

Hochschule Osnabrück

University of Applied Sciences

**Fakultät
Ingenieurwissenschaften und Informatik**

BACHELORARBEIT

Evaluation eines Controllers für die Fortbewegung in einer Virtual Reality anhand einer prototypischen Anwendung für mobile Endgeräte

Autor: Tobias Busch

tobiasbusch@live.de

Fach-Professor: Prof. Dr. Frank M. Thiesing

Zweitprüfer: Andree Josef

Abgabedatum: 09.02.2015

I Kurzfassung

Durch die Entwicklung von Smartphones und des allgemein technologischen Fortschrittes sind die Möglichkeiten die reale Welt hinter sich zu lassen und in eine virtuelle Realität einzutauchen vielfältiger als jemals zuvor. Die Technologien befinden sich im steten Wandel und werden immer weiter entwickelt. Was vor kurzem nur mit speziellen Geräten erreichbar war, ist nun durch Smartphones möglich. Aufgrund der immer höheren Leistung und der Fülle an integrierten Sensoren können Virtual Reality Applikationen auf Smartphones umgesetzt werden und mit entsprechenden Halterungen ist es für den Nutzer möglich ohne großen Kostenaufwand virtuelle Realität zu erleben.

Die Herausforderungen auf die Entwickler und Umsetzer stoßen, ist die Fortbewegung in einer virtuellen Realität. Controller für aktuelle Ausgabegeräte wie die Oculus Rift sind vorhanden und ohne Probleme anzuschließen, was bei mobilen Geräten nicht immer ohne weiteres möglich ist. Außerdem sollen Controller möglichst unsichtbar oder wenig störend für den Nutzer sein, um die Immersion nicht zu verringern.

In dieser Arbeit werden Controller aufgetan, analysiert und anhand bestimmter Kriterien bewertet. Das Ergebnis ist eine Übersicht von Controllern für mobile Geräte sowie eine tiefergehende Analyse von in Entwicklung befindlichen Ansätzen, die sich für die Bewegungssteuerung in virtuellen Welten eignen. Des Weiteren ist eine prototypische Applikation umgesetzt worden, um vorhandene Controller testen zu können.

II Abstract

The current development of mobile devices and in general technological advances in this field created many diverse possibilities to leave the real world behind and immerse in a virtual reality. The technologies are constantly changing and evolving. And this advance in technology changed the way how the user can experience virtual reality. There is no need for special equipment anymore. Due to increased performance and accuracy of integrated sensors of mobile devices virtual reality applications can be implemented for these devices. With special cases the user can experience virtual reality without great expenditure.

The challenge developers encounter is the implementation of locomotion in virtual reality. Controllers for current output devices such as the Oculus Rift are available and can connect without problems, which is not always possible on mobile devices. In addition the controller should not interfere with the feeling of immersion or disturb the user.

In this work, these controllers are researched, analyzed and evaluated based on specific criteria. The result is a list of controllers for mobile devices, as well as a more in-depth analysis of in-development approaches that are useful in virtual worlds for motion control. In addition a prototype application has been implemented to test found controllers.

III Inhaltsverzeichnis

1	EINLEITUNG.....	1
1.1	EINFÜHRUNG IN DIE THEMATIK.....	1
1.2	ZIELE DER ARBEIT.....	2
1.3	VORSTELLUNG DES UNTERNEHMENS	2
1.4	AUFBAU DES BERICHTES.....	3
2	GRUNDLAGEN / STAND DER TECHNIK.....	4
2.1	VIRTUAL REALITY.....	4
2.1.1	<i>Wahrnehmung von Bewegung</i>	4
2.1.2	<i>Ausgabegeräte</i>	5
2.1.3	<i>Eingabegeräte</i>	6
2.1.4	<i>Bewegungskontrolle mit Eingabegeräten</i>	9
2.2	VIRTUAL REALITY FÜR MOBILE GERÄTE	14
2.2.1	<i>Boxx3D</i>	15
2.2.2	<i>Google Cardboard</i>	15
2.2.3	<i>Gear VR</i>	16
2.3	UNITY.....	16
2.3.1	<i>Programmierung über Skripte</i>	17
2.3.2	<i>Objekte im 3D-Raum</i>	22
2.3.3	<i>Kamera</i>	23
2.3.4	<i>Physik</i>	24
2.3.5	<i>Inputmöglichkeiten</i>	25
2.3.6	<i>GUI</i>	29
2.3.7	<i>Prefabs</i>	31
2.3.8	<i>UnityPackage</i>	32
2.3.9	<i>Debugging und Performance</i>	33
2.3.10	<i>Build Prozess</i>	35
3	ANALYSE.....	36
3.1	SYSTEMIDEE	36
3.2	STAKEHOLDER	36
3.2.1	<i>Stakeholdermap</i>	37
3.3	ZIELE	38
3.3.1	<i>Musskriterien</i>	38
3.3.2	<i>Wunschkriterien</i>	39
3.3.3	<i>Abgrenzungskriterien</i>	39
3.4	FUNKTIONALE ANFORDERUNGEN.....	40
3.4.1	<i>Use-Case</i>	40

3.4.2	<i>Anforderungen</i>	41
3.5	NICHT FUNKTIONALE ANFORDERUNGEN.....	41
3.5.1	<i>Technologisch</i>	42
3.5.2	<i>Qualitätsanforderungen</i>	42
3.6	ENTWICKLUNGSUMGEBUNG.....	43
3.7	TESTS.....	43
3.7.1	<i>Testgeräte</i>	44
3.7.2	<i>Testfälle</i>	44
4	ANALYSE DER CONTROLLER.....	45
4.1	GAMEPAD: PLAYSTATION 3 CONTROLLER.....	45
4.2	MAGNETFELD: CARDBOARD	46
4.3	INERTIALENSOREN: SPHERO UND 3DRUDDER	48
4.3.1	<i>Sphero</i>	48
4.3.2	<i>3DRudder</i>	49
4.4	OPTISCHES TRACKING: RGBD KAMERAS	50
4.4.1	<i>Leap Motion</i>	50
4.4.2	<i>Kinect</i>	50
4.4.3	<i>Project Tango</i>	52
4.5	ODT: VIRTUIX OMNI.....	52
5	PROTOTYPISCHE APP.....	54
5.1	CONTROLLER: GAMEPAD UND CARDBOARD.....	54
5.2	PREFABS	54
5.3	IMPLEMENTIERUNG	55
5.3.1	<i>Allgemein</i>	56
5.3.2	<i>Cardboard</i>	57
5.3.3	<i>Gamepad</i>	58
6	TESTS	59
6.1	GAMEPAD.....	59
6.2	CARDBOARD	60
7	BEWERTUNG	62
7.1	ALLGEMEIN.....	62
7.2	VERFÜGBARE CONTROLLER.....	63
7.3	PROTOTYPEN.....	65
7.4	ÜBERSICHT	66
7.5	ERGEBNIS.....	67
8	ZUSAMMENFASSUNG	68
8.1	ERGEBNIS.....	68

III Inhaltsverzeichnis

V

8.2 FAZIT	69
8.3 AUSBLICK.....	69
A REFERENZEN	70
B INHALT DER CD.....	73

IV Abbildungsverzeichnis

ABB. 2.1 PRINZIPIELLE BESTANDTEILE EINES HMDs [DÖR13]	6
ABB. 2.2 MÖGLICHE FEHLER BEI DER DATENAUFNAHME DER POSITION EINES BEWEGTEN OBJEKTES (SCHWARZE LINIE): AUFNAHME MIT LATENZ (BLAU), MIT DRIFT (ORANGE), MIT RAUSCHEN (GRÜN) DARGESTELLT ÜBER DIE ZEIT (HORIZONTALE ACHSE) [DÖR13]	8
ABB. 2.3 ANSICHT VON DER AKTUELLEN KINECT [COM] UND LEAP MOTION [COM]	10
ABB. 2.4 ODT MIT HALTERUNG FÜR DEN NUTZER [COM]	11
ABB. 2.5 SPHERO[ORB] UND 3DRUDDER [RUD]	13
ABB. 2.6 PLAYSTATION 3 CONTROLLER [BUS]	14
ABB. 2.7 VERSCHIEDENE ANSICHTEN DER BOXX3D [ETA]	15
ABB. 2.8 VERSCHIEDENE ANSICHTEN DER CARDBOX [BUS]	15
ABB. 2.9 ANSICHT DER GEAR VR [COM]	16
ABB. 2.10 INSPECTORANSICHT NACH ZUWEISUNG EINER PUBLIC-VARIABLEN.....	19
ABB. 2.11 ANSICHT DES INPUT-MANAGERS.....	26
ABB. 2.12 BEISPIEL EINER GUI IM SCENE UND GAME VIEW	30
ABB. 2.13 ANCHOR-VORGABEN.....	31
ABB. 2.14 ANSICHT DES PROFILERS: DARSTELLUNG DER PERFORMANCE EINES TESTGERÄTES	34
ABB. 2.15 ANSICHT DER BUILD SETTINGS FÜR ANDROID.....	35
ABB. 3.1 USE-CASE DIAGRAMM.....	40
ABB. 4.1 MAGNETIC SWITCH DES CARDBOARD [BUS]	46
ABB. 4.2 VIRTUIX OMNI [COM].....	52
ABB. 5.1 PREFABS FÜR DAS LABYRINTH	54
ABB. 5.2 ANSICHT DES FERTIGEN LABYRINTHS	55
ABB. 5.3 ANSICHT EINES TRIGGER-OBJEKTES VOR DEM ZIEL DES LABYRINTHS	56

V Tabellenverzeichnis

TABELLE 3.1 ÜBERSICHT DER STAKEHOLDER.....	37
TABELLE 3.2 STAKEHOLDERMAP.....	37
TABELLE 3.3 MUSSKRITERIEN.....	39
TABELLE 3.4 WUNSCHKRITERIEN.....	39
TABELLE 3.5 FUNKTIONALE ANFORDERUNGEN NACH RUPP [RS14].....	41
TABELLE 3.6 QUALITÄTSANFORDERUNGEN.....	42
TABELLE 3.7 ÜBERSICHT DER TESTGERÄTE.....	44
TABELLE 3.8 TESTFÄLLE	44
TABELLE 6.1 TESTFÄLLE DER APP	59
TABELLE 6.2 TESTFÄLLE DES CONTROLLERS	60
TABELLE 6.3 TESTFÄLLE DER APP	60
TABELLE 6.4 TESTFÄLLE DES CONTROLLERS	61
TABELLE 7.1 ÜBERSICHT DER CONTROLLER	66
TABELLE B.1 INHALT DER CD	73

VI Listings

LISTING 2.1 GRUNDGERÜST EINES C#-SKRIPTS.....	18
LISTING 2.2 ZUGRIFF AUF EIGENE KOMPONENTE EINES OBJEKTES	18
LISTING 2.3 INSPECTOR ZUWEISUNG EINER PUBLIC VARIABLEN.....	19
LISTING 2.4 AUFBAU EINER COROUTINE.....	19
LISTING 2.5 BEISPIEL COROUTINE.....	20
LISTING 2.6 BEISPIEL FÜR INVOKE.....	20
LISTING 2.7 SPEICHERN VON DATEN	21
LISTING 2.8 LADEN VON DATEN	21
LISTING 2.9 DEBUG-MÖGLICHKEITEN.....	21
LISTING 2.10 BEISPIELE FÜR DIE ERZEUGUNG EINES VEKTORS.....	22
LISTING 2.11 ZUGRIFF AUF EINGABEN DURCH GETAXIS	28
LISTING 2.12 ZUGRIFF AUF EINGABEN MIT GETBUTTON [SEI14].....	28
LISTING 2.13 ZUGRIFF AUF EINGABEN MIT GETKEY.....	29
LISTING 2.14 BEISPIEL FÜR INPUT.ACCELERATION	29
LISTING 2.15 INSTANZERZEUGUNG ÜBER CODE	32
LISTING 2.16 LOGGING ÜBER ADB LOGCAT	34
LISTING 4.1 BEISPIEL FÜR DEN ZUGRIFF AUF DEN MAGNETIC SWITCH	47
LISTING 4.2 AKTIVIERT DIE ÜBERTRAGUNG DER SENSORWERTE.....	48
LISTING 4.3 AUSZUG AUS DEM ZUGRIFF AUF SENSORWERTE [@SPH2]	49
LISTING 5.1 AUSLÖSUNG EINES TRIGGERS.....	57
LISTING 5.2 NEUSTART DES LEVELS	57
LISTING 5.3 PÜRFUNG DES MAGNETIC SWITCH UND AKTIVIERUNG VON AUTOWALK.....	58
LISTING 5.4 AKTIVIERUNG DER EINGABEN DURCH DAS GAMEPAD.....	58

VII Abkürzungsverzeichnis / Glossar

Abk.	Begriff	Erklärung
-	Controller	bezeichnet ein Eingabegerät für die Steuerung von Computerspielen, in diesem Fall alle Eingabegeräte, die eine Bewegungssteuerung in einer Virtual Reality ermöglichen
-	Drift	sich aufaddierende Fehler
-	Gyroskop	Kreiselkompass, dient der genauen Lagebestimmung
-	Namespace	Organisationsstrukturen, die Klassen nach Zusammengehörigkeit gliedern und zusammenfassen
-	RGBD-Kamera	eine Kombination aus Farb- und Tiefenkamera
-	Rotation	Drehung eines Objektes über drei Winkel
-	Translation	Verschiebung eines Objektes über drei Achsen im Raum
ADB	Android Debug Bridge	
App	mobile Anwendung	Eine Anwendung für mobile Geräte, wie Smartphones oder Tablets
DOF	Degrees of Freedom/Freiheitsgrad	beschreibt die Bewegungsmöglichkeiten eines Körpers
FPP	First-person perspective/Egoperspektive	Darstellung der Spielwelt durch die Augen der Spielfigur

GUI	Graphical Unser Interface/grafische Benutzeroberfläche	Die Oberfläche, die dem Nutzer Echtzeitinformationen gibt, wird auch für Menüs genutzt
HMD	Head-Mounted Display	ein auf dem Kopf des Nutzers befestigtes Gerät welche einen Bildschirm enthält der vor die Augen des Nutzers platziert ist
ODT	Omnidirectional Treadmill	ein Laufband, was die Bewegung in mehrere Richtungen erlaubt
VR	Virtual Reality/virtuelle Realität	eine virtuelle Welt, die dem Nutzer das Gefühl der Immersion gibt

VIII Danksagung

Zunächst möchte ich mich an dieser Stelle bei all denjenigen bedanken, die mich während der Anfertigung dieser Bachelor-Arbeit unterstützt und motiviert haben.

Ganz besonders gilt Monika Mewes Dank, die viele Stunden investiert hat Korrektur zu lesen und als Fachfremde immer wieder aufzeigen konnte, wo Erklärungsbedarf bestand.

Genauso gilt mein Dank Herrn Prof. Dr. Frank M. Thiesing, der mich und meine Arbeit betreut hat.

Auch meine Vorgesetzten und Kollegen bei der Firma Die Etagen haben maßgeblich dazu beigetragen, dass diese Bachelorarbeit nun so vorliegt. Ich danke Ihnen, dass Sie mir die Möglichkeit gegeben haben, bei Ihnen zu forschen und zu arbeiten.

Nicht zuletzt gebührt meinen Eltern Dank, da sie mich während des Studiums in jeder Hinsicht unterstützt haben.

1 Einleitung

Die Entwicklung von Anwendungen für mobile Endgeräte, die dem Nutzer die Möglichkeit bieten in einer virtuellen Realität (VR) einzutauchen, ist mit aktuellen Smartphones und Entwicklungsumgebungen möglich. Herausforderungen, die bei der Weiterentwicklung und der Erzeugung immersiver Nutzererfahrungen entstehen, liegen in der Interaktion mit Elementen in der VR sowie die Umsetzung von intuitiven Möglichkeiten der Fortbewegung.

Die Arbeit beschäftigt sich mit der Thematik der Fortbewegung in einer VR und welche Möglichkeiten und Hardware aktuell vorhanden sind, um eine Steuerung zu ermöglichen. Diese Ansätze werden hier vorgestellt und verglichen.

1.1 Einführung in die Thematik

Mit der wachsenden Zahl an Anwendungen, die die Möglichkeit bieten in VR einzutauchen, steigt auch die Nachfrage an VR-ermöglichen Geräten. Diese sollen ein Gefühl der Immersion erzeugen und möglichst kostengünstig sein.

Mit der Entwicklung immer leistungsfähigerer Smartphones ist es nun möglich eine VR zu erzeugen und diese auf dem mobilen Gerät darzustellen. Mithilfe von entsprechenden Gehäusen, die das Smartphone halten und weitestgehend Einflüsse von der Realität ausblenden, ist es möglich ein immersives Gefühl zu erzeugen.

Im Gegensatz zu Head-Mounted Displays sind keine Bildschirme im Gehäuse integriert sondern, das Smartphone wird für die Darstellung genutzt. Es wird auf die integrierten Sensoren des Smartphones zurückgegriffen, um Rotationsbewegungen des Kopfes festzustellen und in die VR zu übertragen.

Es ist also möglich sich in einer VR umzuschauen. Die Herausforderung liegt in der Fortbewegung. Lösungen dafür werden aktuell entwickelt und getestet. Dabei gibt es verschiedene Ansätze wie gängige Gamecontroller, Laufbänder oder das Verfolgen des Nutzers mit Kameras.

1.2 Ziele der Arbeit

Die Arbeit soll Einblick in aktuelle Technologien in Bezug auf VR geben. Erstes Ziel ist die Evaluation eines Controllers anhand von einer festgelegten Bewertungsskala. Der Controller muss bestimmte Kriterien erfüllen, damit eine Nutzung ermöglicht ist. Der Controller soll möglichst intuitiv zu nutzen und einfach mit dem Smartphone zu verbinden sein.

Das zweite Ziel ist die Anbindung des Controllers an ein Smartphone. Das Smartphone soll die Eingaben des Controllers entgegennehmen und in die VR übersetzen. Dabei sollen die Herausforderungen bei der Verbindungsherstellung hervorgehoben und benötigte Software vorgestellt werden.

Das dritte Ziel ist die Erstellung einer prototypischen VR Applikation, um Funktionalitäten des evaluierten Controllers zu testen und die Usability zu überprüfen.

1.3 Vorstellung des Unternehmens

Die Etagen GmbH ist eine Full-Service Werbeagentur mit Hauptsitz in Osnabrück die 1998 gegründet wurde.

Neben dem Hauptsitz existieren noch zwei Standorte in Hamburg und Berlin. Der Standort Berlin ist die Effekt-Etage und realisiert 3D-, Installations- und Film-Projekte und ist mehr im Bereich der visuellen Medien anzusiedeln. Der Schwerpunkt in Osnabrück und Hamburg ist die Erstellung komplexer Kommunikationsmodelle auf dem Gebiet der Digitalen Medien und klassischem Corporate Design.

Im Speziellen bieten sie Leistungen in den Bereichen Corporate Design, Brand Identity, Klassische Kommunikation, Webapplikationen, Mobile Applications, Augmented Reality und E-Commerce an.

Das Unternehmen beschäftigt 40 Mitarbeiter in unterschiedlichen Abteilungen. Zu diesen gehören Projektleitung, Animation, Klassik/Digital Design und Programmierung.

1.4 Aufbau des Berichtes

Der Bericht lässt sich in fünf Teile gliedern. Der erste Teil behandelt die Grundlagen und den aktuellen Stand der Technik. Hier werden Begriffe erläutert, die Entwicklungsumgebung und Arten von Controller vorgestellt und weitere genutzte Werkzeuge aufgezeigt.

Der zweite Teil beschäftigt sich mit der Analyse der Anforderungen an den Controller und die VR Applikation.

Im dritten Teil findet eine Analyse der Controller statt. Verfügbare Eingabegeräte werden dabei vorgestellt und im darauffolgenden Schritt bewertet.

Der vierte Teil beschäftigt sich mit der Umsetzung der VR Applikation und die Anbindung des Controllers an das Smartphone. Des Weiteren werden durchgeführte Tests vorgestellt und die Ergebnisse aufgezeigt.

Der letzte Teil fasst alle Ergebnisse zusammen und gibt einen Ausblick auf mögliche zukünftige sowie in Entwicklung befindliche Technologien.

2

Grundlagen / Stand der Technik

Dieses Kapitel dient der Schaffung von Grundlagen sowie einem Verständnis der genutzten Werkzeuge. Es findet eine detaillierte Einführung in diese statt und auftretende Begriffe werden erläutert. Es werden aktuelle sowie als Prototyp vorhandene Technologien vorgestellt und näher beleuchtet. Dabei wird auf den im Projekbericht gewonnenen Erkenntnissen aufgebaut.

2.1 Virtual Reality

VR ist die Ersetzung der Realität [Dör13]. Der nächste Schritt, nachdem die Realität ersetzt worden ist, besteht in der Schaffung einer möglichst glaubhaften VR, die dem Nutzer das Gefühl gibt sich in einer neuen Umgebung zu befinden. Dafür müssen erst bestimmte Kriterien erfüllt sein. Zwei Kriterien sind die Wahrnehmung von Bewegung und die Bewegungskontrolle. Beide im Zusammenspiel steigern das Gefühl der Immersion für den Nutzer enorm.

2.1.1 Wahrnehmung von Bewegung

Für die Wahrnehmung von Bewegung muss die VR so verändert werden, dass dem Nutzer ein Gefühl der z.B. Vorwärtsbewegung vermittelt wird.

Aus physikalischer Sicht ist Bewegung definiert als Ortsveränderung über einen bestimmten Zeitraum. Für den Menschen bedeutet das, dass ein auf der Netzhaut auftreffendes Bild verschoben wird und so der entsprechende Reiz entsteht. Neben dieser so genannten retinalen Verschiebung werden Bewegungen mit einer bestimmten Geschwindigkeit in eine bestimmte Richtung wahrgenommen, was als physikalische Geschwindigkeit definiert ist. Eine weitere Art der Wahrnehmung ist der vestibuläre Sinn, der Gleichgewichtsinn. Dieser sorgt dafür, dass lineare Beschleunigungen und Drehbeschleunigungen wahrgenommen werden können [Dör13].

Um jetzt einem Nutzer das Gefühl der Bewegung zu vermitteln müssten im Idealfall alle diese Sinne angesprochen werden. So gibt es Bewegungssimulatoren die es ermöglichen den vestibulären Sinn zu beeinflussen. Für die Illusion einer Eigenbewegung ist meist schon die Stimulation der visuellen Wahrnehmung ausreichend. Als Beispiel ist hier das Betrachten eines anfahrenden Zuges aus einem stehenden aufzuführen [Dör13].

Durch diese Stimuli können dementsprechend VR Anwendungen umgesetzt werden, die ohne eigene Steuerung auskommen. Die Bewegung wird hier durch visuelle Stimuli von Eigenbewegung hervorgerufen.

Durch die selbständige Steuerung eines Nutzers, idealerweise durch Eigenbewegung, ist es möglich einen noch höheren Grad der Immersion zu erreichen. Die dafür benötigten und verfügbaren Eingabegeräte sowie Voraussetzungen werden im Folgenden näher beleuchtet.

2.1.2 Ausgabegeräte

Um überhaupt dem Nutzer das Eintauchen in eine virtuelle Welt zu ermöglichen, sind entsprechende Ausgabegeräte erforderlich. Diese müssen so gebaut sein, dass eine möglichst hohe Immersion erreicht wird und der Nutzer sich präsent in der VR fühlt. Die Ausgabegeräte müssen auf Positionsveränderung oder Rotationsbewegungen des Nutzers reagieren [Dör13].

Allgemein kann eine Klassifikation zwischen kabelgebundenen und kabellosen Ausgabegeräten erfolgen. In diesem Fall werden nur kabellose näher betrachtet. Die visuelle Ausgabe erfolgt über ein Displaysystem welches ein Monitor sein kann oder mehrere zusammengesetzten Monitore. Das hier gewählte Ausgabegerät ist eine Art eines Head-Mounted Displays (HMD). Die genutzten Ausgabegeräte werden in Abschnitt 2.2 näher betrachtet. Grundlegende Bestandteile eines HMD sind in Abbildung 2.1 zu sehen.

Um nun dem Nutzer ein Gefühl der Immersion zu geben, werden erzeugte virtuelle Inhalte durch zwei leicht voneinander versetzte virtuelle Kameras erfasst und am HMD ausgegeben. Dabei ist darauf zu achten das eine korrekte Berechnung der Stereobildpaare stattfindet und entsprechende Einstellungen der virtuellen Kameras gemacht werden [Dör13].

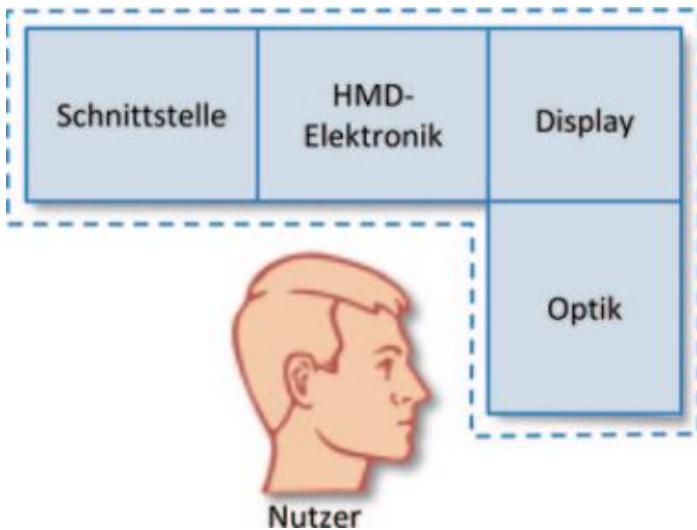


Abb. 2.1 Prinzipielle Bestandteile eines HMDs [Dör13]

2.1.3 Eingabegeräte

Eingabegeräte ermöglichen die Erkennung von Nutzerinteraktionen über Sensoren. Die dadurch gesammelten Daten werden an die VR übertragen und dort entsprechend ausgewertet und als Bewegung in der VR übersetzt. Dabei kann sich die Nutzerinteraktion auf unterschiedliche Arten wiederspiegeln. Der einfachste Fall ist das Betätigen eines Knopfes, was dann ein einmaliges Ereignis auslöst. Komplexere Systeme registrieren Bewegungen der Hand oder sogar des ganzen Körpers um eine Interaktion mit der VR zu ermöglichen. Dieser Vorgang wird als Tracking bezeichnet. Das Tracking beschreibt den Vorgang der kontinuierlichen Verfolgung durch ein Eingabegerät. Dabei werden Position und Orientierung eines Objektes bestimmt [Dör13].

Das Ziel des Tracking ist es, die Werte entsprechend von Freiheitsgraden (engl. Degrees of Freedom, DOF) der verfolgten Körper für die kontinuierliche Interaktion zu bestimmen bzw. zu schätzen. Dadurch wird die Interaktion mit der virtuellen Welt möglich. Die Datenaufnahme erfolgt meist im Bezugssystem des jeweiligen Trackingsystems. Kommen mehrere oder gar unterschiedliche Systeme zum Einsatz, so müssen die Trackingdaten in ein gemeinsames Bezugssystem überführt werden [Dör13].

Die Grundlagen für solche Eingabegeräte lassen sich nach [Dör13] in zehn Kriterien unterteilen, um eine gute Beschreibung von Eingabegeräten zu ermöglichen.

Freiheitsgrad

Der DOF bezeichnet die voneinander unabhängige Bewegungsmöglichkeit eines physikalischen Systems. Dabei kann die Bewegung eines starren Objektes in eine Verschiebung im Raum (Translation) und eine Drehung (Rotation) resultieren. Entsprechend der drei Koordinaten als Position und der drei Winkel zur Beschreibung der Orientierung hat ein starrer Körper 6 DOF. Es ist wünschenswert diese Anzahl der DOF mit einem Eingabegerät zu erreichen, um ein möglichst immersives Gefühl zu vermitteln [Dör13].

Gleichzeitig verfolgte Körper

Hierbei ist es wichtig wie viele Objekte gleichzeitig verfolgt werden sollen. So soll neben der Blickverfolgung auch das Eingabegerät verfolgt und die Daten ausgewertet werden. Hierbei ist eine eindeutige Zuteilung der Objekte nützlich, um den Überblick zu bewahren [Dör13].

Größe der überwachten Fläche

Hier ist eine sehr große Differenz der Fläche möglich je nachdem welches Eingabegerät genutzt wird. Es muss den Eingabegeräten ein entsprechend großer Bereich zur Verfügung gestellt sein, um den kompletten Funktionsumfang nutzen zu können [Dör13].

Als Beispiel unterscheidet sich die Größe der überwachten Fläche bei der Nutzung einer Kamera als Eingabegerät deutlich von der eines Spielecontrollers (Controller), bei dem keine Fläche überwacht werden muss.

Genauigkeit

Die Genauigkeit richtet sich oft nach der Frage des Kostenaufwands. Bessere Kameras liefern bessere Bilder, bessere Controller können eine genauere Steuerung ermöglichen [Dör13].

Wiederholrate

Die Wiederholrate beschreibt das Auflösungsvermögen eines Eingabegeräts anhand der Zeit. Sie beschreibt die Anzahl der Messpunkte einer Bewegung pro Sekunde. Je höher die Wiederholrate, desto mehr Messpunkte sind vorhanden [Dör13].

Latenz

Die Latenz ist die Zeitspanne die ein Eingabegerät zum Reagieren braucht. Diese verschobene Reaktion kann durch das Abarbeiten von Algorithmen oder durch Laufzeiten von Signalen in Kabeln ausgelöst werden [Dör13].

Drift

Eine Drift, auch Messfehler, kann durch sich immer weiter aufaddierende Fehler entstehen. Wenn Eingabegeräte relative Änderungen aufnehmen (z. B. Positionsänderung gegenüber der vorherigen Abtastung bzw. dem vorherigen Messpunkt), dann können Fehler sich über die Zeit aufaddieren, woraus ein fortwährender und anwachsender Fehler folgt [Dör13].

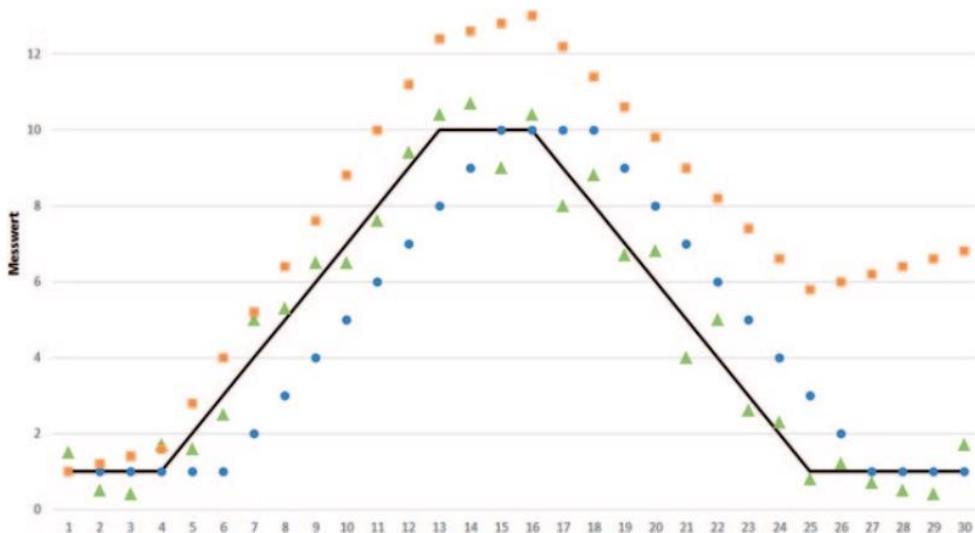


Abb. 2.2 Mögliche Fehler bei der Datenaufnahme der Position eines bewegten Objektes (schwarze Linie): Aufnahme mit Latenz (blau), mit Drift (Orange), mit Rauschen (grün) dargestellt über die Zeit (horizontale Achse) [Dör13]

Äußere Rahmenbedingungen

Äußere Rahmenbedingungen wie Licht, Temperatur oder die Möblierung eines Raumes, können je nach Eingabegerät Einfluss haben auf die Funktionalität. Optische Verfahren können bei gleichmäßiger Beleuchtung besser arbeiten. Bei Verfahren die den Schall nutzen spielen unterschiedliche Temperaturen oder Luftdrücke eine Rolle. Elektromagnetische Verfahren werden von magnetischen Stoffen oder elektromagnetischen Feldern gestört [Dör13].

Kalibrierung

Die Kalibrierung behandelt den Abgleich von Messwerten. Hier werden Einstellungen vorgenommen, um verfolgten realen Bewegungen den Maßen der virtuellen Welt anzupassen und umzuwandeln. Dadurch fühlt sich die Steuerung intuitiv an [Dör13].

Usability

Usability kann vor allem durch die Freiheiten, die das Eingabegerät den Nutzer gibt, beschrieben werden. So sind zum einen Einschränkungen durch das Tragen von bestimmten Sensoren oder das Halten eines Controllers gegeben. Zudem sind über Funk verbundene Eingabegeräte benutzerfreundlicher als über Kabel verbundene. Bei optischen Verfahren, die den Nutzer über Kameras verfolgen, ist der Interaktionsradius ausschlaggebend für das Maß der Usability [Dör13].

2.1.4 Bewegungskontrolle mit Eingabegeräten

Allgemein können Bewegungen durch Eingabegeräte gesteuert werden. Ob Touchscreen, Tastatur, Controller oder anderes Eingabegerät, es werden Eingabedaten geliefert, welche verarbeitet und übertragen und in Bewegung in der virtuellen Welt umgewandelt werden. Im Folgenden werden mögliche Arten von Eingabegeräten und Verfahren zur Kontrolle von Bewegung gezeigt.

Durch Software

Bewegungskontrolle kann ohne externes Eingabegerät durch Software erzeugt werden. Dabei dienen bestimmte Punkte in der VR als Bewegungsauslöser. Diese lösen nach einer bestimmten Zeit, in der sie angeschaut werden müssen, eine Bewegung aus und bewegen die virtuelle Kamera in eine bestimmte Richtung oder lassen diese rotieren.

Optisches Tracking

Optische Trackingverfahren verfolgen die Idee eine Positionierung und Orientierung von Objekten im Raum festzustellen. Diese Feststellung findet anhand von Tiefen- und Farbkameras (RGBD-Kamera) statt. Diese können über ein mit Infrarot projiziertes Muster oder über die Berechnung der Laufzeiten des reflektierten Lichts (Time of Flight, TOF) eine Tiefenerkennung durchführen und Bewegungen erkennen.

Aktuelle Hardware, die diese Technik anwendet, ist die Kinect von Microsoft [Kra12], die es einem Nutzer erlaubt, ohne einen Controller Bewegungen in eine virtuelle Welt zu übertragen [Dör13] sowie die Leap Motion, die im Gegensatz zur Kinect nicht den ganzen Körper aufnimmt, sondern nur die Hände des Nutzers [@Leap]. Eine noch in der Entwicklung befindliche Hardware ist Project Tango von Google, was Ähnlichkeiten zu einem Android-Smartphone besitzt und es ermöglicht 3D Bewegungen, was auch die Translation mit einbezieht, innerhalb eines Raumes zu verfolgen [@Tan] [Kam12].



Abb. 2.3 Ansicht von der aktuellen Kinect [Com] und Leap Motion [Com]

Bewegungsplattformen

Bewegungsplattformen ermöglichen dem Nutzer die Fortbewegung in einer VR. Durch Laufen oder laufähnliche Bewegungen wird diese in die VR übertragen. Als Eingabegerät dienen hier Laufbänder, die eine omnidirektionale Bewegung, d.h. eine Bewegung in alle Richtungen ermöglichen. Solche omnidirektionalen Laufbänder (omnidirectional treadmill, ODT) können aus mehreren kleinen Laufbändern bestehen, die orthogonal zur Hauptrichtung angeordnet sind. Über Tracking wird die Geschwindigkeit der Laufbänder so gesteuert, dass sich der Nutzer immer in der Mitte der ODT befindet. Eine günstigere Alternative zum Tracking ist das Fixieren des Nutzers durch eine Halterung, wie in Abb. 2.4 [Dör13].

Eine weitere Möglichkeit ist die Nutzung von entsprechend gelagerten Kugeln, in denen sich ein Nutzer bewegen kann und auf der Stelle bleibt. Hierbei wird das Laufen erschwert, da dem Nutzer kein ebener Boden zur Verfügung steht. Als Beispiel ist hier die Cybersphere zu nennen [FRE03].

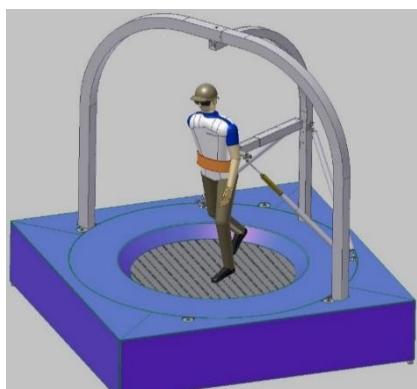


Abb. 2.4 ODT mit Halterung für den Nutzer [Com]

Akustisches Tracking

Das akustische Tracking nutzt Unterschiede in der Laufzeit oder Phase von Schallwellen. Dabei wird Ultraschall verwendet und mithilfe eines Senders und Empfängers wird eine Abstandsbestimmung durchgeführt. Der Sender wird am zu überwachenden Objekt angebracht. Durch die so gesammelten Daten können Bewegungen in die VR übertragen werden. Je mehr Sender und Empfänger genutzt werden, desto höher sind die DOF die erreicht werden können [Dör13].

„Durch Hinzufügen eines zweiten Senders oder eines zweiten Empfängers kann die Position bereits auf eine Kreisbahn (als Schnittpunkt von zwei Kugeln) eingegrenzt werden. Die Erweiterung eines dritten Senders oder Empfängers schränkt dann die Position auf zwei Punkte ein (als Schnittpunkte von drei Kugeln bzw. als Schnittpunkte von zwei Kreisen). Mittels Plausibilitätsüberprüfung wird aus diesen beiden die Position bestimmt.“ [Dör13]

So können mit einem Sender und drei Empfängern die drei DOF der Translation erreicht werden und mit drei Sendern sogar sechs DOF.

Der Vorteil liegt hier in den günstigen Anschaffungskosten. Diese stehen aber einem großen Nachteil gegenüber. Dieser Nachteil besteht in der Empfindlichkeit des Systems in Bezug auf Temperatur- oder Luftdruckänderungen. Eine solche Änderung hat jedes Mal eine Neukalibrierung des Systems zur Folge.

Inertial Tracking

Das Inertial Tracking nutzt Sensoren, die die Beschleunigung messen. Diese werden auch Trägheits- oder Beschleunigungssensoren genannt. Diese können in lineare Initalsensoren und Beschleunigungssensoren unterteilt werden. Erstere messen die Beschleunigung anhand einer Achse und letztere erfassen die Winkelbeschleunigung um eine Achse und werden auch Gyrosensoren genannt aufgrund des ähnlichem Verhaltens zu einem Gyroskop. Die linearen Beschleunigungssensoren werden im Ruhezustand zur Lagebestimmung genutzt [Dör13].

„Dann kann auf Basis der Erdbeschleunigung die Neigung zu deren Richtung (d. h. der Senkrechten) gemessen werden. Da die Ausrichtung in der Horizontalen (y-Achse) senkrecht zur Gravitation liegt, kann diese mit linearen Initalsensoren nicht erfasst werden. Es werden somit maximal Rotationen um die beiden in der horizontalen Ebene liegenden Achsen (x-Achse und z-Achse) erfasst.“ [Dör13]

Neben der Lageerfassung kann auch eine Positionsbestimmung stattfinden. Diese ist aufgrund der geringen Genauigkeit bei der Umwandlung von analogen Messwerten zu digitalen Werten aber nicht optimal und kann zu Drifteffekten führen. Dasselbe gilt für die Messung der Winkelgeschwindigkeit um einen Rotationswinkel zu erhalten [Dör13].

Als Beispiel ist hier Sphero [Sph] und 3DRudder [Rud] aufzuführen. Sphero ist eine Kugel mit entsprechenden Sensoren, die mit einer Hand bedient werden kann und 3DRudder wird mit den Füßen bedient. Auf diese beiden Controller wird im weiteren Verlauf der Arbeit noch genauer eingegangen.



Abb. 2.5 Sphero[Orb] und 3DRudder [Rud]

Magnet

Eine weitere Möglichkeit der Steuerung in einer VR ist über einen Magneten. So können bestimmte mobile Geräte Dockingstationen oder Abdeckungen anhand der eingebauten Magnetsensoren erkennen. Durch die Veränderung der Position eines Magneten im Radius dieser Sensoren kann ein vorher implementierter Vorgang ausgelöst werden, wie z.B. eine Vorwärtsbewegung. Somit kann die Veränderung des magnetischen Feldes um ein mobiles Gerät registriert werden [Car].

Eine weitere Möglichkeit ist, dass Position und Orientierung eines Controllers durch ein Elektromagnetisches Feld bestimmt wird. In diesem Fall besteht keine Gefahr eines Drifts, da hier keine Lage- oder Beschleunigungssensoren genutzt werden [Six].

Gamepad

Das Gamepad ist einer der gängigsten genutzten Controller. Dieses wird oft zur Steuerung von Videospielen an Konsolen und Computer verwendet. Hier ermöglicht die Eingabe mit Daumen und Zeigefingern die Bewegungssteuerung und Interaktion in einer virtuellen Welt. Dabei ist

es möglich über verschiedene Knöpfe sowie Miniaturjoysticks eine Eingabe zu tätigen, welche an die Konsole oder den Computer übertragen wird. Heutige Gamepads ermöglichen eine kabelfreie Verbindung über Bluetooth und besitzen meist mehrere Knöpfe oder bis hin zu Touchscreens [Fch10].

Als Beispiel sind hier die Controller aktueller Konsolen wie die Playstation 4 und Xbox One zu nennen.



Abb. 2.6 Playstation 3 Controller [Bus]

2.2 Virtual Reality für mobile Geräte

Virtual Reality lässt sich relativ einfach über ein HMD erzeugen. Aktuelle Geräte wie die Oculus Rift haben aber hohe Anschaffungskosten und sind per Kabel an einen PC angeschlossen. Dieser PC ist auch unabdingbar, da hier die Spiele oder virtuelle Welten erzeugt werden [@Ocu].

Aufgrund dieser Einschränkungen wurden kostengünstige Instrumente entwickelt, die aktuelle Eigenschaften von Smartphones nutzen und somit eine Verbindung zu einem PC überflüssig machen, da die virtuellen Inhalte vom Smartphone selbst berechnet werden. Diese Instrumente sind Behälter in die das Smartphone hineingelegt werden kann. Grundlegend besitzen alle Behälter zwei Linsen, um das Sichtfeld des Nutzers auf den Handybildschirm zu reduzieren und teilen das Sichtfeld durch eine Trennwand innerhalb des Behälters. Dadurch kann für jedes Auge ein separates Bild erzeugt werden und dem Nutzer ein immersives Gefühl vermittelt werden. Im folgenden Abschnitt werden aktuelle VR-erzeugenden Instrumente für mobile Geräte vorgestellt.

2.2.1 Boxx3D

Die Boxx3D von der Firma Die Etagen ist eine Halterung für Smartphones, um eine immersive VR zu erzeugen. Sie besitzt die Grundlegenden Eigenschaften und unterstützt unterschiedliche Smartphone Größen [Bus14].



Abb. 2.7 Verschiedene Ansichten der Boxx3D [Eta]

2.2.2 Google Cardboard

Cardboard von Google entwickelt ist ähnlich zu der Boxx3D. Auch hier kann ein Smartphone hineingelegt werden und es wird ein immersiver Eindruck vermittelt. Neben den grundlegenden Eigenschaften besitzt Cardboard eine zusätzliche Funktionalität – einen an der Seite befindlichen Schalter. Dieser besteht aus zwei Magneten. Einer ist an der Außenseite und ein weiterer an der Innenseite des Gehäuses befestigt.

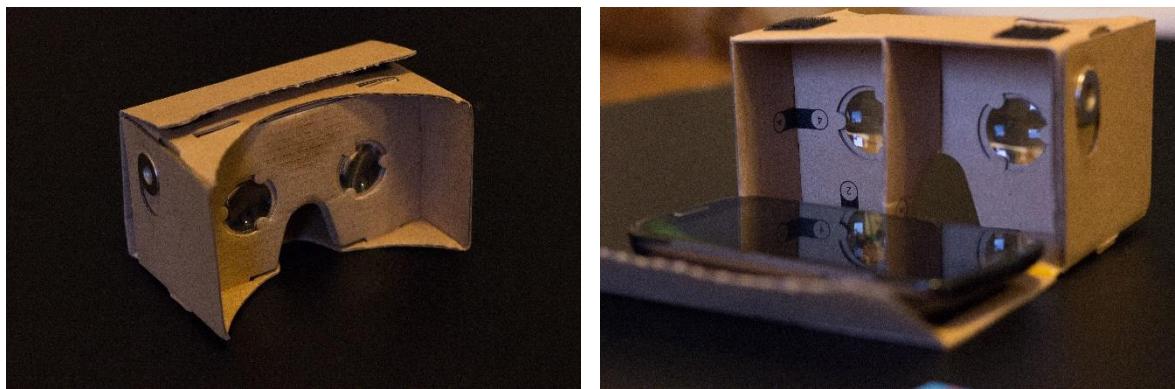


Abb. 2.8 Verschiedene Ansichten der Cardbox [Bus]

Durch das Herunterziehen des Schalters und das darauf folgende Zurückschnellen des äußeren Magneten, wird das Magnetfeld beeinflusst. Dieses wiederum kann von bestimmten Smartphones registriert werden und dann bestimmte Aktionen auslösen [@Car].

2.2.3 Gear VR

Die Gear VR ist durch eine Kollaboration von Samsung Electronics und Oculus VR entstanden. Im Gegensatz zu der Boxx3D und Cardboard besitzt diese eine Kopfbefestigung.



Abb. 2.9 Ansicht der Gear VR [Com]

Die Kosten sind gegenüber den anderen beiden vorgestellten Instrumenten deutlich höher, da das Gehäuse der Gear VR hochwertiger ist und Anschlüsse innerhalb des Gehäuses verbaut sind. Außerdem kann die Gear VR nur mit einem bestimmten Smartphone benutzt werden, was wieder Kosten verursachen kann [@Gear].

2.3 Unity

Unity wird als Entwicklungsumgebung genutzt, um die prototypische Anwendung umzusetzen. Unity ist besonders für Anwendungen im Bereich VR geeignet, da Unity das Programmieren auf Objekte ermöglicht und die Platzierung von virtuellen Kameras trivial gestaltet. Des Weiteren können die Anwendungen auf verschiedene Plattformen veröffentlicht werden, ohne unterschiedliche Implementierungen vorzunehmen [Sei14]. Dieser Abschnitt gibt einen Überblick über die Funktionen und Werkzeuge die genutzt werden und stellt verwendete Plugins vor.

Es wird mit Version 4.6 gearbeitet, die zum Vorgänger Änderungen am GUI-System hat und allgemein eine höhere Leistung aufweist [@Uni].

2.3.1 Programmierung über Skripte

Unity hat für das Programmieren von Skripten eine eigene Entwicklungsumgebung. Diese ist MonoDevelop, welche Open-Source ist und unter dem Aspekt entstand ein leichteres Debuggen von Unity Projekten zu ermöglichen. Verfügbare Programmiersprachen sind C#, JavaScript und Boo. In dieser Arbeit wird mit C# gearbeitet. Skripte sind als eigene Klassen aufzufassen, die als Komponente an Objekte angehängt und dadurch erst genutzt werden können. Jedes Skript muss von der Klasse MonoBehaviour erben, um die Funktionalität eines Skripts als Komponente zu ermöglichen. Skripte lassen sich über das Hauptmenü erstellen oder können direkt als Komponenten eines Objektes zugewiesen werden [Sei14].

Ein neu erstelltes C#-Skript enthält über using eingebundene Namespaces, um bestimmte Grundlegende Funktionalitäten zur Verfügung zu stellen. Nach den Namespaces kommen die Klassendefinition und die MonoBehaviour-Erbung. Diese Erbung erhält jedes Skript Standardmäßig da hier auch die Start- und Update-Methoden bereitgestellt werden [Sei14].

Event-Methoden

Die Start-Methode wird einmalig ausgeführt, und dient der Initialisierung von Variablen. Sobald eine Skript-Instanz aktiviert ist, wird als erstes die Start-Methode ausgeführt. Dadurch ist es möglich die Reihenfolge von Initialisierungen zu bestimmen [Sei14].

Die Update-Methode zählt zu den wichtigsten Methoden in Unity. Update wird jedes Mal aufgerufen bevor ein Frame gerendert wird. Hier ist es z.B. möglich einem Objekt Bewegungen zuzufügen durch das Verändern der Positionsparameter [Sei14].

Neben diesen beiden Methoden gibt es noch weitere Grundlegende Methoden die je nach Notwendigkeit genutzt, aber hier nicht näher beleuchtet werden.

```
using UnityEngine;
using System.Collections;

public class NewScript : MonoBehaviour {

    // Use this for initialization
    void Start () {
    }

    // Update is called once per frame
    void Update () {
    }
}
```

Listing 2.1 Grundgerüst eines C#-Skripts

Komponenten

Der nächste wichtige Aspekt ist die Komponentenprogrammierung. Diese erlaubt es ein Skript zu erstellen, welches an beliebig viele Objekte angehängt werden kann.

„Ein wichtiger Grundsatz von Unity ist, dass jedes GameObject seine eigenen Components besitzt. Sie programmieren also keine Lebensverwaltung, die die Lebensstärke aller Gegner verwaltet, sondern ein Skript, das lediglich die Gesundheit eines einzelnen Gegners verwaltet. Dieses Skript wird dann aber wieder jedem Gegner zugefügt, sodass jeder Gegner seine eigene Verwaltung hat.“ [Sei14]

So ist es möglich über die Variable gameObject auf Komponenten des eigenen Objektes zuzugreifen, um Parameter zu verändern.

```
Transform transform = gameObject.GetComponent<Transform>();
```

Listing 2.2 Zugriff auf eigene Komponente eines Objektes

```
public class ComponentExample : MonoBehaviour {  
    public GameObject user;  
}
```

Listing 2.3 Inspector Zuweisung einer public Variablen

Der Zugriff auf andere Objekte erfolgt über den Inspector. In einem Skript werden public-Variablen gesetzt, die im Inspector erscheinen und in die per Drag&Drop Objekte gezogen werden können [Sei14].



Abb. 2.10 Inspectoransicht nach Zuweisung einer public-Variablen

Diese Form um auf andere Objekte zuzugreifen ist die performanteste und wird bei der Erstellung der prototypischen App verwendet. Hierbei findet der Zugriff zur Entwicklungszeit statt. Auch zur Laufzeit ist es möglich auf Objekte zuzugreifen. Diese Methode kostet aber mehr Ressourcen, da Unity erst das passende Objekt finden muss und bei einer großen Menge von Objekten alle durchgelaufen werden, bis das entsprechende gefunden wurde [Sei14].

Coroutine und Invoke

Ein weiteres wichtiges Element ist das parallele Ausführen von Code. Über sogenannte Coroutines können Codeblöcke parallel zu Update-Aufrufen ausgeführt werden und sich über beliebig viele Frames erstrecken [Sei14].

```
IEnumerator Message () {  
    yield return new WaitForSeconds (1);  
    message = "1";  
}
```

Listing 2.4 Aufbau einer Coroutine

Das gezeigte Beispiel in Code 2.4 wartet eine Sekunde, um dann dem String message einen neuen Wert zuzuweisen. Die Methode WaitForSeconds sorgt dafür, dass die Ausführung der Coroutine, um die angegebene Zeit unterbrochen wird [Sei14].

Um eine Coroutine zu implementieren wird die Schnittstellendefinition IEnumerator sowie der Befehl yield return benötigt. Ersteres ermöglicht das Durchführen der Methode über mehrere Frames und letzteres speichert den aktuellen Stand der Routine. Das ist notwendig, um die Methode beim nächsten Frame an gleicher Stelle fortsetzen zu können. Um eine Coroutine dann ausführen zu können wird der Befehl StartCoroutine genutzt [Sei14].

```
void Start() {
    StartCoroutine(Message());
}
```

Listing 2.5 Beispiel Coroutine

Des Weiteren gibt es verzögerte Funktionsaufrufe, die über den Invoke-Befehl erreicht werden. Hier werden eine Funktion und ein Sekundenwert angegeben. Das führt dazu, dass die angegebene Methode mit bestimmter Verzögerung nach der Start-Methode ausgeführt wird. Dabei ist darauf zu achten, dass sich die Methode im selben Skript befindet [Sei14].

```
Invoke ("ExampleFunction", 2.5f);
```

Listing 2.6 Beispiel für Invoke

Laden und Speichern

Eine weitere wichtige Grundlage ist das Laden und Speichern von Daten, um Werte unabhängig vom System zu speichern. Dazu stellt Unity die PlayerPrefs-Methoden zur Verfügung. Diese erhalten zu speichernde Werte und Unity kümmert sich um die Verarbeitung und das Speichern dieser Daten. So können bei unerwartetem Beenden der App Spielstände wiederhergestellt werden und der Nutzer muss bei einem Spiel nicht immer wieder von vorne anfangen. Dafür stellt PlayerPrefs drei Methoden zur Verfügung. Hier können Werte als int, float oder string gespeichert werden [Sei14].

```
PlayerPrefs.SetInt("Points", 2);  
PlayerPrefs.SetFloat("Power", 1.2f);  
PlayerPrefs.SetString("Name", "Max");
```

Listing 2.7 Speichern von Daten

Der erste Parameter ist die ID des Wertes, der zweite der eigentliche Wert. Wie beim Speichern gibt es auch beim Laden von Werten die Möglichkeit int, float oder string-Werte zu laden. Dafür wird die zuvor gespeicherte ID übergeben [Sei14].

```
int points = PlayerPrefs.GetInt("Points");  
float power = PlayerPrefs.GetFloat("Power");  
string name = PlayerPrefs.GetString("Name");
```

Listing 2.8 Laden von Daten

Debug

Um Fehler und Warnungen zu überprüfen besitzt Unity die Console. Diese werden dort angezeigt und geben Hinweise um aufzutretende Fehler zu beheben. Über die Klasse Debug besteht die Möglichkeit eigene Meldungen auszugeben, um z.B. bestimmte Ausgaben von Funktionen zu überprüfen. Die Varianten LogError, LogException und LogWarning geben die entsprechende Meldung aus und werden zusätzlich noch gekennzeichnet [Sei14].

```
Debug.Log("Success");  
Debug.LogError("Error");  
Debug.LogException("Exception");  
Debug.LogWarning("Warning");
```

Listing 2.9 Debug-Möglichkeiten

2.3.2 Objekte im 3D-Raum

Virtuelle Objekte können in Unity in einer Szene platziert werden. Es ist möglich platzierte Objekte zu skalieren, rotieren oder zu verschieben, um so eine virtuelle Welt für den Nutzer zu erzeugen.

Für die Darstellung von Objekten wird ein linkshändiges Koordinatensystem genutzt. Die Darstellung erfolgt anhand von Punkten, die im dreidimensionalen Raum durch Vektoren beschrieben sind. Unity hat dafür den Typ Vector3, über den die Parameter x, y, z eines Vektors verändert werden können. Neben der Beschreibung von Punkten werden Vektoren auch für die Richtungsbeschreibung und Abfragen von Kollisionen eingesetzt [Sei14].

```
Vector3 v1 = new Vector3(1, 1, 1);
Vector3 v2 = new Vector3();
v2.x = 1;
v2.y = 1;
v2.z = 1;
```

Listing 2.10 Beispiele für die Erzeugung eines Vektors

Viele einzelne Vektoren zusammen ergeben dann ein 3D-Modell. Diese einzelnen Vektoren heißen auch Vertices und drei beieinander liegende Vertices, die eine dreieckige Fläche ergeben, nennt man Polygone. Komplexere 3D-Modelle bestehen also aus vielen Polygonen. Diese Struktur wird oft als Polygonnetz oder Mesh bezeichnet. In Unity wird der Begriff Mesh genutzt. Umso mehr Polygone ein Objekt besitzt desto höher sind die Anforderungen an die Performance. Daher sollte immer darauf geachtet werden möglichst wenige Polygone pro Objekt zu haben, insbesondere wenn Anwendungen für mobile Geräte (Apps) umgesetzt werden [Sei14] [Bla11].

Jedes Objekt in einer Szene in Unity wird als GameObject bezeichnet. Ein GameObject kann mithilfe von Komponenten ein Mesh darstellen. Die erste Komponente ist der MeshFilter, damit das GameObject das Mesh aufnehmen kann. Für das Anzeigen des eigentlichen 3D-Objektes ist im nächsten Schritt der MeshRenderer zuständig. Die Erzeugung der Komponenten findet automatisch statt, wenn ein 3D-Modell aus dem Project Browser in die Szene hineingezogen

wird. Es besteht die Möglichkeit alle Komponenten per Code zu verändern und anzupassen [Sei14].

Eine weitere Komponente, die jedes GameObject erhält, ist die Transform-Komponente. Diese sorgt dafür, dass ein Objekt skaliert, rotiert und verschoben werden kann. Die zugehörigen Parameter können direkt in der Szene über die Transform-Tools geändert oder im Inspector in die entsprechenden Felder eingetragen werden. Der Inspector gibt eine Übersicht der Komponenten eines einzelnen Objektes und der Nutzer hat die Möglichkeit Werte anzupassen. Diese Anpassung kann auch mit Hilfe von Code erfolgen durch den Zugriff auf die Transform-Klasse. Diese stellt Eigenschaften und Methoden bereit um das Objekt zu transformieren [Sei14].

Da ein Mesh ein aus Polygonen bestehendes Netz ist, bietet Unity die Möglichkeit durch Materials die Oberfläche eines solchen Drahtgitters zu verändern. So können Eigenschaften wie die Textur, die Farbe und die Reaktion auf äußere Einflüsse wie Licht festgelegt werden. Die Berechnung des Aussehens findet mit Hilfe von Shadern statt. Shader bestimmen wie Materialien auf Oberflächen gerendert werden sollen. Hier muss bei Apps wieder darauf geachtet werden spezielle mobile Shader zu verwenden, um eine hohe Performance zu erreichen [Sei14].

Das Hinzufügen von 3D-Modellen erfolgt, wie erwähnt, über den Project Browser. Hier stellt Unity schon vorgefertigte Grundobjekte wie Kugeln oder Würfel zur Verfügung. Diese werden oft als Hilfsmittel genutzt. Komplexere 3D-Modelle werden in einem dafür entsprechenden Programm erstellt und in Unity importiert. Die importierten Modelle werden als Prefab angelegt und erhalten, sobald sie in die Szene gezogen werden, die notwendigen Komponenten [Sei14]. Auf Prefabs wird im Abschnitt 2.3.7 näher eingegangen.

2.3.3 Kamera

Die Kameras geben dem Nutzer die Möglichkeit das Spiel zu sehen und geben je nach Position der Kamera und Sichtweite ein ganz individuelles Nutzererlebnis. Insbesondere für das Erstellen der VR ist die Kamera sehr wichtig. Hier werden Parameter festgelegt, um dem Nutzer ein möglichst hohes immersives Gefühl zu vermitteln.

Damit ein GameObject zur Kamera wird benötigt es das CameraComponent. Diese Komponente ermöglicht es der Kamera zu sehen und besitzt verschiedene Parameter. Die für das Erzeugen einer Kamera benötigten Parameter werden im Folgenden kurz beschrieben [Sei14]:

- › **Clear Flags** sorgt dafür wie der Hintergrund aussieht. Hintergrund bezeichnet alles wo keine Objekte gerendert werden. Das kann eine einfache Farbe oder ein Himmel mithilfe einer Skybox sein. Eine Skybox ist ein aus sechs Bildern bestehender Würfel der sich um die gesamte Szene legt und so den Eindruck eines umgebenden Himmels vermittelt.
- › **Background** lässt den Nutzer die Hintergrundfarbe auswählen falls dieses bei Clear Flags angegeben wurde.
- › **Projection** bestimmt die Ansicht der Kamera. Hier kann zwischen einer dreidimensionalen, perspektivischen oder einer orthogonalen Ansicht gewählt werden.
- › **Field of View** legt die Größe des Sichtbereiches fest. Hier muss durch Testen ein Wert ermittelt werden der für VR optimal ist.
- › **Clipping Planes** legt den Abstand von gerenderten Objekten fest. Das heißt Objekte die zu nah dran sind oder zu weit weg werden nicht gerendert.
- › **Normalized View Port Rect** passt den Raum den das Kamerabild auf dem Bildschirm des Nutzers einnimmt an. Bei VR müssen zwei Kameras platziert sein und an dieser Stelle die Breite der beiden Kamerabilder auf die Hälfte reduziert werden, damit zwei nebeneinander liegende Kamerabilder entstehen.

Da Kameras auch GameObjects sind, lassen sich auch hier Skripte an diese anhängen. Insbesondere die Steuerung der Kameras muss in der VR synchron erfolgen. Diese Funktionalität wird über mehrere Plugins gelöst, die im Abschnitt 2.3.7 vorgestellt werden.

2.3.4 Physik

Die Physik-Engine in Unity wird in dieser Arbeit nur genutzt, um Schwerkraft auf den Nutzer in der virtuellen Welt auszuüben und um ihn mit anderen Objekten interagieren zu lassen. Wichtigste Komponente ist dabei die Rigidbody-Komponente. Mit deren Hilfe kann einem GameObject eine Kraft zugefügt werden, wie z.B. die Erdanziehungskraft [Sei14].

Eine weitere Komponente sind sogenannte Collider. Collider registrieren Kollisionen und können dadurch bestimmte Events auslösen. In dieser Arbeit wird der Mesh-Collider verwendet, der sich an die Form des jeweiligen Meshs anpasst[Sei14].

Eine Kombination aus Rigidbody und Collider stellt der Character Controller dar. Dieser fügt einem Objekt einen kapselförmigen Collider zu und ermöglicht es diesen zu bewegen und zu steuern. In Verbindung mit einer Kamera kann der Nutzer aus der Egoperspektive (first-person perspective, FPP) die VR erkunden [Sei14].

2.3.5 Inputmöglichkeiten

Dieser Abschnitt beschreibt die Möglichkeiten, die Unity bietet um Eingabegeräte anzuschließen und zu konfigurieren. Eingabegeräte sind ein wichtiger Bestandteil von VR, da sie es dem Nutzer ermöglichen mit der VR zu interagieren und sich in dieser zu bewegen. Eingabegeräte sind neben den schon beschriebenen, auch Tastatur und Maus sowie Touch-Eingaben. Diese werden hier nicht näher beleuchtet, da auf der einen Seite Touch-Eingaben nicht möglich sind, da sich das mobile Gerät in einer Box befindet und Tastatur und Maus das immersive Gefühl vermindern. Auch das Auswerten von Beschleunigungssensoren oder Kompass eines mobilen Gerätes ist möglich [Sei14].

Virtuelle Achsen und Tasten

Unity bietet die Möglichkeit virtuelle Achsen und Tasten zu belegen. Das führt dazu, dass zwischen verschiedenen Eingabegeräten gewechselt werden kann ohne die Belegung dieser zu verändern. Dieses modulare Mapping von Achsen und Tasten erfolgt über den sogenannten Input Manager [Sei14].

Input-Manager

Der Input-Manager bietet die Möglichkeit Achsen und Tasten in einem Unity-Projekt zu ändern, erweitern oder zu löschen. Standardmäßig sind bei der Erstellung eines neuen Unity-Projekts die wichtigsten Eingaben schon belegt. Diese belaufen sich auf die Navigation, das Schießen und Springen [Sei14].

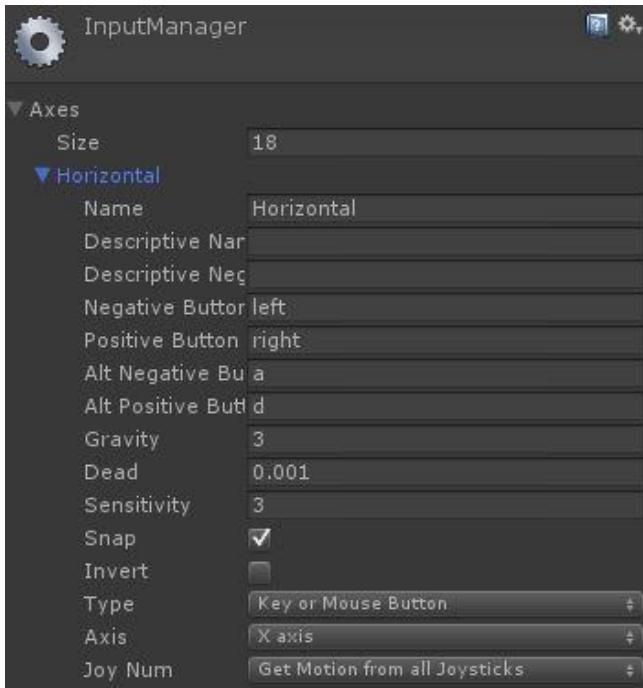


Abb. 2.11 Ansicht des Input-Managers

Die Bedeutung und Einstellungsmöglichkeiten der einzelnen Parameter werden im Folgenden kurz beschrieben und als Beispiel wird auf die horizontale Steuerung in Abb. 2.9 mit Maus und Tastatur eingegangen. Diese Steuerung beschreibt Bewegungen nach rechts und links [Sei14].

- › **Name** ist die Bezeichnung der Abfrage eines Wertes. Hier sollte darauf geachtet werden die Funktionalität möglichst gut zu beschreiben, um eine gute Code-Lesbarkeit zu erreichen.
- › **Descriptive Name** und **Descriptive Negative Name** enthalten den Beschreibungstext für die positive sowie negative Achsenbeschreibung.
- › **Negative Button** und **Positive Button** sind die positiven sowie negativen Tasten für den Achsenwert. In diesem Fall für die Bewegung nach links *left* und für die Bewegung nach rechts *right*. Diese stehen für die Pfeiltasten auf der Tastatur.
- › **Alt Negative Button** und **Alt Positive Button** sind alternative Belegungen für den positiven sowie negativen Achsenwert.
- › **Gravity** beschreibt die Zeit in der der Wert auf 0 zurück fällt nach einem Tastenanschlag
- › **Dead** enthält den Wert ab dem Achsenwerte auf 0 gesetzt werden.
- › **Sensitivity** beschreibt die Zeit in der der Wert auf den Zielwert ansteigt.
- › **Snap** setzt den Achsenwert auf 0 zurück, wenn die entgegengesetzte Richtung gedrückt wird.
- › **Invert** invertiert alle Werte.

- > **Type** bestimmt den Eingabetyp, also Taste, Maustaste oder Joystick Achse
- > **Axis** gibt die Möglichkeit bei angeschlossenem externem Eingabegerät die entsprechende Achse auszuwählen.
- > **Joy Num** ist wichtig wenn mehrere externe Eingabegeräte angeschlossen sind. Dann muss an dieser Stelle das entsprechende Eingabegerät ausgewählt werden.

Der Unterschied zwischen virtuellen Achsen und virtuellen Tasten besteht darin, dass Unity diese verschieden auswertet. So werden Achsen über zwei Tasten gesteuert und dabei kann die eine Taste einen Wertebereich von 0 bis -1 annehmen und die anderen einen zwischen 0 und +1. Virtuelle Tasten dagegen enthalten nur eine Taste, die einen Wertebereich zwischen 0 und +1 annimmt [Sei14].

Da es bei Eingabegeräten sehr viele Unterschiede gibt, bietet Unity mit den Joystick-Inputs eine Möglichkeit um externe Eingabegeräte anschließen zu können. Diese müssen aufgrund der Unterschiedlichkeit oft einmalig justiert werden.

Folgende Parameter sind dabei zu beachten [Sei14]:

- > **Type** muss auf Joystick Axis gestellt sein.
- > **Axis** muss auf die entsprechende Achse gestellt sein.
- > **Joy Num** muss auf den entsprechenden Controller gestellt sein.

Neue Inputs lassen sich über den Input-Manager anlegen, indem die Zahl bei Size erhöht wird [Sei14].

Auswertung von Eingaben

Die Auswertung von Eingaben lässt sich über die Klasse Input realisieren. Diese enthält Methoden mit denen die Abfrage von Eingaben ermöglicht wird. Hierbei können neben den virtuellen Achsen und Tasten auch Tastatur- und Maustasten, Toucheingaben und Beschleunigungssensoren abgefragt werden [Sei14].

Im Folgenden werden entsprechende Methoden kurz vorgestellt und anhand von Beispielen erklärt.

- > **GetAxis** gibt den Wert einer Achse zurück

```
float horizontalSpeed = Input.GetAxis("Horizontal");
```

Listing 2.11 Zugriff auf Eingaben durch GetAxis

- > **GetButton** gibt True oder False zurück, wenn eine Taste gedrückt wird. Dabei wird die ganze Zeit True zurück gegeben, während entsprechende Taste gedrückt wird
- > **GetButtonDown** gibt True in dem Frame zurück in dem die Tasten gedrückt sind.
- > **GetButtonUp** gibt True zurück, wenn die Taste losgelassen wird.

```
public bool autoFire = false;
void Update() {
    autoFire = false;
    if (Input.GetButtonDown("Fire1"))
        audio.Play();
    if (Input.GetButton("Fire1"))
        autoFire = true;
    if (Input.GetButtonUp("Fire1"))
        audio.Stop();
}
```

Listing 2.12 Zugriff auf Eingaben mit GetButton [Sei14]

- > **GetKey** funktioniert wie die GetButton-Methode, benötigt aber einen KeyCode oder den entsprechenden Buchstaben auf der Tastatur. Auch hier gibt es wieder eine GetKeyDown sowie GetKeyUp-Methode, die wie die entsprechenden Gegenstücke bei GetButton funktionieren.

```
...
    If (Input.GetKeyDown(„a“))
        autoFire = true;
...

```

Listing 2.13 Zugriff auf Eingaben mit GetKey

Auswertung des Beschleunigungssensors

Beschleunigungssensoren sind in fast jedem Smartphone oder Tablet zu finden. Diese können für die Steuerung und Orientierung genutzt werden. Unity bietet mit der acceleration-Variable Zugriff auf die Werte des Beschleunigungssensors in Form eines Vector3-Objektes. Der Zugriff erfolgt dabei über die entsprechenden x, y und z-Achsen [Sei14]. Man kann so ein Objekt mit einem Beschleunigungssensor steuern, was im folgenden Beispiel zu sehen ist.

```
void Update() {
    transform.Translate(Input.acceleration.x * 10, 0, 0);
}
```

Listing 2.14 Beispiel für Input.acceleration

Objekte die dieses Skript angehängt bekommen, lassen sich durch den Beschleunigungssensor in x-Richtung verschieben.

2.3.6 GUI

Dieser Abschnitt soll einen groben Überblick über die Möglichkeiten der Gestaltung einer grafischen Benutzeroberfläche (Graphical User Interface, GUI) geben. Es wird nicht im Detail darauf eingegangen, da nur elementare Grundlagen aus Unity genutzt werden. Da ab der Version 4.6 Unity ein neues GUI-System nutzt, wird auch dieses hier vorgestellt. Dieses neue System erlaubt es direkt in der Szene die GUI zu verändern und Arbeit mit 2D-Objekten, was vorher nur über Skripte möglich war [Sei14].

Das Hauptelement, um die GUI nutzen zu können ist das Canvas. Das Canvas muss von jeder GUI-Komponente das Eltern-Objekt sein. Es ist ein zweidimensionales Rechteck, was sich je nach Render Mode selbstständig dem Bildschirm des Ausgabegerätes anpassen kann und vor allen anderen Objekten liegt.

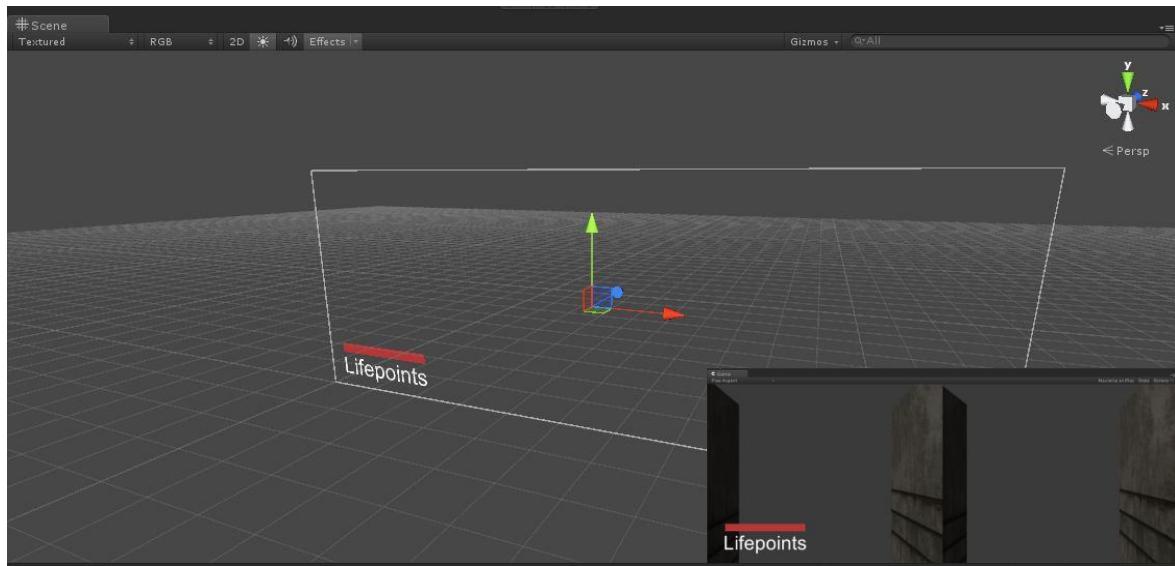


Abb. 2.12 Beispiel einer GUI im Scene und Game View

Die verschiedenen Render Modes auf welche Art das Canvas gerendert wird [Sei14]:

- > **Screen Space – Overlay** passt das Canvas der aktuellen Bildschirmgröße an und legt alle im Canvas enthaltenen Objekte über das Bild der Kamera
- > **Screen Space – Camera** passt das Canvas auch dem Bildschirm an. Hier kann zusätzlich aber eine Kamera zugewiesen werden, die für die Darstellung der GUI zuständig ist.
- > **World Space** ermöglicht es das GUI innerhalb der Szene zu platzieren. Das bedeutet, dass die GUI nicht über das Bild der Kamera gelegt wird, sondern als GameObject platziert werden kann.

Innerhalb des Canvas lassen sich GUI-Elemente platzieren. Diese besitzen keine herkömmliche Transform-Komponente sondern eine RectTransform-Komponente. Diese besitzt zusätzliche Eigenschaften, um die Breite und Höhe der GUI-Komponente zu verändern. Des Weiteren gibt

es einen Pivot-Punkt, der für Orientierung oder Skalierung als Ausgangspunkt genommen wird. Eine weitere Komponente sind Anchors. Diese können festlegen, wie sich ein GUI-Element zum Canvas verhalten soll. So soll z.B. bei Vergrößerung des Canvas die Anzeige für Lebenspunkte immer unten links sein. Unity hält dafür schon Vorgaben bereit, die man verwenden kann.

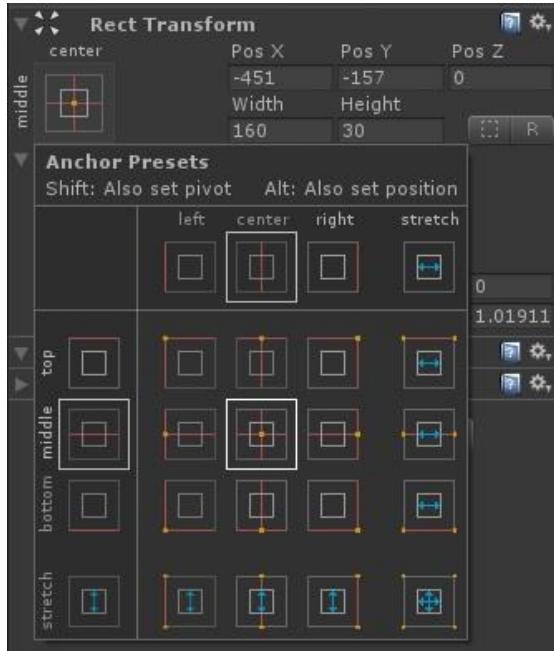


Abb. 2.13 Anchor-Vorgaben

Wie schon erwähnt gibt es noch deutlich mehr Möglichkeiten und Funktionalitäten. Diese werden im Rahmen dieser Arbeit nicht genutzt und daher auch nicht näher betrachtet.

2.3.7 Prefabs

Prefabs sind im eigentlichen Sinne Vorlagen. Diese verkörpern GameObjects, die mit Komponenten bestückt und angepasst wurden. Durch Prefabs ist es möglich beliebig viele Instanzen zur Laufzeit von einem GameObject zu erzeugen. Ein gutes Beispiel ist das Erstellen von Projektilen oder Gegnern, die zufällig in der Spielwelt erzeugt werden sollen [Sei14].

Um ein Prefab zu erzeugen, muss ein GameObject aus der Hierarchy einer Szene in der Project Browser gezogen werden. Unity sorgt dann dafür, dass aus dem GameObject ein Prefab erstellt

wird, welches Materialien, Texturen, Collider, usw. enthält. Diese Prefab erscheint dann im Project Browser [Sei14].

Instanzen kann man dann auf zwei Arten erstellen. Zum einen gibt es die Möglichkeit per Drag&Drop ein Prefab in die Szene zu ziehen. So können beliebig viele Instanzen in eine Szene platziert und auch verändert werden. Dabei wird das Original, also das Prefab nicht verändert [Sei14].

Die zweite Möglichkeit ist die Erstellung von Instanzen per Code. Das Stichwort ist hier Instantiate. Instantiate erzeugt eine Instanz von dem ihm übergebenen Prefab. Das Erzeugen so einer Prefab-Instanz sieht wie folgt aus:

```
public GameObject newPrefab;
void Start() {
    Instantiate(newPrefab, new Vector3(10,0,0), Quaternion.identity);
}
```

Listing 2.15 Instanzerzeugung über Code

Neben dem Prefab wird Instantiate noch eine Positionsangabe sowie Drehung im Quaternion Format übergeben. Quaternion ist ein Datentyp für Rotation. Quaternion.identity bedeutet, dass in diesem Fall die Instanz nicht noch zusätzlich gedreht werden soll [Sei14].

2.3.8 UnityPackage

UnityPackages sind Container, die man für den Austausch von Elementen zwischen verschiedenen Projekten verwendet. So kann z.B. eine konfigurierte Spielerfigur von einem Projekt in das andere übernommen und an dieser Stelle dann für das Projekt angepasst werden [Sei14]. Im Folgenden werden zwei genutzte UnityPackages kurz vorgestellt.

Durovis Dive

Das UnityPackage von Durovis Dive erleichtert die Platzierung von Kameras und den Zugriff auf die Sensoren des Smartphones. So wird ein Dive FPS Player Prefab zur Verfügung gestellt, welches in einer Szene platziert wird und neben einem steuerbaren Spieler-Objekt, auch zwei Kameras und die entsprechenden Skripte für den Zugriff auf die Sensoren enthält [@Dur].

Cardboard SDK für Unity

Das Cardboard SDK wird für den Vergleich zum Package von Durovis Dive hinzugezogen und ermöglicht für Unity neben dem vereinfachten Erstellen von VR-Kameras den Zugriff auf den Magnet-Schalter von Carboard. Dadurch ist es möglich durch das Betätigen des Schalters bestimmte vorher implementierte Ereignisse auszulösen. Des Weiteren liefert die SDK noch weitere Skripte die für die Entwicklung einer App für Cardboard benötigt werden können [@Car].

2.3.9 Debugging und Performance

Das Debugging oder auch Fehlersuche kann in Unity über MonoDevelop erfolgen. Hier kann wie auch in anderen Entwicklungsumgebungen über das Setzen von Breakpoints und dem Beobachten von Variablen eine Fehlersuche erfolgen [Sei14].

Um eine Echtzeitanalyse auf Endgeräten durchzuführen, bietet Unity aber noch weitere Werkzeuge. Diese dienen dazu die ausgeführte Anwendung z.B. auf dem Smartphone zu testen, Verhalten zu beobachten und Ausgaben zu analysieren. Dafür muss das mobile Gerät und der Entwickler-PC im gleichen WLAN sein und bestimmte Parameter aktiviert werden, damit MonoDevelop auf die Ausführung auf dem Gerät reagiert und der Debugger gestartet wird [Sei14].

Es können neben Fehlern auch Performance-Probleme auftreten, die sich erst bei der Ausführung auf der Endanwender-Umgebung bemerkbar machen. Das Werkzeug welches Unity dem Entwickler hier an die Hand legt ist der Profiler. Dieser gibt Auskunft über die Performance, was z.B. die CPU- oder GPU-Nutzung betrifft. Auch die Analyse von ausgeführten Skripten ist hier möglich [Sei14].

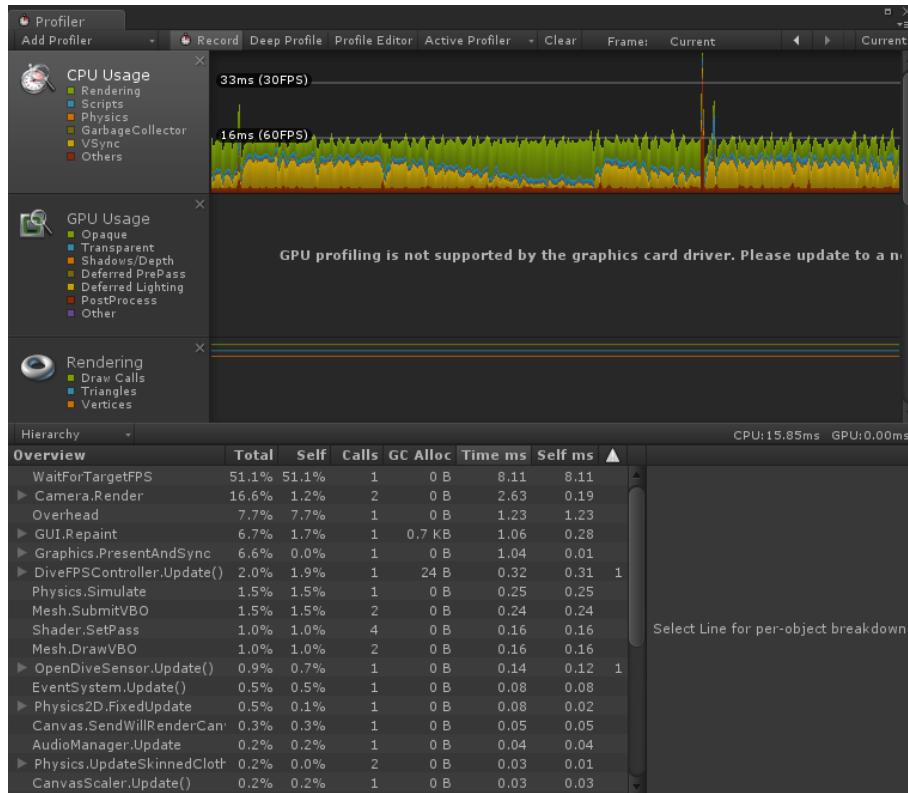


Abb. 2.14 Ansicht des Profilers: Darstellung der Performance eines Testgerätes

Da sich die Entwicklung der prototypischen App auf Android konzentriert bietet sich eine weitere Möglichkeit der Analyse von einer laufenden Anwendung. Die Android Debug Bridge (ADB) erlaubt eine Fehlersuche und Ausgaben zur Laufzeit der App. So kann über einen Aufruf in der Windows Kommandozeile eine Echtzeitüberwachung und Ausgabe gestartet werden. Dieser Aufruf erfolgt mithilfe von logcat, dem Logging System von Android. Diese gibt dann alles aus, was auf dem Gerät passiert und lässt sich über bestimmte Parameter auf Ausgaben von Unity reduzieren [Fin13][@ADB][@Cat].

```
adb logcat
adb logcat -s Unity
adb logcat -s Unity ActivityManager PackageManager dalvikvm DEBUG
```

Listing 2.16 Logging über adb logcat

2.3.10 Build Prozess

Um eine fertige App nun zu Erstellen und auf ein mobiles Gerät zu bringen, werden die Build Settings genutzt. Man kann hier Einstellungen bezüglich des Zielbetriebssystems vornehmen. Bei mobilen Geräten muss dieses per USB an den PC angeschlossen sein [Sei14].

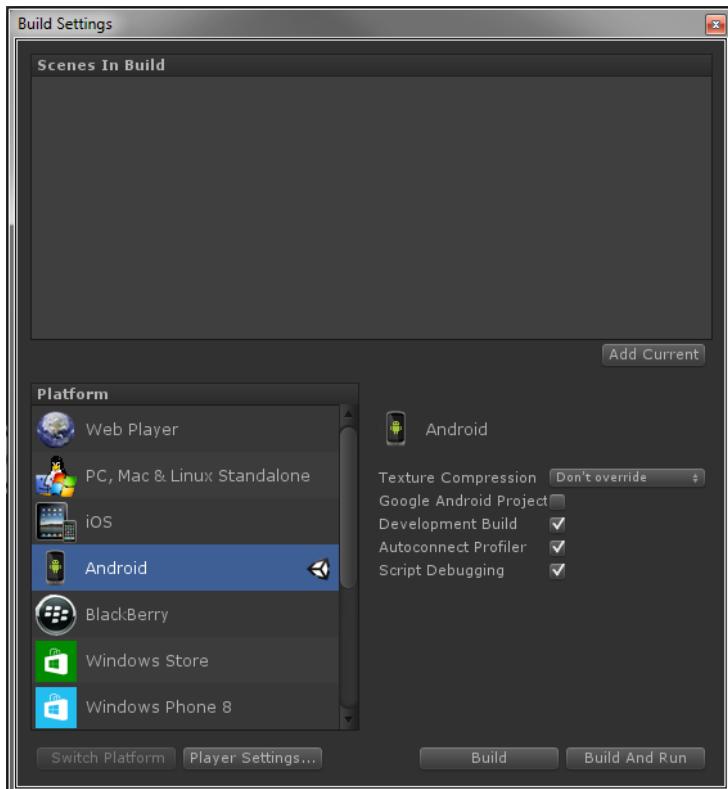


Abb. 2.15 Ansicht der Build Settings für Android

In den Build Settings können neben der Zielplattform auch die zu nutzenden Szenen sowie Einstellungen für das Debuggen und die Performancebeobachtung gemacht werden. Um eine App am Beispiel von Android nun auf das Testgerät zu übertragen, wird Build and Run betätigt. Dadurch erzeugt Unity eine entsprechende Datei, bei Android .apk, auf dem PC und überträgt dieselbe auf das Testgerät [Sei14].

3 Analyse

In diesem Abschnitt findet eine Analyse der umzusetzenden prototypischen Applikation sowie des Controllers statt. Die Evaluation des Controllers ist in den Abschnitten 4 und 5 zu finden. Dieser Teil soll der Abgrenzung dienen und einen Überblick über die Anforderungen an das umzusetzende Produkt geben. Dabei wird auf die Erkenntnisse aus dem Vorbericht [Bus14] aufgebaut.

3.1 Systemidee

Die Idee ist es einen Controller zu evaluieren, der eine prototypische VR-App aus Sicht des Nutzers steuern kann. Weiterhin sollen mögliche in der Entwicklung befindliche oder theoretische Ansätze betrachtet und mit einbezogen werden.

3.2 Stakeholder

Die Rolle der Stakeholder ist minimal, da die gewonnenen Erkenntnisse in kein aktuelles in der Entwicklung befindliches Projekt miteinfließen. Folgende Tabelle stellt eine Übersicht über vorhandene Stakeholder und deren Einfluss dar.

Rolle	Beschreibung	Wissen	Relevanz
Betreuer im Unterneh-	Ansprechpartner, gibt Ziele vor	Kennt sich mit der Thematik aus	Interesse an den gewonnenen Erkenntnissen, Nutzung der gewonnenen Erkenntnisse

Betreuer an der Hochschule	Ansprechpartner, hilft bei möglichen Problemen	Grundwissen vorhanden	Prüft die Ergebnisse
Firma	Stellt Entwicklungsumgebung	Grundwissen vorhanden	Interesse an Ideen oder dem Prototypen der Umsetzung, Nutzung dieser
Anwender	Ist ein Benutzer des Systems	Kein Wissen vorhanden	möglicher Tester der prototypischen App

Tabelle 3.1 Übersicht der Stakeholder

3.2.1 Stakeholdermap

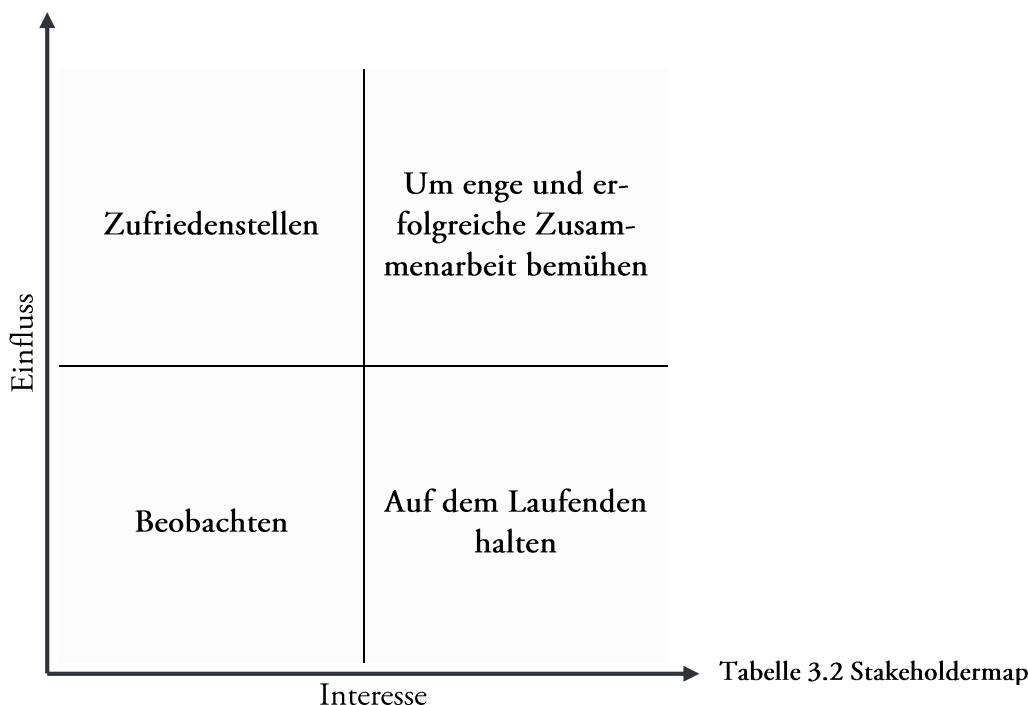


Tabelle 3.2 Stakeholdermap

Die Stakeholdermap verschafft eine Übersicht wie mit Stakeholdern umgegangen werden sollte. Je mehr Einfluss und Interesse ein Stakeholder hat, desto größer muss die Zusammenarbeit sein. Da in diesem Fall die Anzahl der Stakeholder überschaubar ist, findet hier nur eine kurze Einteilung statt.

- › Der **Betreuer im Unternehmen** hat ein großes Interesse an den Erkenntnissen, da diese für eine mögliche zukünftige Umsetzung von Anwendung nutzbar sein können. Der Einfluss ist im mittleren Bereich anzusiedeln, da die Aufgabe sehr frei bearbeitet werden kann und die Benotung der Arbeit auch mit einfließt. Daher muss der Betreuer im Unternehmen **auf dem Laufenden gehalten werden**.
- › Der **Betreuer an der Hochschule** hat ein mittleres Interesse und einen hohen Einfluss, der sich in der Benotung der Arbeit wiederspiegelt. Der Betreuer an der Hochschule ist **zufriedenzustellen und auf dem Laufenden zu halten**.
- › Die **Firma** hat die gleichen Interessen und Einfluss wie der Betreuer und ist daher auch im gleichen Bereich anzusiedeln.
- › Der **Anwender** hat wenig Interesse und einen mittleren Einfluss, der sich durch das Testen der App durch diesen wiederspiegelt. Daher ist der Anwender eher im Bereich **Beobachten** anzusiedeln und in der Funktion als Tester, die beobachteten Erkenntnisse mit einfließen zu lassen.

3.3 Ziele

Die Ziele sind unterteilt in muss, kann und wird nicht umgesetzt werden. Eine weitere Unterteilung in App und Controller wird vorgenommen.

3.3.1 Musskriterien

M1 VR-App

M1.1 Umsetzung für Android

M1.2 Umsetzung für aktuelle Smartphones

- M1.3 Steuerung durch Controller
 - M1.4 Erzeugt immersiven Eindruck durch virtuelle Kameras
- M2 Controller
- M2.1 Evaluation von verfügbaren und in der Entwicklung befindlichen oder theoretischen Controllern
 - M2.2 Steuert App
 - M2.3 Ermöglicht Bewegungssteuerung nach vorne, hinten, links und rechts

Tabelle 3.3 Musskriterien

3.3.2 Wunschkriterien

- W1 VR-App
- W1.1 Implementation von einer einfachen Spiellogik
- W2 Controller
- W2.1 Bewegungssteuerung erweitert durch die Möglichkeit von Sprüngen und in die Hocke gehen
 - W2.2 Kabellose Steuerung
 - W2.3 Steuerung ohne Hände

Tabelle 3.4 Wunschkriterien

3.3.3 Abgrenzungskriterien

Hier ist zu sagen, dass die zu implementierende App keine fertige App sein wird, sondern dem Zweck der Evaluation der Controller dienen soll. Des Weiteren können nicht alle genannten Controller im Zusammenspiel mit der App getestet werden, da mancher erst in einer prototypischen Ausführung vorhanden oder die Anschaffungskosten zu hoch sind.

3.4 Funktionale Anforderungen

Die funktionalen Anforderungen beschreiben Anforderungen an die App und an den Controller. Für einen besseren Überblick wurde ein Use-Case-Diagramm aufgestellt und Anforderungen nach Rupp [RS14] abgeleitet.

3.4.1 Use-Case

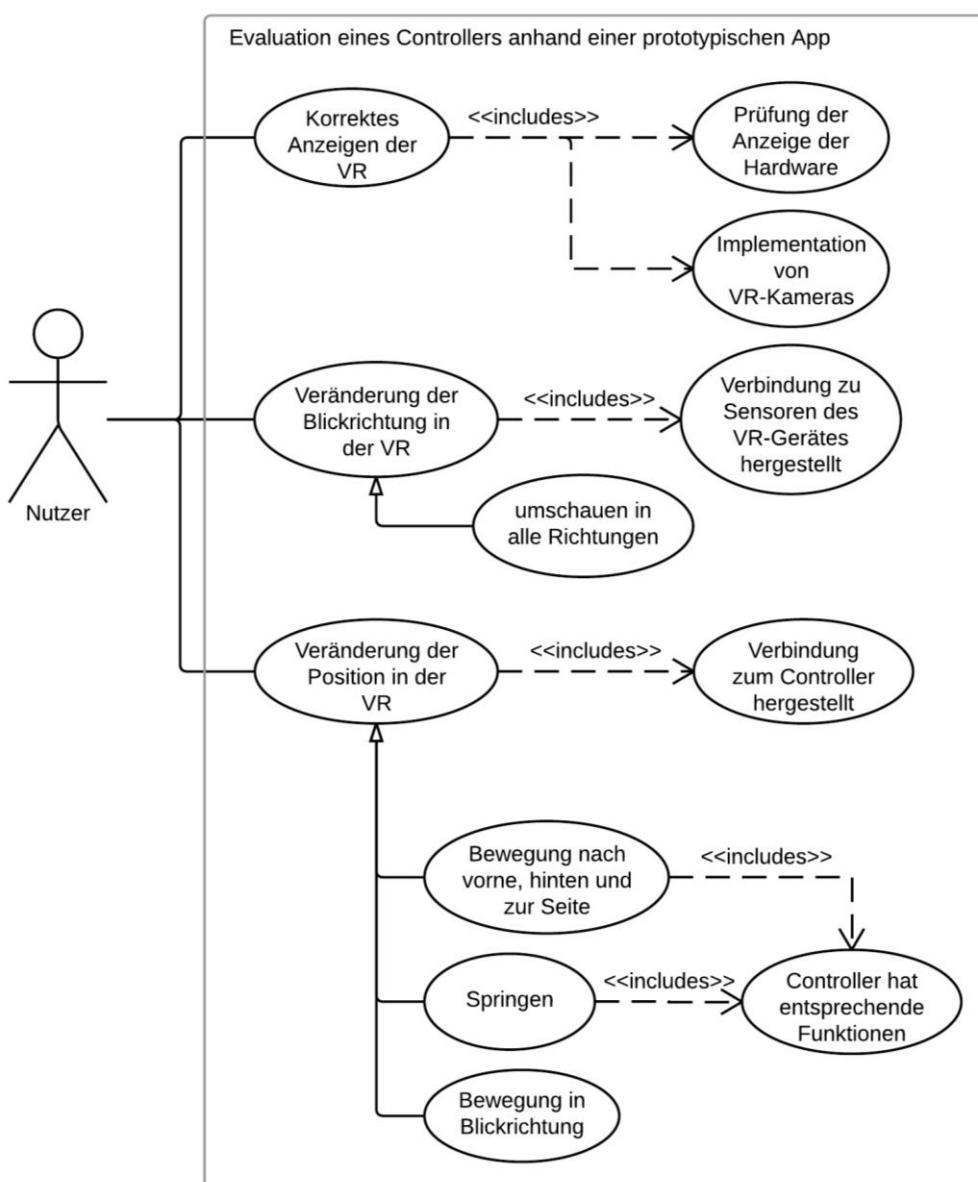


Abb. 3.1 Use-Case Diagramm

3.4.2 Anforderungen

A1 VR-App

- A1.1 Sobald die App gestartet ist muss diese dem Nutzer die Möglichkeit geben sich in der VR umzuschauen
- A1.2 Sobald ein Controller mit der App oder dem mobilen Gerät verbunden ist muss diese dem Nutzer die Möglichkeit geben sich in der VR nach vorne, hinten, links und rechts zu bewegen
- A1.3 Falls kein Controller mit der App verbunden ist sollte die App den Nutzer auffordern einen Controller zu verbinden
- A1.4 Solange die App mit einem Controller verbunden ist, muss eine Steuerung in der VR möglich sein

A2 Controller

- A2.1 Der Controller bietet die Funktionalität eine Bewegung nach vorne, hinten, links und rechts an die App zu senden
- A2.2 Sobald der Controller eine Verbindung mit dem mobilen Gerät hergestellt hat, muss diese Verbindung bestehen bleiben
- A2.3 Der Controller sollte mit einer Hand steuerbar sein
- A2.4 Der Controller sollte eine drahtlose Verbindung mit dem mobilen Gerät herstellen können
- A2.5 Der Controller wird ohne Hände steuerbar sein

Tabelle 3.5 Funktionale Anforderungen nach Rupp [RS14]

3.5 Nicht Funktionale Anforderungen

Dieser Abschnitt geht auf die nichtfunktionalen Anforderungen ein. Diese setzen sich aus mehreren Kategorien zusammen und sind nach Pohl und Rupp [PR11] als Anforderungen und Randbedingungen an die Qualität des Produktes definiert.

3.5.1 Technologisch

Der Controller muss für die Kommunikation mit dem mobilen Gerät Bluetooth unterstützen oder per Micro-USB Kabel an das mobile Gerät angeschlossen werden können. Dabei ist eine Verbindung per Bluetooth vorzuziehen, um dem Nutzer mehr Freiheit zu geben.

3.5.2 Qualitätsanforderungen

Q1 Benutzbarkeit

- Q1.1 Die Steuerung sollte für den Nutzer intuitiv und einfach zu erlernen sein
- Q1.2 Ist ein kabelloser Controller vorhanden sollte die Verbindung automatisch hergestellt werden
- Q1.3 Die VR bietet dem Nutzer ein immersives Gefühl

Q2 Effizienz

- Q2.1 Die VR-App muss so gestaltet sein, dass ein Betrieb auf einem mobilen Gerät möglich ist
- Q2.2 Die Eingaben des Controllers sollten ohne Verzögerung an die VR-App übertragen werden

Q3 Übertragbarkeit

- Q3.1 Eine Verbindung des Controllers mit der App sollte unabhängig vom mobilen Gerät sein
- Q3.2 Die Installation der App sollte auf unterschiedlichen unterstützten Geräten möglich sein

Q4 Zuverlässigkeit

- Q4.1 Bei ordnungsgemäßer Benutzung sollte der Controller nach dem Verbinden mit der App die Verbindung bestehen bleiben

Tabelle 3.6 Qualitätsanforderungen

3.6 Entwicklungsumgebung

Die Nutzung von Werkzeugen zur Entwicklung ergibt sich aus dem Projektbericht [Bus14] und den Erkenntnissen bei der Ermittlung geeigneter Controller. Des Weiteren findet eine Versionierung der App und des Projektes über GitHub statt. In folgender Übersicht sind alle genutzten Werkzeuge aufgelistet und das Zielsystem festgelegt.

Entwicklung

- > Unity
- > MonoDevelop/SublimeText
- > C#, Java
- > Google Cardboard SDK
- > Durovis Dive SDK
- > ADB
- > GitHub, Git

Zielsystem

- > Android

3.7 Tests

In diesem Abschnitt werden die Testgeräte und mögliche Testszenarien festgehalten, um die Funktionalität eines Controllers zu testen und mögliches Fehlverhalten oder Verbindungsabbrüche zu kompensieren. Die Tests werden mit Hilfe der nachfolgend aufgeführten Testgeräte durchgeführt.

3.7.1 Testgeräte

Als Testgeräte werden aktuelle und ältere Android-Smartphones genutzt. Um alle möglichen Controller testen zu können und einen möglichst breites Testresultat zu erhalten, werden Geräte verschiedener Hersteller mit unterschiedlichen Android-Versionen verwendet.

Testgerät	Android-Version	Jahr
Oneplus One	4.4.4	2014
Samsung Galaxy S3	4.0.4	2012
Samsung Galaxy S4	4.2.2	2013
Sony Xperia Z3	4.4.4	2014

Tabelle 3.7 Übersicht der Testgeräte

3.7.2 Testfälle

T1 VR-App

- T1.1 Übersetzung von schnellen Bewegungen des HMDs
- T1.2 Übersetzung von schnellen Eingaben des Controllers
- T1.3 Übersetzung von schnellen Bewegungen des Controllers

T2 Controller

- T2.1 Verbindungsaufbau
- T2.2 Verbindungsabbruch
- T2.3 Reichweite
- T2.4 Bewegungssteuerung nach mehrmaligem Verbindungsaufbau und Verbindungsabbruch

Tabelle 3.8 Testfälle

4 Analyse der Controller

Dieser Abschnitt dient der Analyse und genaueren Betrachtung der ermittelten Controller. Dabei wird noch keine Wertung vorgenommen. Diese findet im Abschnitt 6 statt. Die hier behandelten Controller setzen sich aus verfügbaren sowie nicht verfügbaren Prototypen und theoretischen Überlegungen zusammen. Die Analyse nicht vorhandener Controller und Prototypen findet auf Basis der Erkenntnisse aus bereits existierenden Apps, Produktbeschreibungen und Produktspezifikationen statt.

4.1 Gamepad: Playstation 3 Controller

Gamepads, insbesondere Playstation und Xbox Controller, bieten sich für die Steuerung von VR-Applikationen an, da der Anschluss an Konsolen oder Computer einfach ist und meist das Verbinden per Kabel ausreicht, um eine Verbindung herzustellen. Der Controller von der Wii von Nintendo eignete sich nicht, da hier Verbindungen über Bluetooth ab einer bestimmten Android Version nicht mehr möglich waren. Gamepads werden auch schon für die Nutzung von anderen HMDs wie die Oculus Rift über einen Computer genutzt. Da in diesem Fall der Anschluss mit einem mobilen Gerät erfolgen soll, wird der Controller der Playstation 3 näher betrachtet.

Der Controller der Playstation 3 eignet sich, da er über die nötigen Eingabemechanismen für eine Bewegungssteuerung verfügt und eine kabellose Verbindung mit einem mobilen Gerät zulässt. Weiterhin ist es möglich, Tests der App direkt am Computer durchzuführen bevor diese auf das mobile Gerät gespielt wird, da auch hier eine Verbindung möglich ist.

Die Verbindung zu einem mobilen Gerät erfolgt über eine externe Applikation, Sixaxis Controller, die es ermöglicht den Controller als natives Gamepad zu simulieren. Hier können auch Tastenbelegungen geändert und angepasst werden. Erforderlich für die Nutzung der App ist Root-Zugriff. Durch das Rooten erhält der Nutzer Administratoren-Rechte auf dem mobilen Gerät.

Für die Verbindung muss der Controller einmalig mit dem mobilen Gerät verbunden und gekoppelt werden. Danach ist eine kabellose Verbindung möglich.

Durch die Minijoysticks auf dem Gamepad ist eine präzise Steuerung möglich und durch die insgesamt hohe Anzahl an Knöpfen können noch weitere Eingaben implementiert und genutzt werden.

4.2 Magnetfeld: Cardboard

Cardboard von Google verfolgt eine andere Herangehensweise als das Gamepad. Hier werden Eingaben nicht über Joysticks oder Knöpfe erfasst, sondern über den Magnetischen Schalter (engl. Magnetic Switch) der am Gehäuse des Cardboard befestigt ist. Dieser funktioniert kabellos und ohne Bluetooth, lässt sich mit einer Hand bedienen und durch die Cardboard SDK kann ein Betätigen des Magnetic Switch registriert und verarbeitet werden [@Mag].

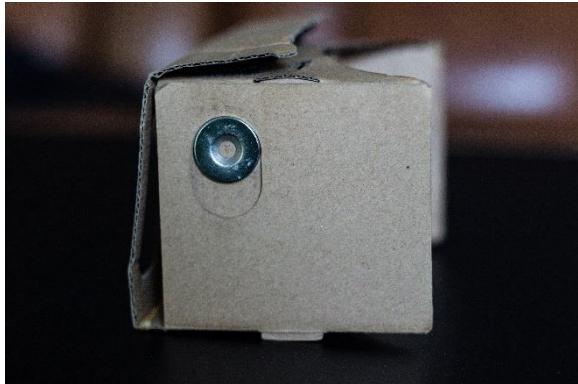


Abb. 4.1 Magnetic Switch des Cardboard [Bus]

Als Einschränkung ist zu nennen, dass mindestens Android 4.1 benötigt wird und das mobile Gerät in der Lage sein muss Veränderungen des Magnetfeldes zu registrieren.

Für die Umsetzung in Unity liefert Google eigene Prefabs mit, die in das Projekt eingebaut werden können. Die Prefabs enthalten folgende Skripte, um das Head Tracking und den Zugriff auf den Magnetic Switch zu gewährleisten.

- > Cardboard.cs – Verbindung zur Cardboard SDK
- > CardboardEye.cs – Ist für die virtuellen Kameras zuständig
- > CardboardHead.cs – Reduziert Latenz und verbessert Head Tracking
- > StereoController.cs – Ermöglicht Stereo sehen, verwaltet virtuelle Kameras

Die Cardboard SDK ermöglicht den Zugriff auf das Betätigen des Schalters durch die Variable CardboardTriggered. Diese boolsche Variable wird auf true gesetzt, wenn der Magnetic Switch betätigt wurde und bleibt genau einen ganzen Frame in diesem Zustand. Danach wird sie wieder auf false gesetzt. Der Zugriff auf diese Variable kann durch Unity aus jedem anderen Skript erfolgen und ermöglicht so diese Variable abzufragen und bestimmte Ereignisse auszuführen wie im Listing 4.1 beispielhaft dargestellt ist.

```
using UnityEngine;
using System.Collections;

public class CharacterController : MonoBehaviour {

    void Start () {
        //Hier Variablen initialisieren
    }

    void Update () {
        if(Cardboard.SDK.CardboardTriggered) {
            this.transform.Translate(Vector3(0,0,1));
        }
    }
}
```

Listing 4.1 Beispiel für den Zugriff auf den Magnetic Switch

Wird dieses Skript an ein Kamera-Objekt angehängt, wird durch das Betätigen des Magnetic Switch, das Kamera-Objekt um eine Einheit in Z-Richtung verschoben. Bei einer Bewegungs-erzeugung ist hier darauf zu achten eine gleichmäßige Verschiebung in die Richtung, in die die Kamera schaut zu implementieren und durch erneutes Betätigen des Magnetic Switch muss die Bewegung anhalten.

Eine weitere Möglichkeit ist, dass die Kamera nur bewegt wird, wenn der Magnetic Switch nach unten gezogen und gehalten wird. Lässt man diesen wieder los, wird auch die Bewegung gestoppt und lässt sich nur durch erneutes Ziehen und Halten wieder starten. Eine detailliertere

Betrachtung der möglichen Funktionen findet sich in Abschnitt 5, wo auf die Implementation der prototypischen App genauer eingegangen wird.

4.3 Inertialsensoren: Sphero und 3DRudder

Dieser Abschnitt behandelt zwei Controller, die beide Intertialsensoren verwenden, um Bewegung in VR zu übertragen. Beide Controller können nur theoretisch behandelt werden und nicht getestet, da der Anschaffungspreis für Sphero zu hoch ist und der 3DRudder nur als Prototyp vorhanden ist und noch nicht zum freien Verkauf zur Verfügung steht (Stand 28. Januar 2015).

4.3.1 Sphero

Sphero ist ein Ball der aus Plastik besteht und mit Gyroskop- sowie Beschleunigungssensoren ausgestattet ist. Er lässt sich über Bluetooth mit einem mobilen Gerät verbinden und über eine App steuern.

Umgekehrt ist es möglich den Ball als Controller für VR Apps auf mobilen Geräten zu verwenden. Sphero unterstützt asynchrones Übertragen von Daten, wodurch die Nutzung als Controller erst möglich gemacht wird. Durch das Kippen, Schütteln und Rotieren des Balles ist es möglich Eingaben zu simulieren und entsprechend umgewandelt an das mobile Gerät und die App zu übertragen. Dieser Zugriff wird durch ein vom Hersteller zur Verfügung gestelltes Unity-Plugin erleichtert. Mithilfe dieses Plugins wird das Abfangen von Daten des Controllers an das mobile Gerät und die Auswertung dieser ermöglicht. Ist eine Verbindung hergestellt, wird das Übertragen der Daten aktiviert und die Werte der Beschleunigungs-, Rotations- und Winkelbeschleunigungssensoren können ausgewertet werden.

```
m_Sphero.EnableControllerStreaming(20, 1,
    SpheroDataStreamMask.AccelerometerFilteredAll |
    SpheroDataStreamMask.QuaternionAll |
    SpheroDataStreamMask.IMUAnglesFilteredAll);
```

Listing 4.2 Aktiviert die Übertragung der Sensorwerte

Für das Steuern eines GameObject, in diesem Falle der Kamera, muss ein Skript an eine Kamera angehängt werden. Über dieses Skript können dann die Sensorwerte abgefragt werden und dadurch kann die Position und auch Rotation, was in diesem Fall nicht benötigt wird, verändert werden.

```
//Event Handler
SpheroDeviceSensorsAsyncData message =
    (SpheroDeviceSensorsAsyncData)eventArgs.Message;

SpheroDeviceSensorsData sensorsData = message.Frames[0];

float xAcceleration = sensorsData.AccelerometerData.Normalized.X;
float yAcceleration = sensorsData.AccelerometerData.Normalized.Y;
Vector3 currentPosition = transform.position;

// Create a new position by filtering the accelerometer data using
// the low pass
// filtering formula (alpha * filteredValue + (1 - alpha) * newValue)
position = new Vector3((0.9f * currentPosition.x + 0.1f * xAcceleration),
    (0.9f * currentPosition.y + 0.1f * yAcceleration), 0.0f);
```

Listing 4.3 Auszug aus dem Zugriff auf Sensorwerte [@Sph2]

Die Entstehung von Drift ist möglich, da durch schnelle Steuerungsbewegungen, sich aufaddierende Messfehler der Sensoren entstehen können.

4.3.2 3DRudder

Der 3DRudder ist ein Controller, der bis jetzt nur als Prototyp vorhanden ist, aber einen anderen Ansatz für die Steuerung von Anwendungen verfolgt. Im Gegensatz zu den bisher vorgestellten Controllern, bei denen die Eingabe mit den Händen erfolgte, werden hier die Füße genutzt. Für die Nutzung liegt der Controller auf dem Boden und die Füße werden darauf gestellt. Die eingebauten Sensoren können dann Neigungen des Controllers nach vorne, hinten, links und rechts sowie Rotationsbewegungen erkennen und über Kabel an ein entsprechend konfiguriertes Gerät bzw. eine App senden. Aktuelle Prototypen werden per USB mit einem PC

verbunden. Die Stromversorgung findet über das USB-Kabel statt. Ob mobile Geräte unterstützt werden ist noch offen.

4.4 Optisches Tracking: RGBD Kameras

Das optische Tracking durch Tiefe erfassende Kameras bietet aktuell das größte und interessanste Feld für die Steuerung in VR. Wichtigster Aspekt ist hier der hohe Grad der Immersion, da kein immersionsstörender Faktor wie ein Controller präsent ist, der mit den Händen oder Füßen bedient werden muss.

4.4.1 Leap Motion

Leap Motion ist darauf optimiert Finger- und Handbewegungen zu verfolgen. Der Leap Motion Controller kann per USB an ein Gerät angeschlossen werden. Er verfolgt im Gegensatz zur Kinect, die auf Ganzkörper-Verfolgung ausgelegt ist, nur einen kleinen Bereich bis zu einem Meter über dem Controller. Da Leap Motion hauptsächlich über die Verfolgung von Fingern und Händen funktioniert, ist es notwendig bestimmte Gesten oder Bewegungen für die Bewegungssteuerung in einer VR-App zu implementieren, damit eine Bewegung in der VR erzeugt wird, wenn eine bestimmte Geste erfolgt. Außerdem ist eine Montage von Leap Motion an das HMD nötig, um eine benutzerfreundlichere Interaktion mit dem Controller zu ermöglichen. Diese Montage gibt dem Nutzer dann die Möglichkeit den Controller nicht nur im Sitzen zu nutzen.

Über eine nicht für die Öffentlichkeit verfügbare SDK kann Leap Motion an einem mobilen Gerät betrieben werden. Diese befindet sich aktuell noch im Alpha-Stadium.

4.4.2 Kinect

Die Kinect bietet im Gegensatz zum Leap Motion Controller die Möglichkeit den ganzen Körper aufzunehmen. Hier kann der Nutzer ganze Körperbewegungen in eine VR übertragen und

auswerten lassen. So ist es möglich Sprünge, das Bewegen von Armen, sowie Lauf- und Gehbewegungen aufzunehmen und diese entsprechend in die VR umzusetzen. Für die Nutzung sind mehrere Komponenten erforderlich. Zum einen wird die Kinect benötigt, die an eine Stromquelle angeschlossen sein muss. Des Weiteren ist es notwendig einen Computer oder eine entsprechende Rechnereinheit zur Verfügung zu stellen, die genug Leistung hat, um übertragende Daten auswerten zu können [KHP11].

Für die Zusammenarbeit mit einem mobilen Gerät ist zudem eine Verbindung über ein Kabel zu dem Computer nötig oder eine Server-Client Verbindung. Ist eine solche Verbindung hergestellt, kann der Computer Bewegungen des Nutzers auswerten und an das mobile Gerät senden. Das mobile Gerät wiederum empfängt die Daten und entsprechenden Befehle werden an die VR-App gesendet. Für die Umsetzung der Server-Client Verbindung werden Kenntnisse in nativer Android Programmierung sowie C++ für die Schnittstelle zur Kinect benötigt [@Point].

Da der Nutzer keinen Controller in Händen hält, ermöglicht die Kinect einen hohen Grad an Bewegungsfreiheit sowie ein gesteigerten Eindruck von Immersion. Mit der Nutzung von mehreren Kinects ist es möglich den Nutzer selbst in die VR zu projizieren und ein dreidimensionales Abbild von diesem zu erschaffen [@Kin2]. Außerdem erhöht die Anzahl der Kinects den Raum in dem sich der Nutzer bewegen kann, ohne dass dieser aus dem Bereich austritt, der durch die Kameras erfasst wird.

Eine weitere Möglichkeit ist die Nutzung der Kinect als mobile Indoornavigationseinheit [@Kin]. Dabei wird der Tiefensor der Kinect genutzt um Abstände zu messen. Die Funktionalität verhält sich dabei genau umgekehrt. Nicht der Nutzer wird aufgenommen und seine Bewegungen verfolgt, sondern die Umgebung des Nutzers wird aufgenommen und ausgewertet. Die Kinect wird so am Nutzer befestigt, dass diese in die Blickrichtung des Nutzers ausgerichtet ist. Bewegt sich der Nutzer nun auf eine Wand, verringert sich der Abstand zu dieser und an das mobile Gerät kann eine Eingabe gesendet werden. Da die Kinect mit Strom versorgt werden muss, um zu funktionieren, muss hier noch eine mobile Stromversorgung bereitgestellt werden und eine Rechnereinheit, um die Daten auszuwerten. Diese Herausforderungen senken die Benutzerfreundlichkeit, da der Nutzer die Kinect, die Stromversorgung und die mobile Rechnereinheit mit sich herumtragen muss. Zudem ist der Formfaktor der Kinect ein weiterer störender Einfluss auf die Benutzerfreundlichkeit.

Genau diesen Herausforderungen der Benutzerfreundlichkeit bei der Umsetzung von Indoornavigation sowie die Möglichkeit, dass mobile Geräte Raum und Bewegung wahrnehmen und verarbeiten können, hat sich Project Tango verschrieben.

4.4.3 Project Tango

Project Tango ist ein von der Google Advanced Technology and Project group erdachtes Konzept für das Erkennen von Raum und Bewegung mit Smartphones und Tablets. Dabei ist benötigte Hardware und Software in das mobile Gerät integriert. Diese ermöglicht das Verfolgen von dreidimensionaler Bewegung und die genaue Bestimmung des mobilen Gerätes im Raum. Eine weitere Funktion ist, dass die aufgenommene Bewegung in ein dreidimensionales Modell umgewandelt werden kann.

Die genaue Funktionsweise ist nicht ersichtlich, da bisher nur Prototypen zur Verfügung stehen und auch ein Zugriff auf die SDK nicht möglich ist. Die zur Verfügung stehenden Informationen und Videomaterial zeigen aber die Möglichkeit, dass eine Fortbewegung in einer VR ohne weitere Controller möglich ist [@Tan2].

4.5 ODT: Virtuix Omni

Virtuix Omni ist ein Controller in Form einer Plattform. Der Nutzer wird hier auf dieser Plattform mit Hilfe eines Ringes an einer festen Position gehalten. Auf der rutschigen leicht konkav gewölbten Plattform kann der Nutzer laufen, rotieren und springen und bleibt durch die Fixierung des Ringes auf derselben Stelle.



Abb. 4.2 Virtuix Omni [Com]

Die Bewegungen werden durch spezielle Schuhe und durch das Haltegeschirr im Ring (Rotation) übertragen. Die Übertragung erfolgt via Bluetooth und lässt so eine einfache Bewegungssteuerung zu [@Omni].

Neben den hohen Anschaffungskosten kann auch die Größe von 1,20m x 1,40m x 1,00m eine Herausforderung an den Nutzer stellen und die Benutzerfreundlichkeit für VR mit mobilen Geräten einschränken.

5 Prototypische App

Die Idee für die prototypische App ist die Umsetzung eines Labyrinths, um dort die Bewegungsrichtungen zu testen. Der Nutzer muss dieses durchqueren und einen Würfel finden. Wird der Würfel gefunden, startet somit das Level neu. Der Nutzer hat zu jeder Zeit die Möglichkeit sich umzuschauen und in alle Richtungen zu bewegen.

5.1 Controller: Gamepad und Cardboard

Die beiden Controller für die die App implementiert wird, sind der Playstation 3 Controller sowie der Magnetic Switch von Google Cardboard.

5.2 Prefabs

Die Wände und der Boden des Labyrinths sind modulare Prefabs, die sich aneinander reihen lassen. Dadurch ist es möglich ein Labyrinth aufzubauen, ohne implementieren zu müssen.



Abb. 5.1 Prefabs für das Labyrinth

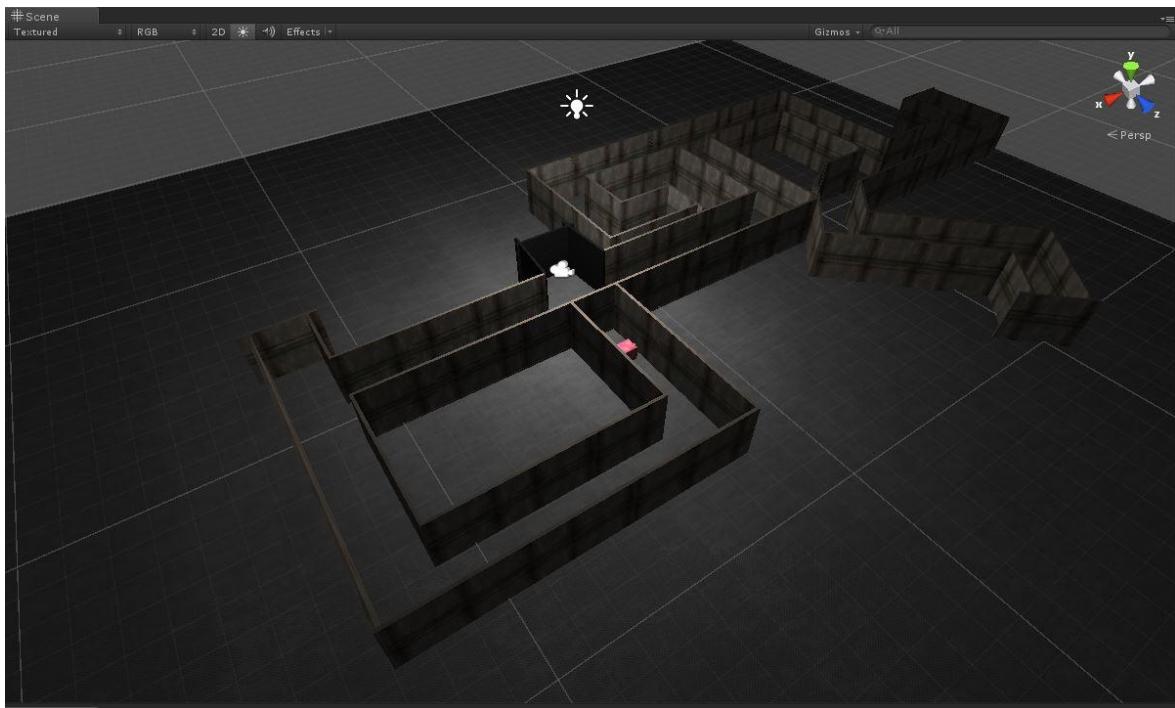


Abb. 5.2 Ansicht des fertigen Labyrinths

Des Weiteren bietet Cardboard Prefabs für die Nutzung des Magnetic Switch und die Erstellung von VR-Kameras. Hier wird das CardboardMain Prefab genutzt, was konfigurierte virtuelle Kameras beinhaltet.

Für das Gamepad wird die CharacterController Komponente von Unity genutzt und wie auch für Cardboard das *FPSInputController.cs* Skript.

5.3 Implementierung

Die Implementierung erfolgt anhand von Skripten, die an die entsprechenden Objekte angehängt werden. Dafür wird auf der einen Seite eine Funktionalität benötigt, die das Erreichen des Ziels registriert und das Level neu startet und auf der anderen Seite ein Skript, was ein automatisches Gehen ermöglicht, für die Umsetzung mit dem Magnetic Switch von Cardboard.

5.3.1 Allgemein

Dieser Abschnitt behandelt das Skript *GameController.cs* was für die Spiellogik verantwortlich ist und von der App genutzt wird. Hier wird das Erreichen des Ziels registriert und dadurch das Level neu gestartet.

Erreichen des Ziels

Für das Erreichen des Ziels wird auf eine Methode von Unity zurückgegriffen, die bei Kollision von zwei Objekten aktiviert wird. Trifft ein anderes Objekt auf diesen Auslöser (Trigger) wird eine Methode aufgerufen. Wie in Abb. 5.3 zu sehen, muss das Objekt eine Collider-Komponente besitzen, damit Kollisionen registriert werden. Des Weiteren muss Is Trigger aktiviert sein, um auf genannte Funktionalität zugreifen zu können.

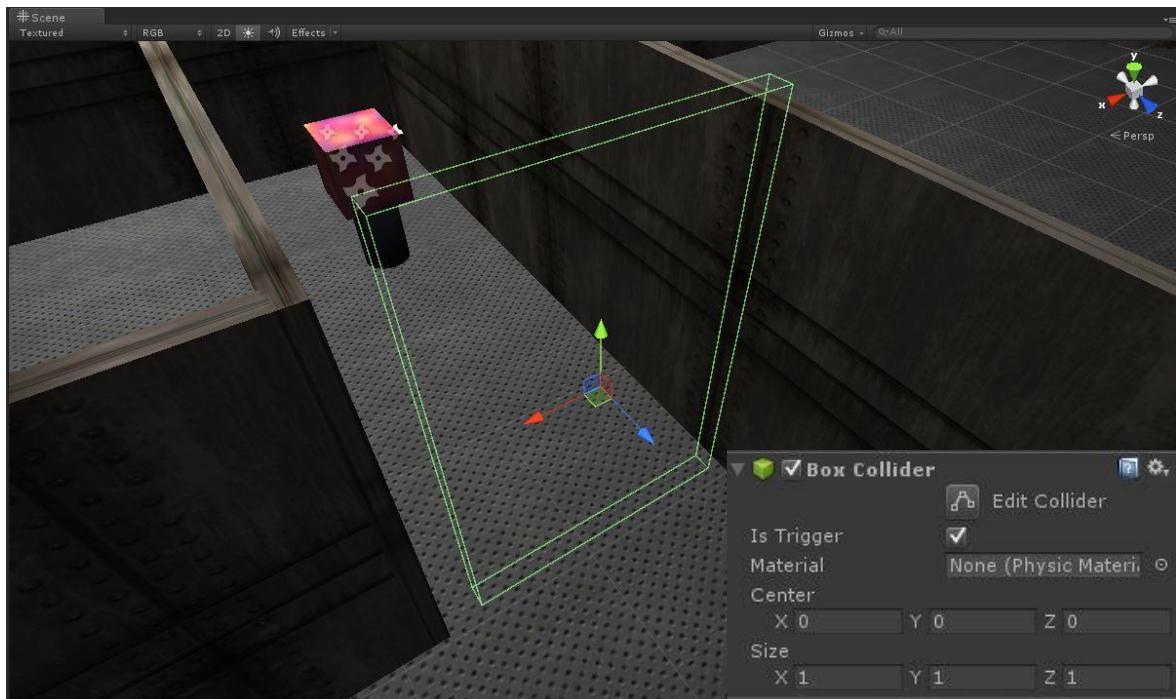


Abb. 5.3 Ansicht eines Trigger-Objektes vor dem Ziel des Labyrinths

Trifft das PlayerObject auf diesen Trigger wird die OnTriggerEnter Methode aufgerufen und der Level-Neustart eingeleitet, nachdem überprüft wurde, ob das PlayerObject das Objekt ist, welches den Trigger ausgelöst hat. Invoke verzögert dabei den Level-Neustart, um eine vorher eingestellte Variable.

```
void OnTriggerEnter (Collider other) {
    Debug.Log ("Finish reached");
    if (other.tag == "Player") {

        Invoke ("restartLevel", restartTime);
    }
}
```

Listing 5.1 Auslösung eines Triggers

Neustart des Levels

Ist die Zeit abgelaufen, wird das aktuelle Level neu geladen und der Nutzer kann von vorne beginnen.

```
void restartLevel () {
    Application.LoadLevel (Application.loadedLevel);
}
```

Listing 5.2 Neustart des Levels

5.3.2 Cardboard

Die Steuerung durch den Magnetic Switch wird durch die *Autowalk.cs* gesteuert. Diese greift auf Funktionen des Cardboard SDK, insbesondere auf die *FPSInputController.cs* zurück, um eine Betätigung des Magnetic Switch zu registrieren und eine automatische Bewegung in Blickrichtung des Nutzers auszulösen.

```
void Update () {  
  
    if(Cardboard.SDK.CardboardTriggered) {  
        GameObject FPSController = GameObject.Find ("Head");  
        FPSInputController autowalk =  
        FPSController.GetComponent<FPSInputController>();  
  
        autowalk.checkAutoWalk = !autowalk.checkAutoWalk;  
    }  
}
```

Listing 5.3 Pürfung des Magnetic Switch und Aktivierung von Autowalk

5.3.3 Gamepad

Auch das Gamepad nutzt die *FPSInputController.cs*. Ist die automatische Bewegung deaktiviert, werden Eingaben über vorher konfigurierte Achsen im InputManager ermöglicht. Dadurch ist eine Steuerung mit dem Gamepad ohne weiteres möglich.

```
if (!checkAutoWalk) {  
    directionVector = new Vector3(Input.GetAxis  
        ("Horizontal"), 0, Input.GetAxis("Vertical"));  
}
```

Listing 5.4 Aktivierung der Eingaben durch das Gamepad

6 Tests

Dieser Abschnitt führt die Tests auf, die mit den genutzten Controllern und der implementierten prototypischen Applikation durchgeführt wurden. Dabei werden die in Abschnitt 3.2.7 aufgeführten Testfälle durchgeführt und die Resultate hier aufgezeigt.

6.1 Gamepad

Testfall	Resultat
T1.1	Kann zu Drift führen und dadurch zu einer Verdrehung der virtuellen Kameras, was zu einer Schieflage des Blickwinkels führt. Diese ändert sich nach einigen Sekunden und die Kameras werden wieder normalisiert.
T1.2	Kann zu einer minimalen Latenz zwischen Eingabe und Bewegung in der VR führen.
T1.3	Bewegungen des Controllers haben keinen Einfluss auf die App, da dieser keine Bewegungssensoren für das Erfassen von Daten verwendet.

Tabelle 6.1 Testfälle der App

Testfall	Resultat
T2.1	Verbindungsaufbau benötigt Bluetooth sowie die einmalige Kopplung des Controllers mit dem mobilen Gerät.
T2.2	Verbindungsabbruch, bei zu weiter Entfernung des Controllers vom mobilen Gerät und bei Deaktivierung von Bluetooth.

-
- T2.3 Die Reichweite beträgt über 5 Meter ohne dass die Verbindung abbricht.
-
- T2.4 Nach einem Verbindungsabbruch ist eine erneute Verbindungsherstellung und das Steuern in der VR möglich, aber der Controller muss erneut mit dem Gerät gekoppelt werden bevor eine Bewegung in der VR erfolgt.
-

Tabelle 6.2 Testfälle des Controllers

6.2 Cardboard

Testfall	Resultat
T1.1	Kann zu Drift führen und dadurch zu einer Verdrehung der virtuellen Kameras, was zu einer Schieflage des Blickwinkels führt. Diese ändert sich nach einigen Sekunden und die Kameras werden wieder normalisiert.
T1.2	Bei zu schnellem Betätigen des Magnetic Switch werden nicht alle Eingaben registriert, bzw. es werden Eingaben übersprungen.
T1.7	Bewegungen des Controllers haben keinen Einfluss auf die App, da dieser keine Bewegungssensoren für das Erfassen von Daten verwendet.

Tabelle 6.3 Testfälle der App

Testfall	Resultat
T2.1	Verbindungsauftbau funktioniert, solange die Magnetfeldsensoren des mobilen Gerätes vorhanden sind und funktionieren.
T2.2	Verbindungsabbruch, bei zu weiter Entfernung des Controllers vom mobilen Gerät und bei Deaktivierung der Magnetfeldsensoren.

-
- T2.3 Der Controller darf nicht weiter als 3cm vom mobilen Gerät entfernt sein, damit eine Steuerung möglich ist.
-
- T2.4 Nach einem Verbindungsabbruch ist eine erneute Verbindungsherstellung und das Steuern in der VR ohne Probleme möglich
-

Tabelle 6.4 Testfälle des Controllers

7 Bewertung

In die Bewertung fließen alle gesammelten Erkenntnisse der behandelten Controller mit ein. Dabei steht der Nutzer eines solchen Controllers im Vordergrund und nicht die Verfügbarkeit. Dementsprechend wird auf Nutzerfreundlichkeit und die Verbindung mit einem mobilen Gerät besonderer Wert gelegt.

7.1 Allgemein

Allgemein ist zu sagen, dass aus Nutzersicht ein Controller zu bevorzugen ist, der die wenigsten Einschränkungen dem Nutzer gegenüber hat, da hier die Nutzererfahrung in Bezug auf VR und Immersion am größten ist. Einen solchen Controller muss der Nutzer nicht in den Händen halten, er funktioniert kabellos, es ist nicht nötig Markierungen für die Auswertung mit Kameras zu verwenden und der Nutzer kann sich frei bewegen sowie frei durch einen Raum gehen. Wichtige Kriterien sind eine möglichst niedrige Latenz, kaum Drift und ein hoher DOF. Ein weiterer Aspekt sind wirtschaftliche Kriterien, wie die Kosten, die aufgewendet werden müssen, um einen Controller nutzen zu können. So kann z.B. das optische Tracking durch den Einsatz von zusätzlichen Kameras verbessert und der Interaktionsbereich vergrößert werden, was wiederum Einfluss auf die Kosten hat, da mehrere Kameras angeschafft werden müssen.

Im Folgenden werden die Controller in zwei Abschnitten vorgestellt. Verfügbar bedeutet, dass die Controller im Handel zu kaufen sind und teilweise getestet werden konnten. Prototypen sind Controller, die es noch nicht im Handel gibt und nur als Prototypen vorhanden sind. Diese werden auf Basis der bekannten Produktspezifikationen und erhaltenen Eindrücke bewertet.

7.2 Verfügbare Controller

Das Gamepad, hier der Playstation 3 Controller, und Cardboard von Google sind zwei Controller die getestet werden konnten. Sphero und die Kinect konnten nicht in der Praxis getestet werden, sind aber auf dem Markt verfügbare Eingabegeräte.

Gamepad

Die Vorteile an einem Gamepad als Controller sind die kabellose Verbindung über Bluetooth zu einem mobilen Gerät und die intuitive Steuerung. Was negativ auffällt ist, dass die erste Verbindung eine Herausforderung darstellen kann und eine externe App benötigt wird, um Controller und mobiles Gerät miteinander zu koppeln. Nachdem aber die erste Verbindung hergestellt ist, wird der Controller als natives Eingabegerät von Android erkannt und ist dadurch mit vielen Apps kompatibel.

Der Nachteil liegt hier darin, dass das Gamepad in den Händen gehalten werden muss und der Nutzer sich nicht selber bewegt, um Bewegung zu erzeugen, was den Grad der Immersion verringert.

Cardboard

Das Cardboard von Google hat den Vorteil, dass der Controller direkt im Gehäuse des HMDs eingearbeitet ist und somit eine extra Anschaffung für einen Controller entfällt, beim Kauf dieses HMDs. Des Weiteren sind die Anschaffungskosten sehr gering und es wird trotzdem in Verbindung mit einem mobilen Gerät ein hoher Grad der Immersion geboten. Ein weiterer Vorteil ist die kabellose Verbindung des Controllers mit der App, da hier keine Verbindung im eigentlichen Sinne auftritt, sondern nur Veränderungen im Magnetfeld als Auslöser für Aktionen dienen. Was wiederum auch einige Nachteile zur Folge hat.

Zum einen erlaubt die Steuerung durch den Magnetic Switch nur eine begrenzte Möglichkeit an Bewegungen, da es entweder den Fall Switch betätigt oder nicht betätigt gibt. Zum anderen können nur Apps, die das Cardboard SDK nutzen entsprechend auf Eingaben reagieren.

Sphero

Sphero hat den Vorteil, dass dieser nur mit einer Hand bedient werden muss und eine intuitive Steuerung ermöglicht. Außerdem ist eine kabellose Verbindung über Bluetooth möglich. Dem gegenüber stehen die hohen Anschaffungskosten und ein hoher Programmieraufwand, da viel ausgetestet werden muss, um eine für den Nutzer optimale Steuerung zu schaffen. Zudem kann Drift auftreten, da die gesammelten Daten auf Inertialsensoren beruhen. Das Gefühl der Immersion kann außerdem für manche Nutzer gestört werden, da hier Vorwärtsbewegung durch das Neigen des Controllers erfolgt.

Kinect

Die Kinect hat gegenüber den bisher genannten Controllern den großen Vorteil, dass hier keine händische Bedienung notwendig ist. Es werden Bewegungen des Nutzers erkannt und können in die VR übertragen werden. Weiterhin ist eine kabellose Übertragung an mobile Geräte möglich, was aber auch einen Aufwand benötigt, da hier eine Client-Server Umsetzung erforderlich ist.

Der Nachteil liegt hier in der Rechnereinheit, die benötigt wird um die Daten der Kinect auswerten zu können. Außerdem ist eine konstante Stromversorgung Voraussetzung, was den mobilen Einsatz der Kinect deutlich verringert.

Leap Motion

Der Leap Motion Controller bietet den Vorteil, dass er klein und portabel ist. Der Controller muss nicht mit den Händen bedient werden. Die Hände sind hier Mittel zum Zweck, um durch Gesten eine Steuerung zu ermöglichen.

Der Nachteil besteht hier in der Montage per Kabel an ein mobiles Gerät und das aktuell keine öffentliche SDK für Android oder Unity zur Verfügung steht [@Leap2].

7.3 Prototypen

Omni

Der Vorteil von Omni liegt in der Fixierung des Nutzers und der Art wie Bewegungen in eine VR übertragen werden. Im Gegensatz zur Kinect, kann der Nutzer nicht aus dem Aufnahmeradius austreten und die Verfolgung von Bewegung kann konstant ohne Abbrüche erfolgen. Der Grad der Immersion ist sehr hoch, da Eigenbewegungen des Nutzers Bewegung in der VR erzeugen. Außerdem ist eine kabellose Übertragung über Bluetooth möglich und mobile Geräte sollen laut Entwickler das Omni als natives Eingabegerät erkennen können, so dass eine einfache Kopplung möglich ist [@Omni2].

Dem gegenüber stehen die sehr hohen Kosten, die sich mit allem benötigtem Equipment auf bis zu 1000\$ steigern können. Ein weiterer negativer Aspekt sind die Maße des Omni. Für den seltenen Gebrauch eines Controllers nimmt dieser zu viel Platz weg. Wie bei der Kinect wird eine Stromversorgung benötigt.

3DRudder

Der 3DRudder stellt eine interessante Alternative zwischen den bisher genannten Controllern dar. Ein großer Vorteil liegt darin, dass die Hände frei sind und die Steuerung ausschließlich durch die Füße erfolgt. Außerdem erfolgt die Stromversorgung durch das USB-Kabel, das zum Verbinden mit einem Computer vorhanden ist. Weiterhin wird die Immersion der VR nur in kleinem Maße beeinträchtigt, da Bewegung in der VR durch das Bewegen der Füße des Nutzers ausgelöst wird.

Der große Nachteil liegt hier in der Verbindung. Aktuell unterstützt der 3DRudder nur die Verbindung per USB und ist daher schlecht geeignet für die Kopplung mit einem mobilen Gerät.

Project Tango

Project Tango hat von allen vorgestellten Controllern den großen Vorteil darin, dass mobiles Gerät und Sensoren für die Steuerung in einem integriert sind. Dadurch wird kein externer Controller benötigt und deshalb muss auch keine Verbindung per Kabel oder Bluetooth hergestellt werden. Es wird ein hoher Grad der Immersion erreicht. Durch die aufgenommenen Daten der integrierten Sensoren ist die Bewegungssteuerung von VR-Apps leicht zu erreichen.

Ein Nachteil können womöglich die anfallenden Kosten sein, da in diesem Fall mobiles Gerät und Controller in einem vorhanden sind.

7.4 Übersicht

	Cardboard	Gamepad	Sphero	Kinect	Leap Motion	ODT	3DRudder	Project Tango
mit Kabel					•		•	
händische Steuerung	•	•	•					
zusätzliche Software benötigt	•	•	•	•	•	•	•	•
Drift kann auftreten				•	•	•	•	•
Latenz kann auftreten	•	•	•	•	•	•		
stört Immersion	•	•	•					
hohe Kosten			•	•	•	•	•	•

Tabelle 7.1 Übersicht der Controller

7.5 Ergebnis

Von allen vorgestellten Controllern ist Project Tango von Google der vielversprechendste. Die schon integrierten Sensoren und Tiefenkameras erlauben die Fortbewegung in einer VR, ohne dass ein externer Controller nötig ist. Insbesondere für VR auf mobilen Geräten stellt Tango einen großen Fortschritt dar, da es sich selbst im Raum lokalisieren kann. Hier bleibt abzuwarten, wie die Entwicklung fortschreitet und in welchem Preisrahmen Geräte wie Project Tango am Ende landen.

Von den verfügbaren Controllern ist das Gamepad am besten geeignet, da es Aufgrund der niedrigen Kosten und einfachen Kopplung mit einem mobilen Gerät mehr Vorteile bietet als Nachteile hat.

.

8 Zusammenfassung

Am Anfang dieser Arbeit wurden die Erkenntnisse aus dem Vorprojekt analysiert. Darauf aufbauend fand eine Analyse der umzusetzenden Aufgabe statt. Dies beinhaltete die Erfassung von den Zielen die erreicht werden sollten, den Anforderungen an den Controller und die Applikation, Erfassung von Testfällen sowie das Bereitstellen von Testgeräten.

Der nächste Schritt bestand in der Recherche verfügbarer und in Entwicklung befindlicher Controller. Dabei wurden geeignete Controller gefunden, die unterschiedliche Ansätze zur Bewegungssteuerung in Virtual Reality bieten. Zwei Controller sind für spätere Tests beschafft worden.

Ein weiterer wichtiger Aspekt war die Umsetzung einer prototypischen Applikation, um die Controller, die zur Verfügung standen, zu testen. Abschließend fand eine Bewertung anhand der Recherche und durchgeführten Tests statt.

8.1 Ergebnis

Als Ergebnis dieser Arbeit sind eine Übersicht sowie eine Bewertung von Controllern für Virtual Reality mit mobilen Geräten entstanden, die Vor- und Nachteile der einzelnen Eingabegeräte darlegen. Weiterhin werden Einblicke in die aktuelle Entwicklung von Virtual Reality gegeben und verschiedene Möglichkeiten der Bewegungssteuerung aufgezeigt. Außerdem wurden in Entwicklung befindliche Controller näher betrachtet und in die Bewertung mit einbezogen. Zudem ist eine Applikation für Android entstanden, die das Testen mit verfügbaren Controllern erlaubte und einen umfangreichen Aufschluss über die negativen sowie positiven Eigenschaften der Controller gab. Ferner wurden neben den Controllern auch die notwendigen Mittel für die Erstellung und das Erleben einer Virtual Reality auf mobilen Geräten aufgezeigt.

8.2 Fazit

Diese Arbeit hat gezeigt, wie wichtig Planung und Zeitmanagement für den Verlauf eines Projektes sind. Außerdem konnten Einblicke und Erfahrungen im Gebiet der Virtual Reality gesammelt und in Form einer erstellten Applikation angewendet werden.

Weiterhin sind die Vorzüge einer Entwicklungsumgebung wie Unity deutlich geworden. Durch den komponentenbasierten Aufbau der Objekte und der Programmierung über Skripte konnte die Applikation schnell umgesetzt werden.

8.3 Ausblick

Interessant ist es zu künftig zu beobachten, wie die Entwicklung von Projekten, wie Project Tango, voranschreitet und ob sich neben den hier vorgestellten Ansätzen der Bewegungssteuerung für Virtual Reality noch weitere entwickeln werden.

Eine Erweiterungsmöglichkeit ist das Testen mit Controllern, die zum Zeitpunkt dieser Arbeit noch nicht zur Verfügung standen. Dafür müsste die App entsprechend angepasst werden, um die Funktionen der Controller nutzen zu können.

A Referenzen

Berichte

- [Bus14] T. Busch: „Einarbeitung in Virtual Reality und Augmented Reality durch die Umsetzung von prototypischen Applikationen“, Osnabrück, November 2014
- [Kam12] F. Kammergruber et al.: „Navigation in Virtual Reality using Microsoft Kinect“, Taipei, November 2012

Bücher

- [Bla11] S. Blackman: „Beginning 3D Game Development with Unity: The World’s most widely used multiplatform game engine“, Apress, New York, Mai 2011
- [Dör13] R. Dörner et al. (Hrsg): „Virtual und Augmented Reality (VR/AR): Grundlagen und Methoden der Virtuellen und Augmentierten Realität“, 1.Auflage, Springer Verlag, Berlin Heidelberg, September 2013
- [Fch10] M. Füchl: „Innovative Bedienkonzepte: Betrachtung der Marktsituation verschiedener Produkte mit Kundenbefragung von Microsoft Surface im Bezug auf zukünftige Erwartungen und Anwendungsfälle“, 1.Auflage, diplom.de-Verlag, April 2010
- [Fin13] T. Finnegan: „Unity Android Game Development by Example Beginner’s Guide“, 1. Auflage, Packt Publishing, Birmingham, Dezember 2013
- [KHP11] S. Kean, J. Hall, P. Perry: „Meet the Kinect: An Introduction to Programming Natural User Interfaces“, 1. Auflage, Apress, Dezember 2011
- [Kra12] J. Kramer et al. : „Hacking the Kinect“, 1. Auflage, Apress, März 2012
- [PR11] K. Pohl, C. Rupp: „Basiswissen und Requirements Engineering: Aus- und Weiterbildung zum Certified Professional for Requirements Engineering – Foundation Level nach IREB-Standard“, 2. Auflage, dpunkt-Verlag, Heidelberg, Juni 2011

- [RS14] C. Rupp, die SOPHISTen: „Requirements-Engineering und -Management: Aus der Praxis von klassisch bis agil“, 6. Auflage, Carl Hanser Verlag, München, Oktober 2014
- [Sei14] C. Seifert: „Spiele entwickeln mit Unity: 3D-Games mit Unity und C# für Desktop, Web & Mobile“, 1. Auflage, Carl Hanser Verlag, München, September 2014

Magazine

- [FRE03] K.J. Fernandes, V. Raja, J. Eyre: „Cybersphere: the fully immersive spherical projection system“, Communications of the ACM 46 Nr. 9, ACM, New York, September 2003

Webseiten

zuletzt am 01.02.2015 abgerufen.

- [@Adb] Android Debug Bridge, <http://developer.android.com/tools/help/adb.html>
- [@Car] Google Cardboard Developer Documentation, <https://developers.google.com/cardboard/overview>
- [@Cat] ADB Logcat, <http://developer.android.com/tools/help/logcat.html>
- [@Dur] Durovis Dive SDK, <https://www.durovis.com/sdk.html>
- [@Gear] Gear VR, <http://www.samsung.com/global/microsite/gearvr/index.html>
- [@Kin] Kinect Indoor Navigation, <http://www.electronicsnews.com.au/features/kinect-arduino-hack-for-indoor-navigation-video->
- [@Kin2] Übertragung des Körpers in eine VR, <http://techcrunch.com/2014/05/14/clever-kinect-hack-brings-this-guys-full-body-into-virtual-reality/>
- [@Leap] Leap Motion Forum, <https://forums.leapmotion.com/>
- [@Leap2] Leap Motion Android SDK, <https://community.leapmotion.com/t/android-sdk-progress-update/2115>

- [@Mag] Magnet Button for Cardboard, <https://www.quora.com/How-does-the-magnet-select-button-for-Android-Cardboard-work>
- [@Ocu] Oculus Rift Support, <https://support.oculus.com>
- [@Omni] Virtuix Omni, <http://www.virtuix.com/>
- [@Omni2] Virtuix Omni Bluetooth Functionality, <http://www.virtuix.com/bluetooth-and-samsung-gear-vr/>
- [@Rud] 3DRudder, <http://www.3drudder.com/>
- [@Six] Sixense STEM System, <http://sixense.com/wireless>
- [@Sph] Sphero, <http://www.gosphero.com/de/>
- [@Sph2] Sphero Unity Plugin: Sensor Streaming, <https://github.com/orbotix/UNITY-PLUGIN/tree/master/ExampleProject/SensorStreaming>
- [@Tan] Project Tango, <https://www.google.com/atap/projecttango/#project>
- [@Tan2] Project Tango Video, <https://www.youtube.com/watch?v=Qe10ExwzCqk#t=50>
- [@Uni] Unity Documentation, <http://unity3d.com/learn/documentation>

Bildquellen

- [Bus] Bilder von Tobias Busch
- [Com] Wikimedia Commons: Eine freie Sammlung von Bildern, www.commons.wikimedia.org
- [Eta] Bilder von der Firma Die Etagen
- [Orb] Bilder von Sphero, Nutzung durch Orbotix genehmigt, <https://brand-folder.com/sphero2>
- [Rud] Bilder von 3DRudder, <http://www.3drudder.com/>

B Inhalt der CD

In der beigefügten CD sind folgende Ordner und Dateien enthalten.

Ordnerverzeichnis	Dateien	Beschreibung
\Bachelorarbeit	Bachelorarbeit_BuschTobias.pdf Bachelorarbeit_BuschTobias.docx	Die Bachelorarbeit im Portable Document Format (PDF) Die Bachelorarbeit im Microsoft Word Format
\Bilder	*.jpg, *.png	Verwendete Bilder in größerem Format
\Quellen	*.pdf	Benutzte Internetseiten
\unity-project	*.* \builds*.apk	Enthält Testapplikationen, die mit Unity erstellt wurden In dem builds-Ordner befinden sich installierbare *.apk-Dateien für die Installation auf Android-Geräten
\Videos	*.mp4	Videos der getesteten Controller und der Applikation

Tabelle B.1 Inhalt der CD

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche einzeln kenntlich gemacht. Es wurden keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

.....
Ort, Datum

.....
Unterschrift