese wird im Folgenden näher eingegangen.

Titel	**Fortbewegung**	**Primäre GUI**
Pavlov VR	Continuous	Schwebende Displays, Palette
Hot Dogs, Horseshoes & Hand Grenades	Teleport, Dash, Continuous, Swinging Arms	Schwebende Displays, Palette, Objekte
Beat Saber	—	Schwebende Displays
The Elder Scrolls V: Skyrim VR	Teleport, Continuous	Schwebende Displays, Palette
Arizona Sunshine®	Teleport, Continuous	Schwebende Displays
Zero Caliber VR	Continuous	Schwebende Displays, Palette
Fallout 4 VR	Teleport, Continuous	Schwebende Displays, Armbanduhr, Palette
Blade and Sorcery	Continuous, Swinging Arms	Schwebende Displays
GORN	Continuous, Drag	Objekte
BONEWORKS	Continuous	Schwebende Displays, Objekte
SUPERHOT VR	—	Objekte
VR Kanojo	Teleport	Schwebende Displays
Onward	Continuous	Schwebende Displays
Pistol Whip	Rails	Schwebende Displays
Contractors	Continuous	Schwebende Displays
Half-Life: Alyx	Teleport, Dash, Continuous	Schwebende Displays, Objekte, Armbanduhr
Vacation Simulator	Teleport	Objekte
Budget Cuts	Teleport	Schwebende Displays
Virtual Desktop	—	Schwebende Displays
Rick and Morty: Virtual Rick-ality	Teleport	Schwebende Displays, Objekte
Sairento VR	Dash, Continuous	Schwebende Displays, Armbanduhr
Moss	—	Objekte
Creed: Rise to Glory™	Swinging Arms	Schwebende Displays
Job Simulator	—	Objekte

Abbildung 3.1: VR Topseller in der Steam Bestseller Liste von 2019 in den Kategorien “Platinum” und “Gold”[6].

3.3 MÖGLICHE FORTBEWEGUNGSARTEN

Die Traversierung einer virtuellen Umgebung ist für viele Anwendungen ein essentieller Bestandteil des Erlebnisses und wird auch für diese Anwendung in zumindest einer Form benötigt. In dieses Sektion wird eine Auswahl an populären Fortbewegungsarten vorgestellt und anhand anderer Arbeiten gegeneinander abgewogen, um für die Anwendung eine informierte Entscheidung zu treffen.

3.3.1 *Teleport*

Unter Teleport ist hier eine augenblickliche Änderung der Spielerposition zu verstehen. Hierbei definiert Benutzer seine gewünschte Zielposition durch Zeigen. Dieses Zeigen kann beispielsweise durch eine gerade Linie von Controller oder Kamera zur Zielposition stattfinden oder, wie in den meisten in Tabelle 3.1 gelisteten Titeln der Fall, durch einen vom Controller ausgehen-

den Bogenwurf. Teleportation ist, wie in der Tabelle zu sehen, eine der am häufigst verwendeten Fortbewegungsarten. Verglichen mit Kontinuierlicher Bewegung (3.3.2) verursacht Teleportation deutlich geringere Symptome von Simulatorkrankheit, ohne den Benutzer bei der Traversierung signifikant zu verlangsamen [5].

Auf die Frage, welche Methode zur Auswahl der Zielposition die Beste darstellt, konnte während der Recherche keine wissentschafftliche Antwort gefunden werden. Fast alle untersuchten Titel mit Teleportation verwenden eine Version des Bogenwurfs. Eine mögliche Erklärung ist, dass der Bogenwurf den positiven Effekt besitzt, die maximal mögliche Teleportationsdistanz natürlich und dem Benutzer verständlich zu limitieren. Gleichzeitig ist es möglich, sich an Positionen zu teleportieren, zu denen der Benutzer keine direkte Sichtlinie besitzt.

Eine leichte Abwandlung von der Teleportation ist das in Tabelle 3.1 als *Dash* bezeichnete Verfahren. Hierbei wird der Benutzer nicht unmittelbar an die neue Position versetzt, sondern in einer schnellen Bewegung, die meist einen Bruchteil einer Sekunde andauert. Das Ziel der Dash Teleportation ist es, eine Desorientierung des Benutzers zu vermeiden, indem dieser die Bewegung visuell mitverfolgen kann [1].

3.3.2 *Kontinuierliche Bewegung*

Die kontinuierliche Bewegung (manchmal auch als “Full Locomotion” oder “Joystick Locomotion” bezeichnet) zeichnet sich, wie der Name vermuten lässt, dadurch aus, dass der Benutzer den Raum in einer kontinuierlichen Bewegung traversieren kann. Der Benutzer wird im virtuellen Raum anhand von Joystick oder Tasteneingaben in einer kontinuierlichen Bewegung verschoben, ohne dass er sich in der Realität bewegt. Benutzt wird dieses Verfahren häufig in Simulationen, in denen eine Teleportation wegen fehlender Immersion nicht wünschenswert ist. Oft wird es auch als Alternative zu anderen Bewegunsmethoden angeboten.

Ein großes Problem der kontinuierlichen Bewegung ist die Verursachung von Simulatorkrankheit. Durch die fehlende Übereinstimmung zwischen Gleichgewichtssinn und visueller Wahrnehmung ist sie in diesem Bereich deutlich schlechter als Teleportation [5].

3.3.3 *Swinging Arms und Walk-in-Place*

Zwei involviertere Bewegungsmethoden sind *Swinging Arms* und *Walk-in-Place*. Bei beiden geht es darum, den Benutzer aktiv eine Laufbewegung ausführen zu lassen, um Übelkeit zu vermeiden. Swinging Arms fokussiert sich auf die Bewegung der Arme, die beim Laufen vor und zurück schwingen. Walk-in-Place hingegen lässt den Benutzer auf der Stelle laufen und übersetzt dies in eine Bewegung in der Anwendung. Es gibt mehrere Studien, die diese Methoden miteinander vergleichen. Eine Studie kam zu dem Ergebnis, dass Walk-in-Place eine bessere räumliche Wahrnehmung ermöglicht und

weniger Desorientierung verursacht als Swinging Arms [46]. Dies wird von der Studie von Pai und Kunze[21] ebenfalls unterstützt. Allerdings sorgt in dieser Studie Walk-in-Place für stärkere Symptome von Simulatorkrankheit und es wurde eine deutlich höhere Anstrengung gegenüber Swinging Arms festgestellt.

3.3.4 *Andere*

Jegliche Fortbewegungsarten, die sich auf zusätzliche Hardware (wie z.B. 2D Tretmühlen) stützen, sind für diese Anwendung grundsätzlich ungeeignet, da das Ziel der Mobilität nicht gewährleistet wird. Daher werden diese hier nicht weiter beachtet. In Tabelle 3.1 wurden jeweils ein Mal *Drag* und *Rails* aufgelistet. Bei Drag bewegt man sich, indem man die Umgebung oder Luft greift und seine Hand bewegt um sich durch den Raum zu ziehen. Dies ist eine sehr langsame Art der Fortbewegung und daher hierfür ungeeignet. Bei Rails bewegt man sich auf einer festen Strecke, über die man kaum bis keine Kontrolle hat. Aufgrund der variierenden Geometrie kann auch diese Art der Fortbewegung hier nicht eingesetzt werden.

3.4 GRAFISCHE BENUTZEROBERFLÄCHE

Die Möglichkeiten für Grafische Benutzeroberflächen in VR entwickeln sich mit jeder Generation von VR-Hardware weiter und werden noch aktiv erforscht. Leider konnten keine nennenswerten Studien, welche verschiedene GUI Ansätze vergleichen, lokalisiert werden. Daher sind im folgenden Abschnitt solche Ansätze aufgelistet, die in populären Anwendung häufig Verwendung finden (Tabelle 3.1).

3.4.1 *Objekte*

Ingame Objekte wie dreidimensionale Hebel und Knöpfe sind für viele VR Anwendungen ein wichtiger Bestandteil der Erfahrung. Spiele wie *Job Simulator* verlassen sich sogar komplett auf die Interaktion mit solchen Ingame Objekten. Damit wird der Benutzer nicht durch traditionellere Menüs wie schwebenden Displays abgelenkt und idealerweise die Immersion erhöht.

3.4.2 *Schwebende Displays*

Die häufigste Form von GUIs in VR bilden die schwebenden Menüs. Hierbei erscheint meist auf Tastendruck eine stationäre zweidimensionale Fläche vor dem Benutzer, welche typische UI Elemente wie Knöpfe, Scrollbars, Checkboxen und Slider enthält. Diese werden dann durch einen Zeiger/Auswahlstrahl, welcher vom Controller ausgeht, bedient. Diese Menüs werden meist in kurzer bis mittlerer Distanz (geschätzt 1-5m) vom Benutzer entfernt eingesetzt und für Hauptmenüs und Einstellungsmenüs verwendet.

3.4.3 *Palette und Armbanduhr*

Verwandt mit den schwebenden Displays sind das Paletten- und Armbanduhrmenü. Bei beiden wird ein zweidimensionales Display oder andere interaktive Elemente relativ zur Controller-Position und Rotation platziert, sodass es immer schnell erreichbar ist. Beim Armbanduhrmenü ist diese Position der Handrücken. Zur Bedienung schaut der Benutzer "auf seine Armbanduhr". In manchen Anwendungen wie *Half-Life: Alyx* ist dieses Menü immer vorhanden und wird tatsächlich als eine Art Armbanduhr dargestellt, in anderen erscheint das Menü nur, wenn der Benutzer direkt darauf schaut. Das Palettenmenü Funktioniert ähnlich, befindet sich jedoch an der Handfläche oder leicht darüber und wird in keinem der Untersuchten Titel permanent angezeigt. Beide Menü Varianten werden in den untersuchten Titeln für häufig auftretende Interaktionen verwendet.

4 KONZEPT

Aus der Auflistung von möglichen Interaktions- und Bewegungsverfahren in Abschnitt 3 wurden geeignete Features gewählt und in einem kompletten Konzept für die Anwendung zusammengeführt. Zur Veranschaulichung werden hier Screenshots der fertigen Anwendung verwendet.

4.1 ÜBERNAHME VON FEATURES

Die Basisanwendung besitzt mehrere Features, die übernommen werden sollen. Hierzu gehören:

Anzeigen der Fußgänger
: Die Anwendung kann Fußgänger anzeigen und animieren. Diese Funktion ist für die geplante Anwendung essentiell. Hierzu gehört auch die Möglichkeit, den Abspielvorgang per Knopfdruck zu pausieren und wieder aufzunehmen, sowie ein Slider zur genaueren Kontrolle der aktuellen Zeit. Es ist schon ein Fußgänger-Asset mit Animationen vorhanden, welches hierfür benutzt wird. Der Detailgrad der Fußgänger ist nicht fotorealistisch, allerdings für Demonstrationszwecke ausreichend.

Anzeigen der Geometrie
: Die Geometrie aus den Simulationsdaten wird in halbdurchsichtiger Form erstellt und angezeigt und ist genauso wichtig wie die Fußgänger selbst. Bis auf eventuelle Texturanpassungen soll dieses Feature so übernommen werden.

Anzeigen von Dichte/Geschwindigkeit
: Der Benutzer kann die Fußgänger farblich hervorheben lassen um sich entweder die Fußgängerdichte anzuzeigen (alle Fußgänger, die in einer konfigurierbaren Dichte stehen, werden rot hervorgehoben) oder deren Geschwindigkeit zu verdeutlichen (farbliche Markierung von rot bis grün)

Anzeigen von Laufpfaden
: Die Laufpfade der Fußgänger können durch eine Linie pro Person dargestellt werden.

Vorerst nicht übernommen wird die Funktion, ein Diagramm anzuzeigen, da diese Funktion nach Konvertierung des Projekts auf eine modernere Unity Version nicht mehr funktionsfähig war, sie für die VR Funktionalität nicht essentiell ist und Beheben des Fehlers potentiell zu viel Zeit in Anspruch

nehmen würde. Ebenso wird die Möglichkeit nicht übernommen, alle Fußgänger zu zählen, die eine durch den Benutzer festgelegte Linie überschreiten. In eigenen Tests hat diese Funktion in der ursprünglichen Anwendung auch Fußgänger auf anderen Stockwerken gezählt und müsste korrigiert werden, was zunächst nicht im Umfang der Arbeit liegt. Die Anwendung erlaubt es dem Benutzer, die Kamera per WASD-Tasten und Maus zu navigieren. Diese Art der Steuerung kann so nicht in VR übernommen werden.

4.2 BETRACHTUNGSMODI

Die Anwendung soll dem Benutzer die Möglichkeit geben, die Simulation in Echtgröße zu sehen, um eine gute räumliche Vorstellung der Geometrie zu bekommen oder wenn gewünscht als Miniatur-Variante zu betrachten, um sich einen schnellen Überblick zu verschaffen. Hierfür sollen zwei unterschiedliche Betrachtungsmodi eingesetzt werden, zwischen denen der Benutzer frei wechseln kann. Beide Modi sollen auf einer endlosen neutralen Fläche mit Gitter-Textur stattfinden um die Tiefenwahrnehmung zu unterstützen ohne Referenzobjekte benutzen zu müssen.

4.2.1 *Modus: Echtgröße*

Hier soll der Benutzer die Simulation in ihrer tatsächlichen Größe betrachten, sich umher bewegen und die Situation der Fußgänger aus nächster Nähe einschätzen können. Räumliche Wahrnehmung ist der größte Vorteil von Virtual Reality und sollte daher auch voll genutzt werden. Möglicherweise lässt dieser Modus eine bessere Beurteilung der Simulation zu, als die Betrachtung auf einem traditionellen Monitor. Besonders die Dichte der Menschenmenge könnte sich hiermit deutlich besser wahrnehmen lassen und diese Anwendung zu einem geeigneten Tool zur Präsentation von Daten gegenüber Außenstehenden machen. In Abbildung 4.1 steht der Benutzer innerhalb der Simulation, während sie abgespielt wird. Die Texturen der Geometrie in diesem Modus sollen vorerst konsistent mit der Miniatur und dementsprechend halbtransparent sein. Dies sorgt für eine gute Übersicht vor allem in mehrstöckigen oder engen Geometrien, könnte aber bei Personen mit Höhenangst diese auslösen. In späteren Versionen könnte eine Texturauswahl implementiert werden, um Betroffenen eine Alternative zu bieten.

4.2.2 *Modus: Miniatur*

Die Miniaturansicht der Simulation soll vom Benutzer rotierbar, skalierbar und verschiebbar sein, damit er die Simulation aus beliebigen Positionen betrachten kann, ohne seine eigene Position zu verändern. Damit bleibt die Anwendung auch in kleinen Räumen oder selbst sitzend benutzbar. Zur Präsentation auf Torsohöhe soll ein höhenverstellbares Podest zum Einsatz kommen, auf dem die Simulation platziert ist. Das Podest soll deaktivierbar sein,

Abbildung 4.1: Benutzer steht in der Simulation, während sie abgespielt wird.

um die Simulation auf dem Boden zu betrachten, was gerade bei flachen Geometrien eine gute Übersicht bieten könnte. Abbildung 4.2 zeigt eine Simulation im Miniaturmodus auf dem Podest. Rotation, Skalierung und Verschiebung soll über Controllereingaben stattfinden. Hierfür eignet sich eine Multitouch-ähnliche Umsetzung, bei der der Benutzer durch Betätigen einer der Greif-Tasten verschiebt oder rotiert (z.B. Verschiebung mit linkem Controller, Rotation mit rechtem Controller) und bei Betätigung beider Tasten eine Skalierung durchführt.

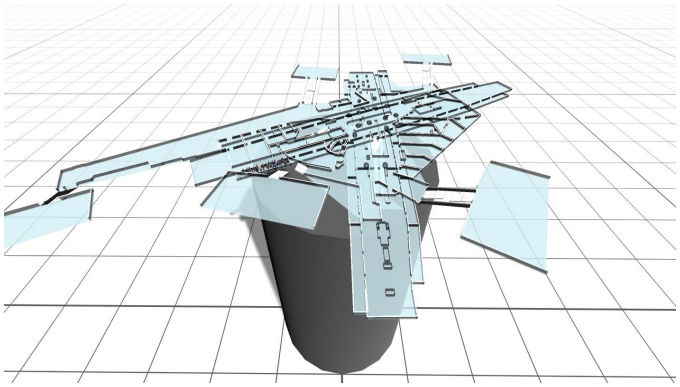

Abbildung 4.2: Benutzer betrachtet die Simulation im Miniaturmodus. Hier einen Datensatz der U-Bahn Haltestelle *Osloer Straße* in Berlin.

4.3 GUI

Für alle Menüs sollen Flächen verwendet werden, die der Benutzer durch Zeigen mit den Controllern bedienen kann. Wenn der Benutzer auf ein Menü zeigt, muss also ein Strahl die "Mausposition" anzeigen. Zum einen sind Benutzer durch Computermäuse schon mit dieser Art der Bedienung vertraut, zum anderen können die Menüs dadurch ohne große Anstrengung benutzt werden. Gesteigerte Immersion durch Elemente, die der Benutzer mit seinen virtuellen Händen berühren kann, ist für diese Anwendung nicht nötig und würde nur die Komplexität erhöhen.

4.3.1 *Hauptmenü*

Für Hauptmenüs werden, wie in Sektion 3.4.2 beschrieben, meist stationäre, schwebende Menüs eingesetzt. In dieser Anwendung soll ein solches schwebendes Menü verwendet werden, welches beim Drücken der Menütaste mehrere Meter vor dem Benutzer auf Augenhöhe erscheint. Es sollte groß genug sein, um alle Elemente trotz geringer Displayauflösung klar erkennen zu können und dennoch klein genug, um komfortabel in das Sichtfeld des Benutzers zu passen. Alle Untermenüs sollen in ähnlicher Größe knapp vor dem Obermenü erscheinen, womit potentiell unangenehme Übergangsanimationen vermieden werden und der Benutzer seinen Kopf während der Bedienung nicht bewegen muss. Die in Sektion 4.3.2 genannten Armmenüs sollen über ein Optionsmenü konfigurierbar sein. Das umgesetzte Hauptmenü ist in Abbildung 4.3 zu sehen. Abbildung 4.4 zeigt das Untermenü für die Armmenü-Einstellungen. Erkennbar hier ist, dass ein anderes Menü dahinter liegt (Einstellungs-Obermenü), welches wieder zum Vorschein kommt, wenn sich das aktuelle Menü schließt. Eine genaue Erklärung der Armmenü-Einstellungen findet sich im folgenden Abschnitt.

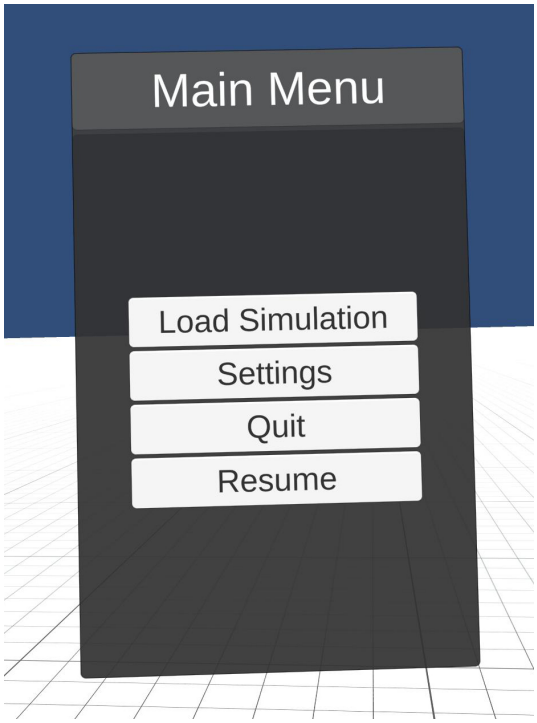

Abbildung 4.3: Hauptmenü der Anwendung.

4.3.2 *Armmenüs*

Die Anwendung erfordert zur Kontrolle des Abspielvorgangs mehrere schnell zu erreichende GUI Elemente wie Play/Pause Knopf, Zeitlinie, Checkboxen für Anzeige von Dichte und Geschwindigkeit der Fußgänger. Gleiches gilt

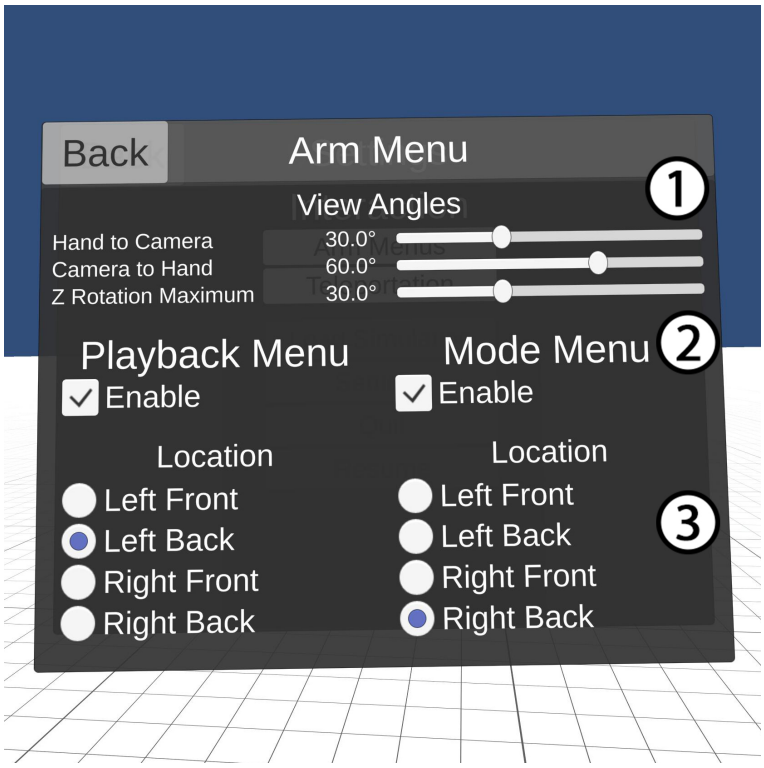

Abbildung 4.4: Einstellungen für die Armmenüs.

für Manipulation der Modi, für welche Knöpfe zum Zurücksetzen der Betrachterposition oder Knöpfe zur Konfiguration des Miniaturpodestes benötigt werden. Ein schwebendes Menü wie für das Hauptmenü ist für den häufigen Gebrauch zu langsam, daher fiel die Entscheidung auf die Verwendung von Arm- bzw. Handmenüs wie in 3.4.3 beschrieben. Es soll für jede Hand ein Menü verwendet werden. Ein Menü enthält Kontrollelemente für die Simulation selbst, wie Abspielen der Simulation und Anzeigen der Fußgängerdichte (Abb. 4.5). Das andere Menü enthält Elemente für die Manipulation der Betrachterposition und des aktuellen Betrachtungsmodus (Abb. 4.6). Für manche Individuen sind gewisse Armpositionen unkomfortabel, daher soll es möglich sein, die Menüs entweder am Handrücken wie eine Armbanduhr (Abb. 4.5) oder am Handteller wie eine Farbpalette (Abb. 4.6) anzuzeigen. Jedes der Menüs kann dann durch Zeigen mit der jeweils anderen Hand bedient werden.

Damit die Menüs den Benutzer nicht ablenken, indem sie permanent sichtbar sind, sollen sie nur angezeigt werden, wenn der Benutzer direkt zu ihnen schaut und die Menüs auch zum Benutzer gedreht sind. Abbildung 4.4 zeigt die verfügbaren Einstellungen für die Menüs. Die Slider (1) erlauben es, die Winkel, in denen die Armmenüs für den Benutzer sichtbar sind, beliebig anzupassen (Genaueres dazu in Sektion 5.5.2). Die Checkboxen (2) erlauben es, die Menüs zu aktivieren/deaktivieren, für den Fall dass eines der beiden nicht gebraucht wird. Die Gruppen von jeweils vier Radio-Knöpfen (3) erlauben es, die Position und Seite der Menüs zu konfigurieren.

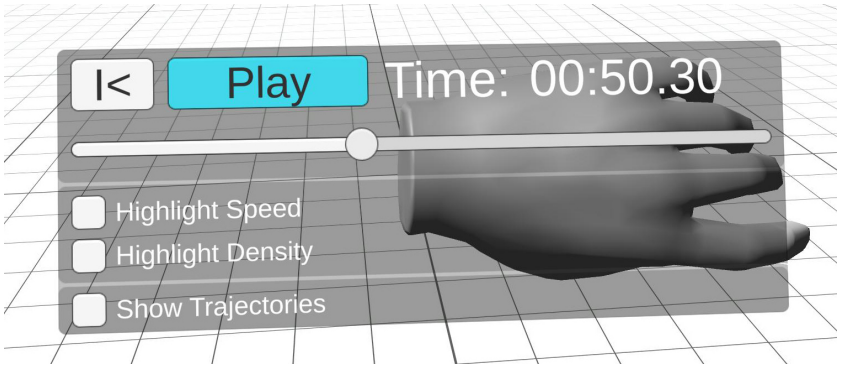

Abbildung 4.5: Armmenü mit Elementen zum Abspielen der Simulation positioniert wie eine Armbanduhr.

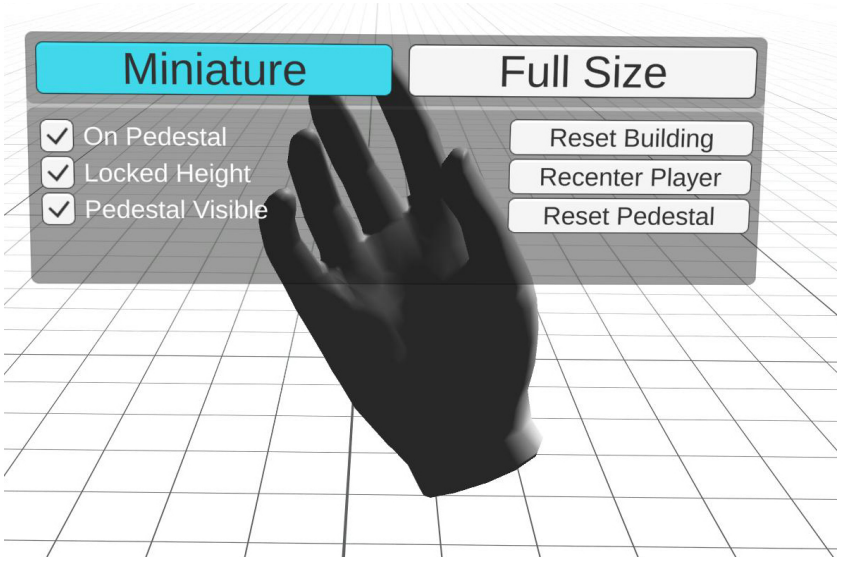

Abbildung 4.6: Armmenü mit Elementen zur Modusverwaltung positioniert an der Handvorderseite.

4.4 BEWEGUNG VIA TELEPORTATION

Als Fortbewegungsmethode wurde Teleportation gewählt, da sie unkompliziert ist und dennoch alle Ziele abdeckt. Für diese Anwendung spielt Immersion keine so große Rolle wie in Computerspielen, daher sind komplexe Methoden wie "Walk-in-Place" (Sektion 3.3.3) nicht erforderlich. Kontinuierliche Bewegung (Sektion 3.3.2) kann aufgrund der erzwungenen Kamerabewegung schnell für Simulatorkrankheit sorgen (Sektion 2.1.2), daher wird auch diese nicht verwendet. Teleportation (Sektion 3.3.1) hingegen ist in der Regel angenehm für den Benutzer, wenn auch gelegentlich desorientierend für VR-Neulinge. Das Ziel der Teleportation soll durch einen Bogenwurf bestimmt werden. Damit ist die Reichweite limitiert und es wird keine direkte Sichtlinie zum Ziel benötigt. Die normale Teleportation soll hauptsächlich im Echtgrößenmodus zum Einsatz kommen, jedoch spricht nichts dagegen, sie auch im Miniaturmodus zuzulassen. Im Miniaturmodus soll es noch eine

spezielle Version der Teleportation geben. Der Benutzer kann mit dem Bogenwurf in die Simulation zeigen, was einen direkten Wechsel in den Echtgrößenmodus und Teleportation an die gewählte Stelle auslösen soll. Damit kann der Benutzer die Miniatur zur Übersicht nutzen und dann schnell zu jeder beliebigen Position in der Simulation springen.
Damit die Anwendung benutzt werden kann, ohne sich ständig umdrehen zu müssen, soll der Benutzer durch Bedienung der Controller nach links oder rechts rotieren können. Anstelle von kontinuierlicher Rotation findet dies schrittweise statt, um Simulatorkrankheit zu vermeiden[19]. Beispielsweise rotiert sich der Benutzer mit einer Eingabe von "links" immer um 45° nach links.

4.5 REPRÄSENTATION DER CONTROLLER

Die Anwendung soll mit verschiedensten VR-Headsets kompatibel sein, daher sollen generische Hände anstelle von spezifischen Controller Meshes verwendet werden. Wenn möglich sollten animierte Hände umgesetzt werden, welche Controller-Eingaben intuitiv visuell darstellen. Sowohl spezifische Meshes für jede Controllerart als auch animierte Hände sind ein großer Aufwand, daher ist zunächst das Ziel, statische Hände als Mesh zu verwenden. Als Meshes werden hierfür Hände aus dem von Oculus bereitgestellten *Unity Sample Framework*[20] benutzt.

4.6 BUTTON-LAYOUT

Das Kontrollschema der Anwendung soll möglichst intuitiv gestaltet werden. Die Interaktion mit der Miniatur soll einem Greifen ähneln und benutzt daher an den Controllern hierfür vorgesehene Hand Trigger. Bedienung der GUI erfolgt über einen Zeiger, weshalb passenderweise auch der Zeigefinger zum Aktivieren von GUI-Elementen benutzt werden soll. Jegliche Bewegung im Raum soll gruppiert über das Joystick passieren. Bei Berühren des Joysticks wird die Teleportationslinie angezeigt und durch Drücken erfolgt die Teleportation zur gewählten Position. Auch die Rotation des Benutzers soll über den Joystick stattfinden. Eingaben auf der X-Achse (nach links und rechts) verursachen eine Drehung in die jeweilige Richtung. Es ist anzumerken, dass durch die Verwendung von Unity XR keine konkreten Layouts für Controller bestimmter Hersteller, sondern generische Eingabefunktionen benutzt werden. Im Folgenden findet sich eine übersichtliche Auflistung des Kontrollschemas.

Zeigefinger Trigger Links/Rechts
Bedienung der GUI. Kurzes Drücken für Knöpfe, Drücken und Halten für Slider

Hand Trigger Links/Rechts
Greifen der Geometrie im Miniaturmodus. Standardmäßig links Greifen für Translation, rechts für Rotation und beide Seiten für Skalierung.

Primärer Knopf Links/Rechts
Anzeigen und Verstecken des Hauptmenüs

Joystick Links/Rechts
Anzeigen der Teleportationslinie bei Berührung und Teleport durch Drücken. Rotation des Benutzers bei Stick-Bewegung nach links oder rechts.

5 REALISIERUNG

In diesem Abschnitt wird die Implementierung des in Kapitel 4 vorgestellten Konzeptes erklärt. Das fertige Projekt inklusive aller Assets und Code ist auf der beiliegenden CD und im Github Repository von Marvin Weisbrod zu finden [45].

Verwendete Hardware

Während der Entwicklung wurde das Projekt mit einer *Oculus Rift CV1* und zwei *Oculus Touch* Controllern getestet. Als Computer kam ein *Windows 10* PC mit *NVidia GeForce GTX 1660 SUPER* Grafikkarte und *Intel i5 3570K* Prozessor zum Einsatz.

5.1 VR FUNKTIONALITÄT

Unity unterstützt schon seit mehreren Jahren die populärsten HMDs wie die Oculus Rift und HTC Vive. Hierbei kamen bisher Headsetspezifische Packages wie die Oculus Integration zum Einsatz, mit denen man nur bestimmte Hardware betreiben konnte, sofern man nicht diese Packages abstrahiert und Plugin Loader benutzt, um nur die HMD Unterstützung zu laden, die aktuell benötigt wurde. Dies bedeutete einen großen Aufwand, wenn mehrere Headsets unterstützt werden sollten.

Mit Unity 2019.3 wurde Unity XR[36] eingeführt, um dieses Problem zu lösen. Unity XR bietet ein Framework zur Entwicklung von Virtual Reality, Mixed Reality und Augmented Reality Anwendungen anhand abstrahierter Strukturen, die in Projekten genutzt werden können, ohne direkt mit Headsetspezifischen Plugins kommunizieren zu müssen. Die Verwaltung der einzelnen Plugins übernimmt Unity XR selbst, mit nur geringer Konfiguration von Seiten des Entwicklers.

Indem für dieses Projekt Unity XR zum Einsatz kommt, lassen sich alle gängigen VR Headsets unterstützen und einige wertvolle Funktionen des Frameworks nutzen, die nicht mehr neu implementiert werden müssen. Darunter fällt ein rudimentäres System zur Teleportation und GUI Interaktion.

5.1.1 *Grundlegende VR Funktionalität*

Für die Benutzung von Unity XR müssen in Unity Version 2019.3 mehrere Packages installiert werden:

- **XR Management**[41]
- **XR Interaction Toolkit**[39]
- **XR Interaction Subsystems**[40]
- **XR Plugin** für jedes zu unterstützende Headset[38]

Das Verwalten und Laden der Plugins für die Headsets übernimmt von da an das XR Plugin Management System.

Für den Spielercharakter bietet Unity XR das XRRig Prefab, welches Kameras, Tracking Skripte und Controller enthält. Dieses dient als Grundlage für den Spieler in diesem Projekt und wird in den folgenden Sektionen um Funktionen erweitert. Abbildung 5.1 bietet eine Übersicht der GameObject Hierachie des XRRig Prefabs, auf die in den folgenden Kapiteln noch weiter eingegangen wird.

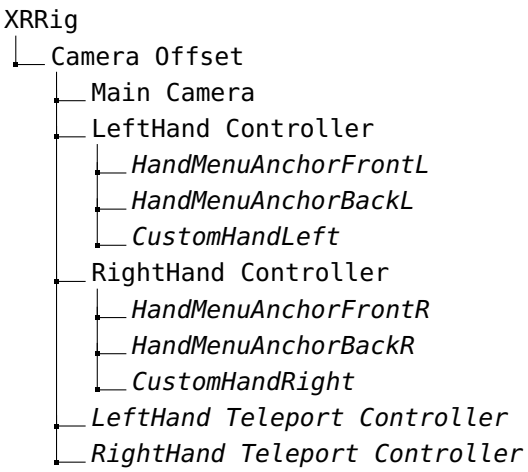

Abbildung 5.1: Hierachiebaum des XR Rigs. Kursive angegebene GameObjects wurden für dieses Projekt hinzugefügt.

5.1.2 *Input Verwaltung*

Unity XR verwaltet eine Liste an Eingabegeräten, auf die mit verschiedenen Methoden zugegriffen werden kann. Zur Verwaltung der Controller für beide Hände eignet sich die Funktion `InputDevices.GetDevicesAtXRNode` [29], die alle InputDevices zurückgibt, die von den XR Plugins als einem bestimmten XRNode (Entspricht einer Position am Körper), beispielsweise der linken Hand, zugehörig markiert wurden. Damit können auch gemischte Hardwarekonfigurationen unterstützt werden. Um zu überprüfen, ob ein

Button betätigt wurde, muss nun der Button-Status von jedem InputDevice am gleichen XRNode abgefragt werden[35].

In dieser Sektion wird auf die verwendete Abstraktion eingegangen, die es ermöglicht aus anderen Skripten einfache Input Abfragen zu tätigen. Im Folgenden sind die für diese Abstraktion erstellten Structs und Klassen beschrieben.

STRUCTS

ManagedButton
: Beinhaltet den Status eines Buttons im aktuellen und vorherigen frame und stellt getter und setter zur Verfügung.

ManagedAxis1D
: Beinhaltet den aktuellen Wert einer Achse sowie getter und setter.

ManagedAxis2D
: Beinhaltet den aktuellen Wert zweier Achsen, die gemeinsam den Zustand von beispielsweise Steuerknüppeln angeben, sowie getter und setter.

KLASSEN

HandDeviceManager
: Aktualisiert regelmäßig die Liste von InputDevices für die beiden Controller und implementiert structs, um für Buttons und Achsen generische Methoden wie “isPressed()” und “isReleased()” zur Verfügung zu stellen. Beinhaltet auch Methoden zur Aktualisierung der structs.

ManagedHand
: Beinhaltet für jeden Input des Controllers eine Instanz der Structs, die vom HandDeviceManager implementiert wurden sowie eine Update Methode zur Aktualisierung dieser in jedem Frame.

HandManager
: Beinhaltet jeweils eine ManagedHand Instanz für den linken und rechten Controller.

Durch diese Abstraktion benötigt jedes Skript, das den Status eines Inputs abfragen muss, nur eine Referenz zur HandManager Instanz und kann damit durch aufrufe wie

```
if (handManager.Left.buttonPrimary.isPressed()) {...}
```

auf die Eingabe reagieren.

5.2 ANPASSUNG DER URSPRÜNGLICHEN ANWENDUNG

Das Basisprojekt wurde für diese Arbeit von Unity Version 5.5.0f3 zu Version 2019.3 konvertiert, um die aktuellsten Unity Packages und Features benutzen zu können. Version 2019.3 (Veröffentlicht Ende Januar 2020[42]) war zu Beginn der Realisierung die aktuellste veröffentlichte Unity Version.

5.2.1 *Entfernte Funktionalität*

Zusätzlich zu den in Sektion 4.1 erwähnten Komponenten, die nicht in dieses Projekt mit übernommen werden, wurde die existierende GUI vollständig entfernt. Die 2D GUI wurde mit dem IMGUI System von Unity umgesetzt. Dieses GUI System funktioniert in VR nicht und verursacht in diesem Fall sogar Performance-Probleme. Daher wurde sie entfernt und sollte in Zukunft mit den moderneren Canvas UIs ersetzt werden, wenn noch eine 2D GUI benötigt wird.

5.2.2 *GameObject Pooling*

In dem Basisprojekt wird, nachdem die Fußgängerdaten aus der XML Datei geladen wurden, für jeden Fußgänger ein GameObject mit SkinnedMeshRenderer[1] Komponente und Verwaltungsskript erzeugt. Dieses Skript erhält die Fußgängerdaten und aktualisiert das GameObject Transform jedes Frame über die eigene Update-Funktion.

Die Instanziierung dieser GameObjects sorgte während der Entwicklung aufgrund der vielen Komponenten bei Hunderten von Fußgängern für Verzögerungen von mehreren hundert Millisekunden jedes mal, wenn ein neues Fußgänger-Datenset geladen wurde. Gerade für VR Anwendungen haben signifikante Einbrüche der Bilder pro Sekunde (FPS) oder Einfrieren des Bildes einen direkten Einfluss auf das Befinden des Benutzers. Um dieses Problem möglichst zu vermeiden oder zumindest auf einen besseren Zeitpunkt zu verschieben wurde ein simples Pooling-System umgesetzt.

Hierbei werden die eingelesenen Fußgängerdaten von den Fußgänger GameObjects getrennt und in zwei Listen von einem `PedestrianSystem` Objekt verwaltet. Das `PedestrianSystem` besitzt eine Liste von unbenutzten GameObjects und weist neuen Fußgängerdaten eines zu oder erstellt bei Bedarf ein neues GameObject. Pool-Objekte welche in Benutzung sind, werden aus dem Pool entfernt. In der Update-Methode des `PedestrianSystem` werden dann die notwendigen Berechnungen für Position, Rotation und Animationsgeschwindigkeit für jeden Fußgänger vorgenommen und das zugewiesene GameObject entsprechend manipuliert. Diese Herangehensweise hat den Nebeneffekt, dass an Stelle von potentiell Hunderten von Update-Aufrufen pro Frame nur noch ein Update-Aufruf für alle Fußgänger stattfindet, was durch den vermiedenen Overhead einen weiteren kleinen Performance-Gewinn bedeuten sollte. Eine Messung dieser einzelnen Verbesserung wurde aufgrund des grundlegend unterschiedlichen Aufbaus der beiden Systeme nicht durchgeführt.
In Abbildung A.3 ist ein Teil des Codes zur Pool-Verwaltung zu sehen. Zu Beginn werden durch die Funktion “CreatePoolObject” neue GameObjects basierend auf dem Fußgänger Prefab erstellt und dem Pool (Abb. A.3 Zeile 1) hinzugefügt. Wenn ein neuer Fußgänger geladen wurde, wird durch die

1 Ein Skinned Mesh Renderer kann einen Mesh anhand eines Skelettes verformen und animieren.

Funktion "AddPedestrianEntity" ein unbenutztes Pool-Objekt dem neuen Fußgänger zugewiesen und aus dem Pool entfernt. In Zeile 14-17 wird noch die Animation des GameObjects korrekt eingestellt.

5.2.3 *Kombinieren der Geometry Meshes*

In dem Basisprojekt wird für jede Wand und jeden Flur ein GameObject mit eigenem Mesh erstellt. Dies bedeutet, dass bei komplexer Geometrie möglicherweise tausende GameObjects erstellt werden müssen, die alle separate draw calls[2] generieren. Für einen merklichen Performancegewinn bietet sich an, diese draw calls zu kombinieren und wie schon in 5.2.2 erwähnt, sollte die Erstellung von neuen GameObjects möglichst vermieden werden.

Umgesetzt wird dies dadurch, dass der `FileLoader` beim Laden der Geometrie anstelle von GameObjects nur Meshes erzeugt und diese zu einem einzelnen Mesh kombiniert. Unity stellt für diese Zwecke das struct CombineInstance und die Mesh Methode CombineMeshes zur Verfügung [26]. Hiermit wird nun jeweils ein Mesh für den Flur, Treppen, Oberseite von Wänden und Seite von Wänden erzeugt und jedem davon einem Renderer zugewiesen. Diese Aufteilung erfolgt, damit die Meshes bei Bedarf auch als Collider Mesh[3] verwendet werden können, was für die Teleportation benötigt wird.

5.2.4 *Threaded Loading*

Das Laden der Geometrie dauert aufgrund der Dateigröße und des verbesserten Erstellens der Geometrie (siehe Sektion5.2.3) nur wenige Millisekunden und ist damit für diesen Anwendungsfall akzeptabel. Laden der Fußgängerdaten allerdings benötigt wegen der Benutzung von XML für viele Megabyte große Dateien mehrere Sekunden und kann so nicht ohne Anpassungen in einer VR Anwendung genutzt werden.

Daher wurde das Laden der Fußgängerdaten auf einen anderen Thread ausgelagert. Die hierfür zuständige Funktion "loadPedestrianFile" wurde zu einer Coroutine[27] umgebaut, was ihr erlaubt jedes Frame zu überprüfen, ob der in ihr erstellte Thread fertig ist. Coroutinen sind ein Unity Feature, welches Funktionen ermöglicht, die Exekution für das aktuelle Frame zu beenden und sie an der gleichen Stelle im nächsten Frame wieder aufzunehmen. Damit lassen sich beispielsweise teure Operationen über mehrere Frames strecken oder wie hier auf eine Resource warten. Durch diesen Aufbau kann die Anwendung im Hintergrund die Datei laden, ohne FPS Einbrüche zu verursachen. Zu sehen ist diese Funktionalität in Abbildung A.1. Zeilen 9-13 Zeigen Thread-Erstellung und Start. In Zeile 14-15 ist zu sehen, wie die Funktion den Programmfluss an die vorige Funktion zurückgibt, wenn der Thread noch nicht fertig ist.

2 Ein "draw call" ist ein Grafik API Befehl, welcher Objekte zeichnet.
3 Mit Collider Meshes werden in Unity Kollisionen berechnet.

5.2.5 *Zentrales Objekt für den Simulationsstatus*

Das Projekt soll eine experimentelle Arbeitsweise unterstützen und in Zukunft potentiell um mehrere GUI Varianten erweitert werden. Hierfür ist es wichtig, die auf der GUI dargestellten Informationen an einem Punkt zu verwalten und sie nicht auf mehrere Objekte zu verteilen. Um das zu ermöglichen, sind alle für den Abspiel-Vorgang notwendigen Variablen in ein zentrales GameState Objekt verlegt worden. Dieses Objekt besitzt sowohl die Variablen als private Member mit getter und setter Methoden, als auch ein Event für jedes Member, in dem sich andere Objekte eintragen können, um benachrichtigt zu werden, wenn sich der jeweilige Wert ändert. Hierfür werden C# Events mit Delegates benutzt (Abb. A.2 Zeile 3). Andere Objekte können ihre eigenen Handler bei diesen Events registrieren und werden bei einem Setter-Aufruf benachrichtigt (Abb. A.2 Zeile 7-12).

5.3 BETRACHTUNGSMODUS VERWALTUNG

Zur Umsetzung der in 4.2 genannten Betrachtungsmodi wurde ein System entwickelt, welches die Geometrie- und Benutzerposition durch Manipulation der Transforms und Parents verändert. Die Simulationsgeometrie und Fußgängerobjekte sind hierbei in einem Containerobjekt enthalten, welches je nach Modus an das Ende eines bestimmten Transformationszweigs gehängt wird. In Abbildung 5.2 ist der für den Betrachtungsmodus relevante Teil des Hierachiebaums abgebildet. Im Miniatur Modus ist der Simulationscontainer ein child des MiniatureNormalizer GameObjects, andernfalls des FullsizeView GameObjects. Die Verschachtelung der abgebildeten Miniatur Objekte dient der Interaktion mit der Miniatur und ist in Kapitel 5.4 weiter erläutert. Zur Verwaltung des Betrachtungsmodus gehören die folgenden Klassen:

KLASSEN

ViewModeManager
: Verwaltet den aktuellen Betrachtungsmodus, lädt den Modus bei Programmstart und koordiniert das Ändern des Modus zwischen `FullsizeViewMode` und `MiniatureViewMode`

FullsizeViewMode
: Speichert die letzte Position des Benutzers vor Verlassen des Echtgrößenmodus und stellt bei Aktivieren des Modus die Position des Benutzers wieder her und setzt das Parent GameObject der Simulationsobjekte zum FullsizeViewMode Container Objekt.

MiniatureViewMode
: Gleiche Funktion wie `FullsizeViewMode`, nur mit den Daten des Miniaturmodus.

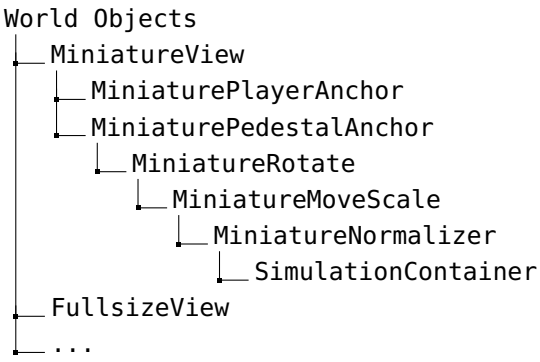

Abbildung 5.2: Hierachiebaum der Betrachtungsmodus-Architektur mit Simulationscontainer als child des Miniaturzweigs.

5.4 INTERAKTION MIT DER MINIATUR

5.4.1 *Transformationshierachie*

Für die Umsetzung der Interaktion wurde sich zunächst für eine geeignete Transformationshierachie entschieden. Da sich die Miniatur immer um die Mitte des Podests drehen soll statt um sich selbst, muss die Rotation vor der Translation stattfinden. Ein Transform arbeitet allerdings in umgekehrter Reihenfolge, weswegen die beiden Schritte auf verschachtelte Transforms aufgeteilt wurden. Die Translation muss unbeeinflusst von der Skalierung bleiben, daher muss die Skalierung nach der Translation angewandt werden, was mit der Reihenfolge eines Transforms übereinstimmt und daher keine weitere Verschachtelung erfordert. Dadurch ergeben sich das in Abbildung 5.2 aufgeführte GameObject-Paar "MiniatureRotate" und "MiniatureMoveScale".

Als nächstes muss die Größe der Miniatur normalisiert werden, damit alle geladenen Geometrien den vorgesehenen Betrachtungsbereich möglichst gut ausfüllen, ohne über ihn hinaus zu ragen. Hierfür wird unter das Translation/Skalierungs Transform ein weiteres Transform gehängt, dessen Skalierungskomponente beim Laden einer Geometrie so gesetzt wird, dass die Geometrie genau eine Unity Einheit (in dieser Anwendung äquivalent zu 1m) lang oder breit ist, je nachdem welche Dimension die längere ist. Dieses Transform ist im Hierachiebaum als "MiniatureNormalizer" gelistet.

Die Geometrie und alle Fußgänger selbst befinden sich in einem letzten verschachtelten "SimulationContainer" GameObject. Das Transform dieses GameObjects verschiebt die Geometrie so, dass die Mitte nach diesem Schritt in der Mitte der eingenommenen XZ Fläche und an der Y Achse zur untersten Y Koordinate verschoben liegt. Zur Veranschaulichung zeigt Abbildung 5.3 die Miniatur ohne Korrektur, wodurch der Nullpunkt im verwendeten Datensatz die Mitte bildet und somit für einen Offset sorgt, und daneben mit Korrektur.

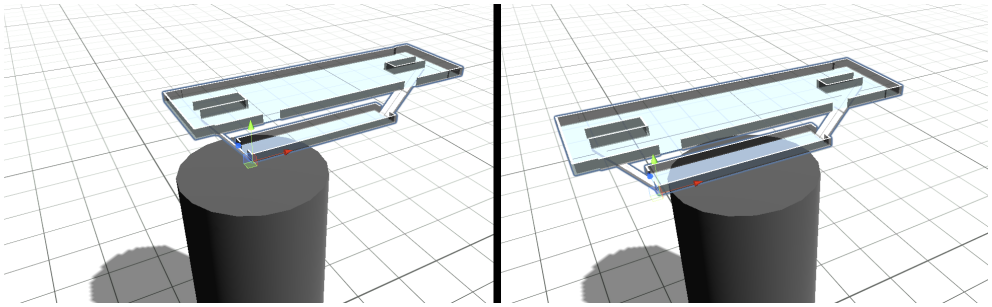

Abbildung 5.3: Geometrie Miniatur ohne (links) und mit (rechts) korrigierender Translation zur Mitte.

5.4.2 *Transform Berechnung*

Jeder Controller ist für einen Manipulationsmodus zuständig. Der linke Controller löst Translation aus während der rechte Controller die Rotation übernimmt. Beide Controller zusammen sorgen für die Skalierung. Der Status der Greifachse beider Controller wird für jedes Frame überprüft. Falls eine Achse betätigt oder losgelassen wurde, wird die aktuelle Position der Controller als Ausgangsposition für die Transformationen gespeichert. Damit wird ein fließender Übergang zwischen den Manipulationsmodi erreicht. Die Formeln für die Manipulationen sind in der folgenden Auflistung erläutert. Sofern nicht anderweitig angegeben, sind alle Transformationen als lokale Transformationen des in 5.4.1 genannten Transform-Paares zu verstehen.

Translation

Ein Vektor v berechnet sich aus der aktuellen Position des Controllers $cpos_t$ abgezogen von der Anfangsposition $cpos_0$. Die inverse Rotation der Miniatur rot^{-1} wird mit der X- und Z-Komponente von v multipliziert.

$$v = cpos_0 - cpos_t \tag{5.1}$$

$$translation = rot^{-1} * \begin{pmatrix} v_x \\ 0 \\ v_z \end{pmatrix} \tag{5.2}$$

Die Multiplikation mit der Rotation macht die Rotation um das Miniaturpodest rückgängig und wird durch die in 5.4.1 erläuterte Hierachie notwendig. Das Verwerfen von v_y sorgt dafür, dass jegliche Translation die Y Achse ausschließt und die Miniatur auf gleicher Höhe mit dem Podest bleibt. Der Höhenunterschied von Start- und aktueller Position wird, sofern die Option am rechten Arm Menü aktiviert wurde, dem im Hierachiebaum (Abb. 5.2) benannten MiniaturePedestalAnchor hin-

zugefügt, um sowohl Miniatur als auch Podest zeitgleich zu verschieben.

Rotation

Die neue Rotation wird mit dem Winkel α zwischen den beiden Vektoren berechnet, die von der globalen Podest Position pg zur globalen Controller-Startposition $cposg_0$ und zur globalen aktuellen Controllerposition $cposg_t$ verlaufen. Die Rotation selbst ergibt sich aus der Startrotation multipliziert mit dem Quaternion α_q welches sich aus dem Winkel α um die Y Achse errechnet. Von den Vektoren v und w wird jeweils nur die X- und Z-Komponente benutzt, um allein die Rotation um die Y Achse zu beachten und somit den Benutzer die Miniatur nicht kippen zu lassen.

$$v = (cposg_0 - pg) \tag{5.3}$$

$$w = (cposg_t - pg) \tag{5.4}$$

$$\alpha = \angle \begin{pmatrix} v_x \\ 0 \\ v_z \end{pmatrix} \begin{pmatrix} w_x \\ 0 \\ w_z \end{pmatrix} \tag{5.5}$$

$$rotation = rot_0 * \alpha_q \tag{5.6}$$

Skalierung

Die neue Skalierung berechnet sich durch die Startdistanz der beiden Controller, geteilt durch die aktuelle Distanz, multipliziert mit der Startskalierung.

$$skalierung = \frac{|(cposa_0 - cposb_0)|}{|(cposa_t - cposb_t)|} * scale_0 \tag{5.7}$$

Hierdurch kann der Benutzer mit der anfänglichen Distanz der Controller bestimmen, ob er feine oder grobe Skalierungsänderungen vornimmt.

5.5 UMSETZUNG DER GUI

Die hier verwendeten GUIs bestehen alle aus einem Unity Canvas und benutzen dementsprechend das Unity Event System. Für XR Funktionalität werden mehrere vom XR Interaction Toolkit zur Verfügung gestellte Komponenten benutzt[39]. Dem Event System GameObject wird ein `XRUIInputModule` hinzugefügt und jedes Canvas wird um einen `TrackedDeviceGraphicRaycaster` ergänzt. Die Bedienung eines solchen Canvas erfolgt durch den dem Controller zugehörigen `XRRayInteractor`. Somit werden Interaktionen vom Controller über das Canvas zum Event System weitergeleitet.

5.5.1 *Hauptmenü*

Das Hauptmenü und alle Untermenüs wurden jeweils als World Space Canvas mit den standardmäßigen Unity GUI Elementen gebaut. Als Textelement kommt TextMeshPro[33] Text zum Einsatz, welcher besonders für World Space UI deutlich bessere Resultate liefert als das Standard Textelement. Im Folgenden werden die implementierten Menüs gelistet.

Main Menu
Das Hauptmenü besitzt Buttons zum Laden von Simulationsdaten, Öffnen der Einstellungen, Beenden der Anwendung und Schließen des Menüs.

Settings
Ein simples Menü, welches Buttons für diverse Untermenüs anzeigt.

Arm Menu Settings
Das Einstellungsmenü für die Armmenüs lässt den Benutzer die beiden Menüs deaktivieren oder ihre Position ändern. Ebenso lassen sich hier die Winkel für die Blickerkennung anpassen. Genaueres dazu findet sich in Sektion 5.5.2.

Teleportation Settings
In diesem Menü kann aktuell ausgewählt werden, mit welchem Controller der Benutzer sich teleportieren kann.

Funktionsweise

BEDIENUNG: Der Benutzer kann mit den Controllern auf ein Canvas zeigen, wodurch ein Strahl sichtbar wird, der die Stelle des Cursors markiert. Durch Drücken des Triggers wird ein Klick ausgelöst, beim Halten des Triggers das Analog zu einem Halten eines Mausklicks.

HANDLER: Jedes der Menüs führt soweit möglich die notwendige Funktion im über den Inspector konfigurierten Button Handler aus und besitzt wenn nötig ein entsprechendes Handler Skript, welches Eingaben in der GUI verarbeitet. Diese Skripte übernehmen auch das Laden der letzten Einstellungen bei Programmstart, das Speichern der Einstellungen und das Registrieren von Event Listenern im GameState Objekt, das in 5.2.5 beschrieben wurde.

NAVIGATION: Das Öffnen der Untermenüs funktioniert, indem der entsprechende Knopf das Canvas des Untermenüs aktiviert, es so sichtbar wird und das vorherige Menü überlagert. Der Zurück Button deaktiviert das Canvas wieder, womit das ursprüngliche Canvas wieder komplett sichtbar und bedienbar wird. Menüs und Untermenüs sind auf der Z-Achse so verschoben, dass immer das Untermenü sichtbar ist.

Dateibrowser

Aufgrund des zeitlichen Aufwands, einen gut funktionierenden Dateibrowser selbst zu schreiben, wurde hierfür das SimpleFileBrowser Asset von Süleyman Yasir Kula[15] benutzt. Dieses wurde um eine Instanz der Unity XR Komponente `TrackedDeviceGraphicRaycaster` erweitert und als World Space Canvas in die Szene eingefügt.

5.5.2 *Armmenüs*

Die Armmenüs bestehen jeweils aus einem kleinen Canvas und können wie auch das Hauptmenü durch Zeigen und Klicken bedient werden und besitzen ebenfalls Handler Skripte, welche für die Ausführung der Eingaben zuständig sind.

Da die Menüs an den Armen befestigt sein sollen, besitzen die Controller-Anker GameObjects, deren Transform die endgültige Ausrichtung der Menüs bestimmt. Für jeden Controller gibt es einen vorderen Anker, welcher sich vor der Handfläche befindet und von ihr wegzeigt, sowie einen hinteren Anker, welcher sich am Handrücken befindet und von diesem wegzeigt 5.1.

In jedem Frame wird das Transform der Menüs zu dem des für das jeweilige Menü konfigurierten Ankers gesetzt. Dies findet, wie auch das Anzeigen der Menüs, in dem hierfür geschriebenen `ArmPanelController` Skript statt.

Anzeigen der Menüs erfolgt durch Berechnung der Winkel zwischen Menü und Kamera wie in Abbildung 5.4 dargestellt. Winkel α ist die Abweichung der Canvas Normale von der Luftlinie zwischen Canvas-Mitte und Kameramitte, Winkel β die Abweichung der Blickrichtung von der genannten Luftlinie. Ein dritter Winkel γ ist die relative Rotation um die Blickachse und Normale zwischen Canvas und Kamera. Nur wenn alle drei Winkel unter den konfigurierten Maxima liegen, wird das Canvas angezeigt. Während der Entwicklung haben sich die Maxima $\alpha = 30°$, $\beta = 60°$ und $\gamma = 30°$ als angenehm erwiesen und wurden daher als Standardwerte festgelegt.

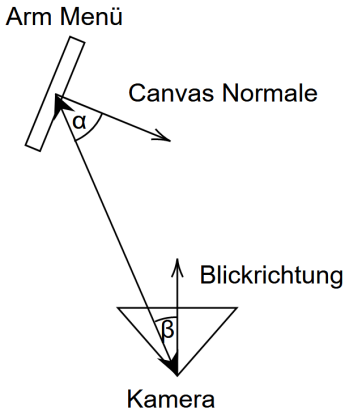

Abbildung 5.4: Relevante Winkel für das Anzeigen der Arm Menüs.

5.6 UMSETZUNG DER BEWEGUNG

Die Bewegung via Teleportation kann größtenteils durch das von Unity XR zur Verfügung gestellte Teleportationssystem umgesetzt werden. Dem XR-Rig werden die Unity XR-Skripte `LocomotionSystem`, `TeleportationProvider` und `SnapTurnProvider` hinzugefügt. Der `SnapTurnProvider` ermöglicht es dem User den Avatar zu rotieren, ohne sich selbst zu drehen.
Der `TeleportationProvider` verwaltet Teleportationsanfragen, die durch Zeigen eines Controllers auf ein GameObject mit einem `TeleportationArea` gestellt werden.

Das Auswählen eines Zielorts soll mittels Bogenwurf bewerkstelligt werden, was ein Problem darstellt, da das User Interface wie in 5.5 implementiert weiterhin mit einem geraden Strahl bedient werden soll. Hierfür werden dem XRRig noch ein weiteres Paar Controller hinzugefügt, deren `XRRayInteractor` und `LineRenderer` zu einem Bogen konfiguriert wurden. Diese sind in Abbildung 5.1 als "LeftHand Teleport Controller" und "RightHand Teleport Controller" aufgeführt. Es wurden zusätzlich zwei rote Kugeln in die Szene eingefügt, die zur Verdeutlichung der Zielposition des Strahls als Markierung eingesetzt werden. Abhängig davon, ob eine GUI bedient oder ein Teleportziel ausgewählt werden soll, schaltet ein zu jeden der neuen Teleport Controller hinzugefügtes `RayModeSwitcher` Skript dafür, dass das `XRInteractorLineVisual` Objekt des jeweils anderen Controllers an der gleichen Hand deaktiviert wird. Wenn dieses Objekt deaktiviert ist, kann der zugehörige `XRRayInteractor` keine Canvas Interaktionen mehr auslösen.

Zur intuitiven Unterscheidung ob der Benutzer eine GUI bedienen möchte oder ein Teleportziel auswählt, wird angenommen, dass der Benutzer zunächst immer eine GUI bedienen möchte. Da alle aktuell gängigen Controller Touch-Sensoren für alle Eingabeelemente verbauen, können diese zur Deutung der Absicht des Benutzers verwendet werden. Daher wird der Teleportstrahl nur aktiv, wenn der Joystick berührt wird. Ein Drücken des Joysticks löst dann den Teleportvorgang aus. Diese Logik ist im weiter oben genannten `RayModeSwitcher` umgesetzt, welcher das in 5.1.2 eingeführte Input System nutzt, um die `XRInteractorLineVisuals` zu aktivieren und deaktivieren.

Als Teleportationsziel im normalen Raum wird dem Flur-Objekt eine Instanz des Unity XR Skriptes `TeleportationArea` hinzugefügt. Für das Umsetzen des in 4.4 aufgeführten Teleportierens in die Geometrie wird dem Boden der Geometrie ein hierfür geschriebenes `GeometryTeleportationArea` Skript zugewiesen, welches wie `TeleportationArea` von `BaseTeleportationInteractable` erbt, aber eine modifizierte Funktion zum generieren von Teleportationsanfragen besitzt. Diese löst zusätzlich zur Teleportation auch den Wechsel des ViewModes zur Echtgrößenansicht des Modells durch den in 5.3 beschriebenen `ViewModeManager` aus. Damit sowohl der Teleportstrahl als auch der rote Zielmarker durch ihre Größe die Miniatur nicht verdecken, wenn man in sie hineinzielt, wurde das `RayMarkerScaler`

Skript erstellt, welches mit EventHandlern für die Skalierung sorgt. Abbildung A.4 zeigt den Code der beiden Handler, welche bei Anwählen und Abwählen der Miniatur die Skalierung von Marker und Strahl aktualisieren. Hierbei wird der Strahl auf die gleiche Skalierung gesetzt wie die Miniatur selbst (Abb. A.4 Zeile 5), sofern sie zwischen einem festen Minimum und Maximum liegt. Feststellen, ob tatsächlich die Miniatur angewählt wurde, findet über eine Abfrage der GameObject-Layer[4] statt (Abb. A.4 Zeile 3 und 13. Der Boden der Geometrie befindet sich auf layer 11).

4 GameObjects können Schichten zugewiesen werden, welche eine grobe Zuordnung in Kategorien ermöglicht. Dies wird vor allem von Kameras, Collidern und Raycasts benutzt, um bestimmte Objekttypen zu filtern.

6 FAZIT UND AUSBLICK

6.1 FAZIT

Im Rahmen dieser Arbeit wurde eine existierende Unity Anwendung zur Darstellung von Geometrie- und Personendaten um Virtual Reality Funktionalität erweitert. Hierbei wurden Bewegungsmethoden und GUI Varianten in populären Anwendungen untersucht und daraus ein Konzept für eine fertige VR Anwendung erstellt. Dieses Konzept wurde mit Unity XR als VR-Plugin implementiert. Weiterhin wurden viele Codefragmente der ursprünglichen Anwendung optimiert, sodass eine flüssige Darstellung und Interaktion mit den Simulationsdaten möglich ist. Ein paar wünschenswerte Aspekte wie animierte Hände und ansprechender Texturen und Modellen wurden wegen fehlender Expertise nicht umgesetzt, was jedoch die Features der Anwendung nicht beeinträchtigt.

6.2 AUSBLICK

Für ein besseres Benutzererlebnis können im Anschluss an diese Arbeit mehrere visuelle Verbesserungen umgesetzt werden. Das aktuell benutzte Handmodell ist noch nicht animiert und die für die Geometrie benutzten halbdurchsichtigen Texturen sollten optional durch undurchsichtige Texturen ersetzt werden, um Personen mit Höhenangst nicht zu belasten. Des Weiteren sollte noch ein Tutorial oder ausführliche Hilfe-Menüs eingebaut werden, um VR-Neulingen den Einstieg zu erleichtern.

Im Anschluss darauf kann das entstandene Projekt durch Mitarbeiter von JuPedSim auf Nutzbarkeit in verschiedenen Einsatzbereichen evaluiert werden. Denkbar ist der Einsatz bei Präsentationen für Investoren, der Analyse aufgenommener oder simulierter Personendaten sowie digitale Besichtigung der Geometrie. Für jeden dieser Anwendungsfälle kann das Projekt noch verbessert werden, um beispielsweise eine schönere Umgebung oder weitere Fortbewegungsmodi zu bieten.

Teil I

ANHANG

A

CODEFRAGMENTE

```
IEnumerator loadPedestrianFile(string filename) {
    if (!System.IO.File.Exists(filename)) {
        Debug.Log("Error: File " + filename + " not found.");
        yield break;
    }

    List<PedestrianEntity> result = new List<PedestrianEntity>();
    float totalTime = 0;
    var thread = new Thread(() =>
    // [Code zum parsen der XML Datei]
    );
    thread.Name = "PedestrianLoader";
    thread.Start();
    while (thread.IsAlive) {
        yield return null;
    }
    thread.Join();

    gameState.TotalTime = totalTime;
    foreach (var entity in result) {
        pedestrianSystem.AddPedestrianEntity(entity);
    }
}
```

Abbildung A.1: Ausschnitt der loadPedestrianFile Funktion, welche die Fußgänger XML Datei als Coroutine mit einem Thread lädt.

```
[SerializeField] private bool isPlaying = false;

public event Action<bool> isPlayingEvent;

public bool IsPlaying {
    get { return isPlaying; }
    set {
        if (value != isPlaying)
        {
            isPlaying = value;
            isPlayingEvent.Invoke(value);
        }
    }
}
```

Abbildung A.2: Ausschnitt aus der GameState Klasse. Code für eine von mehreren Statusvariablen.

```
1  LinkedList<PedestrianPoolObject> pedestrianPool = new LinkedList<
       PedestrianPoolObject>();
2  List<PedestrianEntity> pedestrianEntities = new List<
       PedestrianEntity>();
3
4  [...]
5
6  public void AddPedestrianEntity(PedestrianEntity entity) {
7      if (pedestrianPool.Count == 0) CreatePoolObject();
8
9      PedestrianPoolObject poolObject = pedestrianPool.Last.Value;
10     pedestrianPool.RemoveLast();
11     entity.poolObject = poolObject;
12     pedestrianEntities.Add(entity);
13
14     if (gameState.IsPlaying)
15         poolObject.componentAnimation.Play();
16     else
17         poolObject.componentAnimation.Stop();
18     poolObject.obj.SetActive(true);
19 }
20
21 [...]
22
23 private void CreatePoolObject() {
24     GameObject gameObject = (GameObject)Instantiate(Resources.Load(
           pedestrianPrefab.name));
25     gameObject.transform.parent = pedestrianParentObject.transform;
26     gameObject.SetActive(false);
27     PedestrianPoolObject poolObject = new PedestrianPoolObject(
           gameObject
28         , new Color(UnityEngine.Random.value, UnityEngine.Random.
               value, UnityEngine.Random.value),
29         pedestrianParentObject.transform);
30     pedestrianPool.AddLast(poolObject);
31 }
```

Abbildung A.3: Ausschnitt aus dem PedestrianSystem Skript, welcher die Logik zum Hinzufügen von neuen Fußgängern zeigt.

```
1  public void HandleHoverEnter(XRBaseInteractable interactable) {
2      // check for geometry layer
3      if (marker && interactable && interactable.gameObject.layer ==
           11) {
4          float scale = interactable.transform.lossyScale.x;
5          scale = Mathf.Clamp(scale, minimumScale, maximumScale);
6          marker.transform.localScale = new Vector3(scale, scale,
               scale);
7          lineVisual.lineWidth = defaultLineWidth * scale;
8      }
9  }
10
11 public void HandleHoverExit(XRBaseInteractable interactable){
12     // check for geometry layer
13     if (marker && interactable && interactable.gameObject.layer ==
           11) {
14         marker.transform.localScale = Vector3.one;
15         lineVisual.lineWidth = defaultLineWidth;
16     }
17 }
```

Abbildung A.4: Ausschnitt aus dem RayMarkerScaler Skript, welches die Teleportlinie zur Benutzung innerhalb der Miniatur skaliert.

LITERATUR

[1] Jiwan Bhandari, Paul MacNeilage und Eelke Folmer. "Teleportation without Spatial Disorientation Using Optical Flow Cues". In: *Proceedings of Graphics Interface 2018*. GI 2018. Toronto, Ontario, 2018, S. 162–167. ISBN: 978-0-9947868-3-8. DOI: 10.20380/GI2018.22.

[2] Nikolai Bockholt. *VR, AR, MR und was ist eigentlich Immersion?* Online, aufgerufen am 14.08.2020. URL: https://www.thinkwithgoogle.com/intl/de-de/marketingkanaele/innovative-technologien/vr-ar-mr-und-was-ist-eigentlich-immersion/.

[3] Daniel Büchele. *SumoVizUnity Github*. Online, aufgerufen am 26.08.2020. URL: https://github.com/danielbuechele/SumoVizUnity.

[4] Daniel Büchele. "Visualisierung von Fußgängersimulationsdaten auf Basis einer 3D-Game-Engine". Masterarbeit. Fakultät für Bauingenieur- und Vermessungswesen, Technische Universität München, Juli 2014. URL: https://raw.githubusercontent.com/danielbuechele/SumoVizUnity/master/thesis.pdf. Online, aufgerufen am 26.08.2020.

[5] Chris G. Christou und Poppy Aristidou. "Steering Versus Teleport Locomotion for Head Mounted Displays". In: *Augmented Reality, Virtual Reality, and Computer Graphics*. Hrsg. von Lucio Tommaso De Paolis, Patrick Bourdot und Antonio Mongelli. Cham: Springer International Publishing, 2017, S. 431–446. ISBN: 978-3-319-60928-7.

[6] Valve Corporation. *VR 2019 Top Sellers*. Online, aufgerufen am 16.08.2020. URL: https://store.steampowered.com/sale/2019_top_vr/.

[7] Valve Corporation. *Valve Index® Base Station*. Online, aufgerufen am 26.08.2020. URL: https://store.steampowered.com/app/1059570/Valve_Index_Base_Station/.

[8] Google Developers. *Degrees of freedom*. Online, aufgerufen am 26.08.2020. URL: https://developers.google.com/vr/discover/degrees-of-freedom.

[9] Nancy Gupton. *What's the Difference Between AR, VR, and MR?* Online, aufgerufen am 14.08.2020. URL: https://www.fi.edu/difference-between-ar-vr-and-mr.

[10] Philip Hammer. *Virtual Reality: Die Erschaffung neuer Welten*. Online, aufgerufen am 14.08.2020. URL: https://www.zukunftsinstitut.de/artikel/virtual-reality-die-erschaffung-neuer-welten/.

[11] Jason Jerald. *The VR Book: Human-Centered Design for Virtual Reality*. Association for Computing Machinery und Morgan & Claypool, Okt. 2015, S. 45. ISBN: 9781970001129. URL: https://doi.org/10.1145/2792790.

[12] JuPedSim. *JuPedSim*. Online, aufgerufen am 26.08.2020. URL: https://www.jupedsim.org/.

[13] *KapaKrit*. Online, aufgerufen am 26.08.2020. URL: http://www.kapakrit.de/kapakrit/DE/Projekt/projekt_node.html.

[14] J. Kim, W. Kim, S. Ahn, J. Kim und S. Lee. "Virtual Reality Sickness Predictor: Analysis of visual-vestibular conflict and VR contents". In: *2018 Tenth International Conference on Quality of Multimedia Experience (QoMEX)*. Cagliari, Italy, 2018, S. 1–6. DOI: 10.1109/QoMEX.2018.8463413.

[15] Süleyman Yasir Kula. *Runtime File Browser*. Online, aufgerufen am 02.05.2020. URL: https://assetstore.unity.com/packages/tools/gui/runtime-file-browser-113006.

[16] Facebook Technologies LLC. *Our first all-in-one gaming headset.* Online, aufgerufen am 26.08.2020. URL: https://www.oculus.com/quest/.

[17] Microsoft. *Inside-out tracking*. Online, aufgerufen am 26.08.2020. URL: https://docs.microsoft.com/en-us/windows/mixed-reality/enthusiast-guide/tracking-system.

[18] K E Money. "Motion sickness." In: *Physiological Reviews* 50.1 (1970). PMID: 4904269, S. 1–39. DOI: 10.1152/physrev.1970.50.1.1. URL: https://doi.org/10.1152/physrev.1970.50.1.1.

[19] Facebook Technologies LLC Oculus VR. *Locomotion*. Online, aufgerufen am 19.08.2020. URL: https://developer.oculus.com/learn/bp-locomotion/.

[20] Facebook Technologies LLC Oculus. *Unity Sample Framework*. Online, aufgerufen am 27.08.2020. URL: https://developer.oculus.com/documentation/unity/unity-sample-framework/.

[21] Yun Suen Pai und Kai Kunze. "Armswing: Using Arm Swings for Accessible and Immersive Navigation in AR/VR Spaces". In: New York, NY, USA: Association for Computing Machinery, 2017. ISBN: 9781450353786. DOI: 10.1145/3152832.3152864. URL: https://doi.org/10.1145/3152832.3152864.

[22] *SumoVisUnity JuPedSim Github*. Online, aufgerufen am 26.08.2020. URL: https://github.com/chraibi/SumoVizUnity.

[23] *SumoVizUnity accu:rate Github*. Online, aufgerufen am 26.08.2020. URL: https://github.com/accu-rate/SumoVizUnity.

[24] Unity Technologies. *Game engines-how do they work?* Online, aufgerufen am 16.08.2020. URL: https://unity3d.com/what-is-a-game-engine.

[25] Unity Technologies. *Unity Documentation, Canvas*. Online, aufgerufen am 15.08.2020. URL: https://docs.unity3d.com/2020.1/Documentation/Manual/UICanvas.html.

[26] Unity Technologies. *Unity Documentation, CombineMeshes*. Online, aufgerufen am 03.07.2020. URL: https://docs.unity3d.com/ScriptReference/Mesh.CombineMeshes.html.

[27] Unity Technologies. *Unity Documentation, Coroutine*. Online, aufgerufen am 03.07.2020. URL: https://docs.unity3d.com/Manual/Coroutines.html.

[28] Unity Technologies. *Unity Documentation, GameObject*. Online, aufgerufen am 06.08.2020. URL: https://docs.unity3d.com/ScriptReference/GameObject.html.

[29] Unity Technologies. *Unity Documentation, InputDevices*. Online, aufgerufen am 20.08.2020. URL: https://docs.unity3d.com/ScriptReference/XR.InputDevices.html.

[30] Unity Technologies. *Unity Documentation, MonoBehavior*. Online, aufgerufen am 06.08.2020. URL: https://docs.unity3d.com/ScriptReference/MonoBehaviour.html.

[31] Unity Technologies. *Unity Documentation, Prefab*. Online, aufgerufen am 20.08.2020. URL: https://docs.unity3d.com/Manual/Prefabs.html.

[32] Unity Technologies. *Unity Documentation, Scripting*. Online, aufgerufen am 05.08.2020. URL: https://docs.unity3d.com/Manual/ScriptingSection.html.

[33] Unity Technologies. *Unity Documentation, TextMeshPro*. Online, aufgerufen am 20.08.2020. URL: https://docs.unity3d.com/Manual/com.unity.textmeshpro.html.

[34] Unity Technologies. *Unity Documentation, Transform*. Online, aufgerufen am 03.07.2020. URL: https://docs.unity3d.com/ScriptReference/Transform.html.

[35] Unity Technologies. *Unity Documentation, XR Input*. Online, aufgerufen am 12.04.2020. URL: https://docs.unity3d.com/Manual/xr_input.html.

[36] Unity Technologies. *Unity Documentation, XR*. Online, aufgerufen am 03.04.2020. URL: https://docs.unity3d.com/Manual/XR.html.

[37] Unity Technologies. *Unity Game Engine*. Online, aufgerufen am 26.08.2020. URL: https://unity.com/.

[38] Unity Technologies. *Unity Packages, Oculus XR*. Online, aufgerufen am 20.08.2020. URL: https://docs.unity3d.com/Manual/com.unity.xr.oculus.html.

[39] Unity Technologies. *Unity Packages, Unity XR Interaction Toolkit*. Online, aufgerufen am 07.08.2020. URL: https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@0.9/manual/index.html.

[40] Unity Technologies. *Unity Packages, XR Interaction Subsystems*. Online, aufgerufen am 20.08.2020. URL: https://docs.unity3d.com/Manual/com.unity.xr.interactionsubsystems.html.

[41] Unity Technologies. *Unity Packages, XR Management*. Online, aufgerufen am 20.08.2020. URL: https://docs.unity3d.com/Packages/com.unity.xr.management@3.2/manual/index.html.

[42] Thomas Krogh-Jacobsen Unity Technologies. *Unity 2019.3 is now available*. Online, aufgerufen am 26.08.2020. URL: https://blogs.unity3d.com/2020/01/28/unity-2019-3-is-now-available/.

[43] Oculus VR. *VR Sickness, The Rift, and How Game Developers Can Help*. Online, aufgerufen am 19.08.2020. URL: https://developer.oculus.com/blog/vr-sickness-the-rift-and-how-game-developers-can-help/.

[44] Séamas Weech, Sophie Kenny und Michael Barnett-Cowan. "Presence and Cybersickness in Virtual Reality Are Negatively Related: A Review". In: *Frontiers in Psychology* 10 (2019), S. 158. ISSN: 1664-1078. DOI: 10.3389/fpsyg.2019.00158. URL: https://www.frontiersin.org/article/10.3389/fpsyg.2019.00158.

[45] Marvin Lars Weisbrod. *SumoVisUnity*. Online, aufgerufen am 26.08.2020. URL: https://github.com/marvinweisbrod/SumoVizUnity.

[46] Preston Tunnell Wilson, William Kalescky, Ansel MacLaughlin und Betsy Williams. "VR Locomotion: Walking > Walking in Place > Arm Swinging". In: New York, NY, USA: Association for Computing Machinery, 2016. ISBN: 9781450346924. DOI: 10.1145/3013971.3014010. URL: https://doi.org/10.1145/3013971.3014010.

[47] *accu:rate GmbH*. Online, aufgerufen am 26.08.2020. URL: https://www.accu-rate.de/en/.