

Hochschule Osnabrück

University of Applied Sciences

Fakultät

Ingenieurwissenschaften und Informatik

BACHELORARBEIT

Evaluation eines Controllers für die Fortbewegung in einer Virtual Reality anhand einer prototypischen Anwen- dung 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

Abstract

II 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.....</i>	9
2.2	VIRTUAL REALITY FÜR MOBILE GERÄTE	13
2.2.1	<i>Boxx3D.....</i>	14
2.2.2	<i>Google Cardboard.....</i>	14
2.2.3	<i>Gear VR.....</i>	15
2.3	UNITY.....	16
2.3.1	<i>Programmierung über Skripte.....</i>	16
2.3.2	<i>Objekte im 3D-Raum</i>	21
2.3.3	<i>Kamera</i>	22
2.3.4	<i>Physik</i>	23
2.3.5	<i>Inputmöglichkeiten.....</i>	24
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>	32
2.3.10	<i>Build Prozess.....</i>	34
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
3.4	SYSTEMKONTEXT	39
3.5	SYSTEMABGRENZUNG	39
3.6	FUNKTIONALE ANFORDERUNGEN.....	39

II Inhaltsverzeichnis

3.6.1	<i>Use-Case</i>	39
3.6.2	<i>Anforderungen</i>	40
3.7	NICHT FUNKTIONALE ANFORDERUNGEN.....	40
3.7.1	<i>Technologisch</i>	40
3.7.2	<i>Benutzeroberfläche</i>	40
3.7.3	<i>Qualität</i>	40
3.7.4	<i>Durchzuführende Tätigkeiten</i>	40
3.7.5	<i>Rechtlich-vertraglich</i>	40
3.7.6	<i>Hier fehlt noch eine</i>	40
3.8	TESTS.....	40
3.8.1	<i>Testgeräte</i>	40
3.8.2	<i>Testszenarien</i>	41
4	ANALYSE VORHANDENER CONTROLLER.....	42
4.1	VERFÜGBARE CONTROLLER.....	42
4.1.1	<i>Veränderung des Magnetfeldes</i>	42
4.1.2	<i>Gyroskop</i>	42
4.1.3	<i>Gamepad</i>	42
4.1.4	<i>Kamera die Bewegungen aufnimmt kinect und tango</i>	42
4.1.5	<i>ODT Virtuix Omni</i>	42
5	BEWERTUNG	43
5.1	MÖGLICHE ANSÄTZE	43
5.2	THEORETISCHE ANSÄTZE	43
5.3	AUSWAHL.....	43
6	IMPLEMENTATION.....	44
6.1	GUI	44
6.2	VERBINDUNG DES MOTION CONTROLLERS.....	44
6.3	AUSLESEN DER DATEN	44
6.4	SZENE	44
6.5	SPIELLOGIK	44
7	TESTS	45
7.1	SCHNELLE BEWEGUNGEN	45
7.2	BEWEGUNGSRÄUME BEI FALSCHER HANDHABUNG	45
7.3	ERFAHRUNGEN	45
8	FAZIT UND AUSBLICK.....	46
8.1	BEWERTUNG.....	46
8.1.1	<i>Controller</i>	46
8.1.2	<i>Applikation</i>	46

8.2 AUSBLICK.....	46
8.2.1 <i>Prototypen von Controllern, was kommt ist in Arbeit.....</i>	46
A REFERENZEN.....	47
B INHALT DER CD.....	50

III Abbildungsverzeichnis

ABB. 2.1 PRINZIPIELE 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 KINECT UND LEAP MOTION.....	10
ABB. 2.4 ODT MIT HALTERUNG FÜR DEN NUTZER	11
ABB. 2.5 VERSCHIEDENE ANSICHTEN DER BOXX3D.....	14
ABB. 2.6 VERSCHIEDENE ANSICHTEN DER CARDBOX.....	15
ABB. 2.7 ANSICHT DER GEAR VR.....	15
ABB. 2.8 INSPECTORANSICHT NACH ZUWEISUNG EINER PUBLIC-VARIABLEN.....	18
ABB. 2.9 ANSICHT DES INPUT-MANAGERS.....	25
ABB. 2.10 BEISPIEL EINER GUI IM SCENE UND GAME VIEW	29
ABB. 2.11 ANCHOR-VORGABEN	30
ABB. 2.12 ANSICHT DES PROFILERS: DARSTELLUNG DER PERFORMANCE EINES TESGERÄTES	33
ABB. 2.13 ANSICHT DER BUILD SETTINGS FÜR ANDROID.....	34

IV 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 ÜBERSICHT DER TESTGERÄTE.....	40
TABELLE 3.6 TESTSzenarien.....	41
TABELLE B.1 INHALT DER CD	50

V Listings

LISTING 2.1 GRUNDGERÜST EINES C#-SKRIPTS.....	17
LISTING 2.2 ZUGRIFF AUF EIGENE KOMPONENTE EINES OBJEKTES	18
LISTING 2.3 INSPECTOR ZUWEISUNG EINER PUBLIC VARIABLEN.....	18
LISTING 2.4 AUFBAU EINER COROUTINE.....	19
LISTING 2.5 BEISPIEL INVOKE.....	19
LISTING 2.6 SPEICHERN VON DATEN	20
LISTING 2.7 LADEN VON DATEN	20
LISTING 2.8 DEBUG-MÖGLICHKEITEN.....	21
LISTING 2.9 BEISPIELE FÜR DIE ERZEUGUNG EINES VEKTORS.....	21
LISTING 2.10 ZUGRIFF AUF EINGABEN DURCH GETAXIS	27
LISTING 2.11 ZUGRIFF AUF EINGABEN MIT GETBUTTON [SEI14].....	27
LISTING 2.12 ZUGRIFF AUF EINGABEN MIT GETKEY	28
LISTING 2.13 BEISPIEL FÜR INPUT.ACCELERATION	28
LISTING 2.14 INSTANZERZEUGUNG ÜBER CODE	31
LISTING 2.15 LOGGING ÜBER ADB LOGCAT	34

VI | Abkürzungsverzeichnis_[A1] / Glossar

Abk.	Begriff	Erklärung
-	Controller	bezeichnet ein Eingabegerät für die Steuerung von Computerspielen, in diesem Fall sind damit Joysticks und Gamepads gemeint
-	Drift	sich aufaddierender 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

VII Danksagung

X

VII Danksagung

1

Einleitung

[A2]

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.

Diese 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 auf der einen Seite fähig sein ein Gefühl der Immersion zu erzeugen, aber auch 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. Des Weiteren 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. Dieses 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^[A3]

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 im 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 vier 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. Des Weiteren findet eine Bewertung anhand von bestimmten Kriterien statt, um einen geeigneten Controller zu evaluieren.

Der dritte 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. Des Weiteren 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 in einer neuen Umgebung zu sein. 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 Mensch 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. Ein 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 und nur durch visuelle Stimuli Wahrnehmung von Eigenbewegung hervorrufen.

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, werden entsprechende Ausgabegeräte benötigt. Diese müssen so gebaut sein, dass eine möglichst hohe Immersion erreicht wird und der Nutzer sich präsent in der VR fühlt. Des Weiteren müssen diese 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 zusammengesetzte. 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].

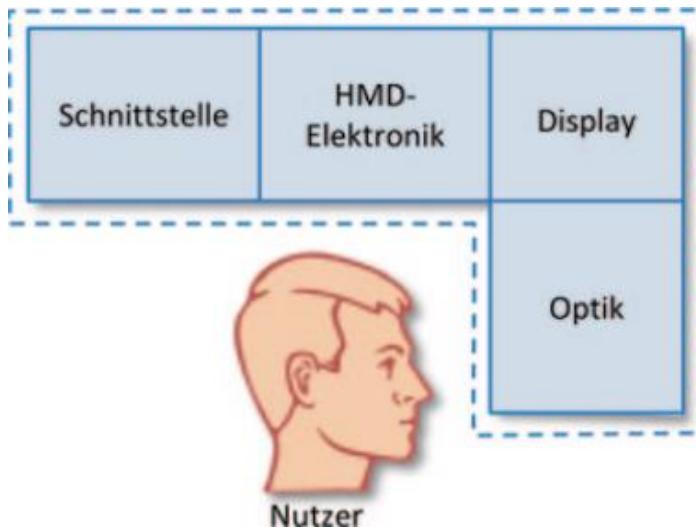


Abb. 2.1 Prinzipielle Bestandteile eines HMDs [Dör13]

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 werden 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 werden, 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 in 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].

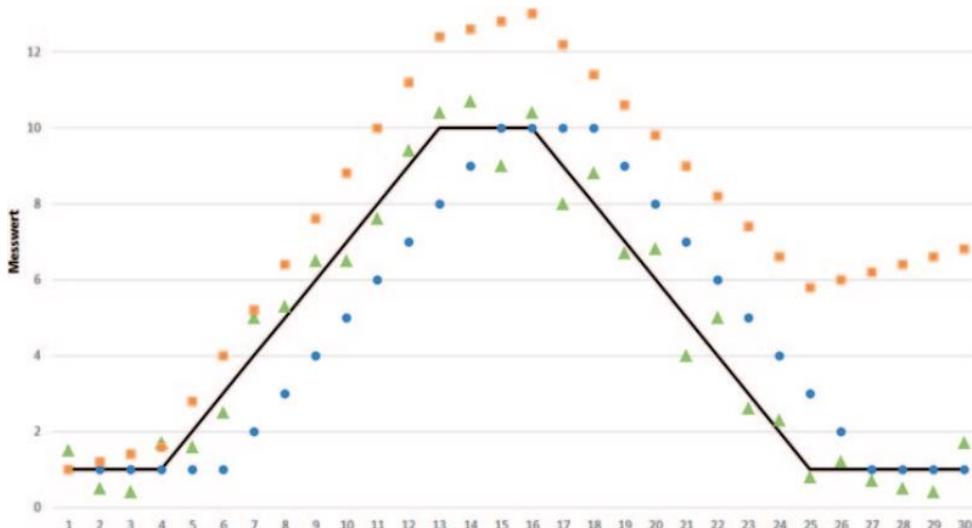


Abb. 2.2 Mögliche Fehler bei der Datenaufnahme der Position eines bewegten Objektes (schwarze Linie): Aufnahme mit Latenz (blau), mit Drift (Orange), mit Rauschen (grün) dargestellt über die Zeit (horizontale Achse) [Dör13]

Äußere Rahmenbedingungen

Äußere Rahmenbedingungen wie Licht, Temperatur oder die Möblierung eines Raumes, können je nach Eingabegerät Einfluss haben auf die Funktionalität dessen. 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 verfolgte reale 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. Des Weiteren sind über Funk verbundene Eingabegeräte benutzerfreundlicher als über Kabel verbundene. Bei optischen Verfahren, die den Nutzer über Kameras verfolgen ist der Interaktionsradius bestimmend für die Usability [Dör13].

2.1.4 Bewegungskontrolle

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

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, 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].



Abb. 2.3 Ansicht von Kinect und Leap Motion

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 omnidirektionale 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].

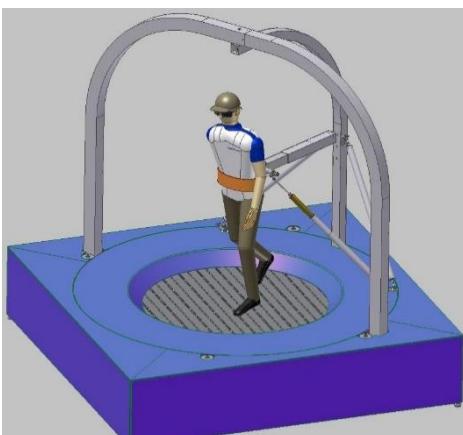


Abb. 2.4 ODT mit Halterung für den Nutzer

Akustisches Tracking

Das akustische Tracking nutzt Unterschiede in der Laufzeit oder Phase von Schallwellen. Dabei wird Ultraschall verwendet und mithilfe eines Senders, der am zu überwachenden Objekt angebracht ist, und Empfängers wird eine Abstandsbestimmung durchgeführt. 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 an den günstigen Anschaffungskosten. Diese stehen aber einem großen Nachteil gegenüber. Dieser 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 Inertialsensoren 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 Inertialsensoren 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, die später genauer behandelt werden.

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].

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].

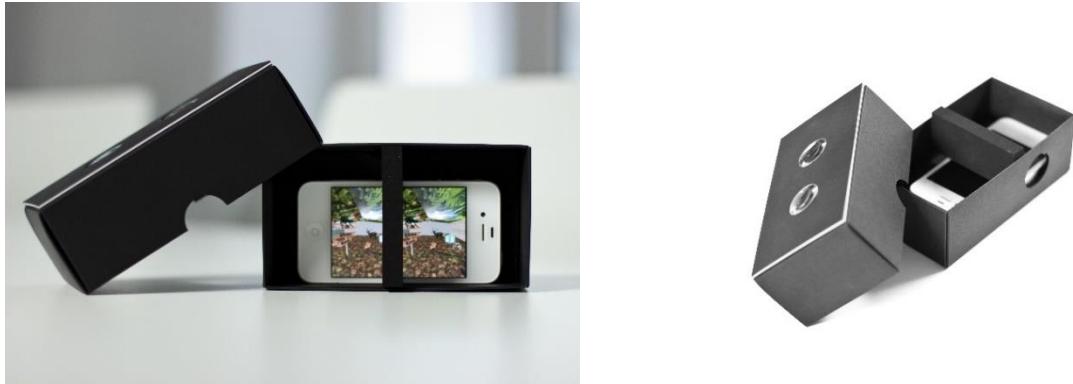


Abb. 2.5 Verschiedene Ansichten der Boxx3D

2.2.2 Google Cardboard

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

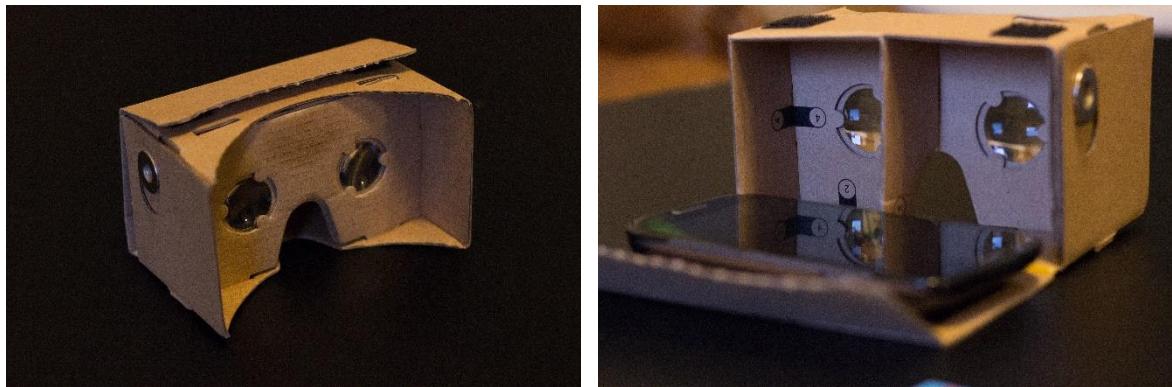


Abb. 2.6 Verschiedene Ansichten der Cardbox

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. Des Weiteren sind die Kosten 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].



Abb. 2.7 Ansicht der Gear VR

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 die Anwendung zu ändern [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 entstanden ist 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 werden und dadurch erst genutzt werden können. Jedes Skript muss von der Klasse MonoBehaviour erben und die Funktionalität eines Skripts als Komponente ermöglicht. Skripte lassen 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 kommt 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 einenm 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

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].

```
public class ComponentExample : MonoBehaviour {

    public GameObject user;

}
```

Listing 2.3 Inspector Zuweisung einer public Variablen



Abb. 2.8 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. Erstes 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());
}
```

Des Weiteren gibt es verzögerte Funktionsaufrufe, die über den Invoke-Befehl erreicht werden. Hier wird eine Funktion und eine 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 2.5 Beispiel 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 2.6 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 muss die zuvor gespeicherte ID übergeben werden [Sei14].

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

Listing 2.7 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 können auch eigene Meldungen ausgegeben werden, um z.B. bestimmte Ausgaben von Funktionen zu überprüfen. Die Varianten.LogError, LogException und.LogWarning geben die entsprechenden Meldung aus und werden zusätzlich noch entsprechend gekennzeichnet [Sei14].

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

Listing 2.8 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 werden. 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 2.9 Beispiele für die Erzeugung eines Vektors

Viele einzelne Vektoren zusammen ergeben dann ein 3D-Modell. Diese einzelnen Vektoren werden auch Vertices genannt und drei beieinander liegenden Vertices, die eine dreieckige Fläche ergeben, werden Polygone genannt. 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. Alle Komponenten können per Code verändert und angepasst werden [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 werden 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. Dies Komponente ermöglicht es der Kamera zu sehen und besitzt verschiedene Parameter. Die für das erzeugen einer VR-Kamera benötigten 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 werden 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. 2.9 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 auch 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 Tasten, 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 2.10 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 wird.
- > **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 2.11 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.GetKey(„a“))
    autoFire = true;
...
```

Listing 2.12 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]. So kann ein Objekt mit einem Beschleunigungssensor gesteuert werden, was im folgenden Beispiel zu sehen ist.

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

Listing 2.13 Beispiel für `Input.acceleration`

Objekte die dieses Skript angehängt bekommen, können durch den Beschleunigungssensor in x-Richtung verschoben werden.

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 werden. 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 zwei dimensionales Rechteck, was sich je nach Render Mode selbstständig dem Bildschirm des Ausgabegerätes anpassen kann.

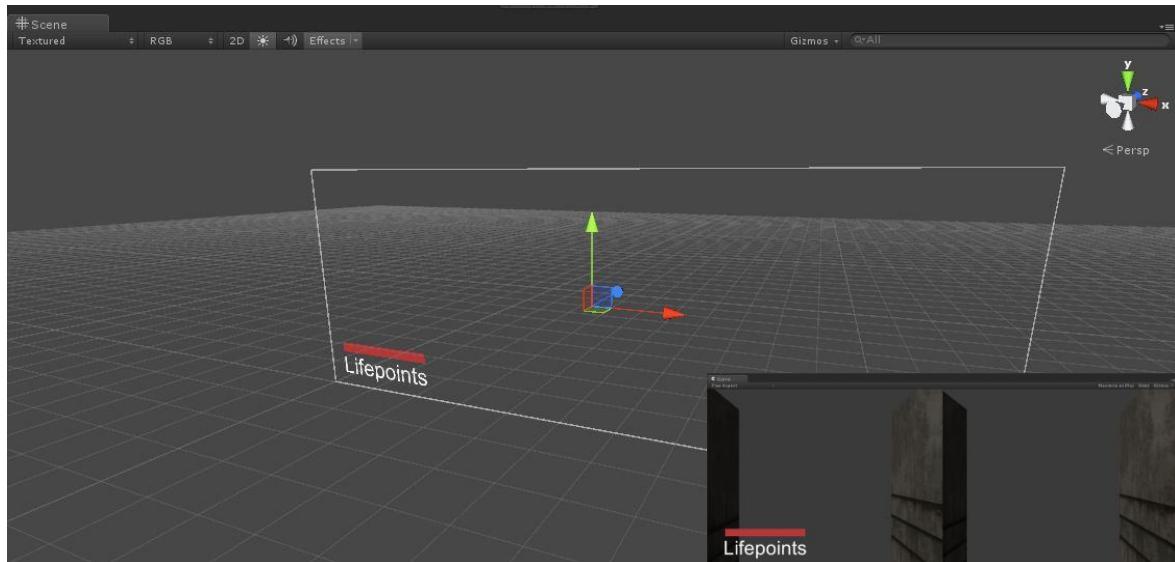


Abb. 2.10 Beispiel einer GUI im Scene und Game View

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 können GUI-Elemente platziert werden. 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. Ein 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 genutzt werden können.

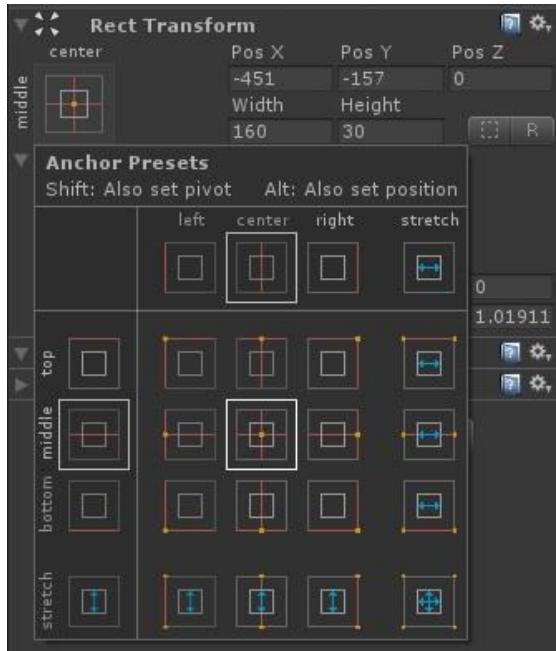


Abb. 2.11 Anchor-Vorgaben

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 können dann auf zwei Arten erstellt werden. 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 werden 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 2.14 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 für den Austausch von Elementen zwischen verschiedenen Projekten verwendet werden. So kann z.B. eine konfigurierte Spielerfigur von einem Projekt in das andere übernommen werden und an dieser Stelle dann für das Projekt angepasst werden. Im Folgenden sollen zwei genutzte UnityPackages kurz vorgestellt werden [Sei14].

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 Cardboard. 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 Mono-Develop auf die Ausführung auf dem Gerät reagiert und der Debugger gestartet wird [Sei14].

Des Weiteren 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].

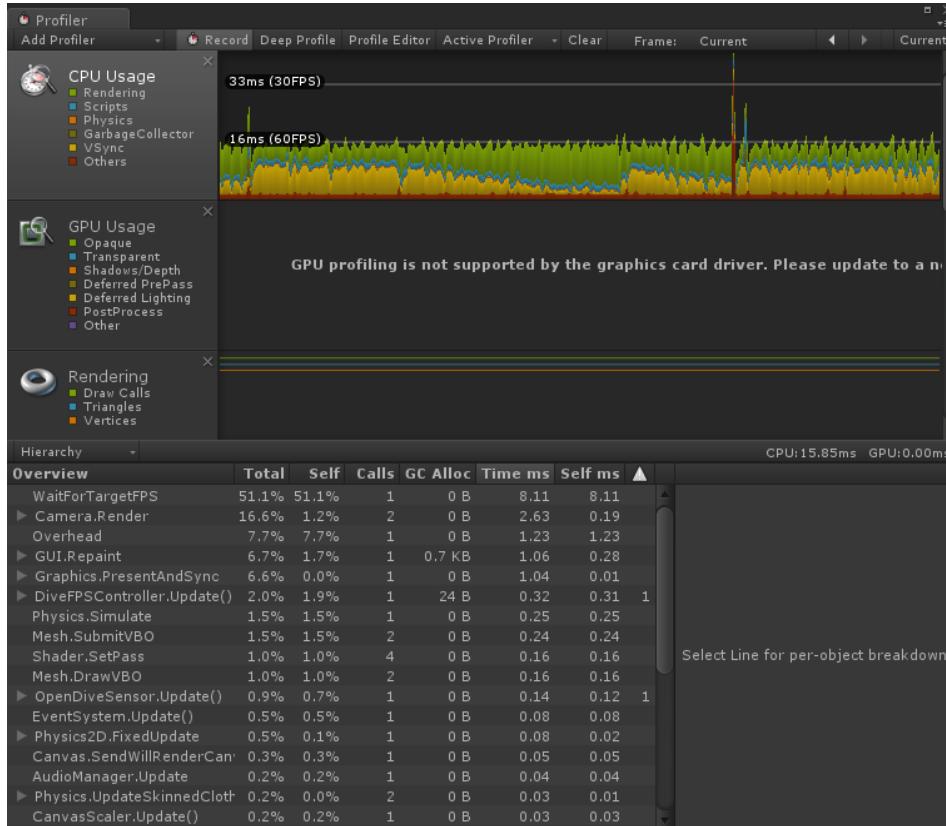


Abb. 2.12 Ansicht des Profilers: Darstellung der Performance eines Tesgerätes

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. Dieses 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 2.15 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. Hier können Einstellungen bezüglich des Zielbetriebssystems gemacht werden. Bei mobilen Geräten muss dieses per USB an den PC angeschlossen sein [Sei14].

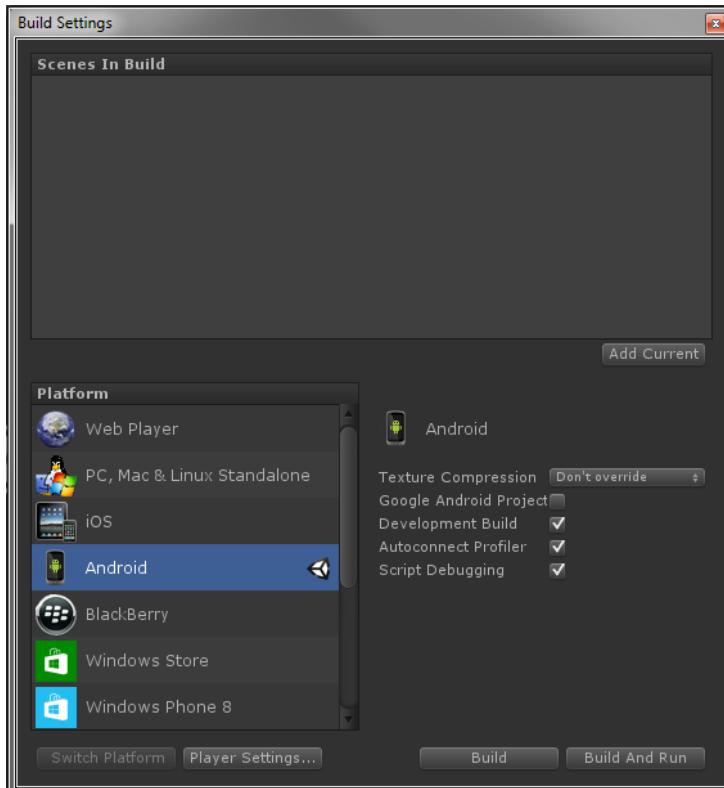


Abb. 2.13 Ansicht der Build Settings für Android

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, muss Build and Run betätigt werden. 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. Des Weiteren 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 Unternehmen	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

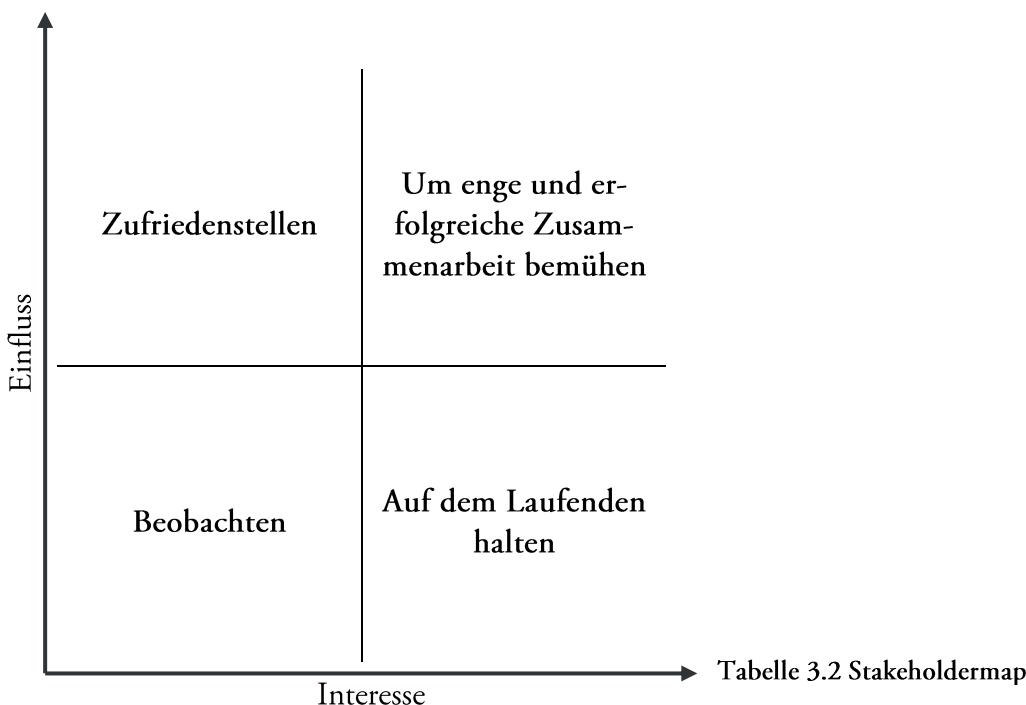


Tabelle 3.2 Stakeholdermap

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. Daher ist der Betreuer an der Hochschule **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. Es wird weiterhin eine weitere Unterteilung in App und Controller 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 durch Sprünge und ducken
 - W2.2 Kabellose Steuerung
 - W2.3 Steuerung ohne Hände möglich

Tabelle 3.4 Wunschkriterien

3.4 Systemkontext

3.5 Systemabgrenzung

3.6 Funktionale Anforderungen

3.6.1 Use-Case

Bild

3.6.2 Anforderungen

3.7 Nicht Funktionale Anforderungen

3.7.1 Technologisch

3.7.2 Benutzeroberfläche

3.7.3 Qualität

3.7.4 Durchzuführende Tätigkeiten

3.7.5 Rechtlich-vertraglich

3.7.6 Hier fehlt noch eine

3.8 Tests

In diesem Abschnitt werden die Testgeräte und mögliche Testszenarien festgehalten, um die Funktionalität eines Controllers zu testen.

3.8.1 Testgeräte

Als Testgeräte werden aktuelle und ältere Android-Smartphones genutzt. Um alle möglichen Controller testen zu können, werden Geräte verschiedener Hersteller verwendet.

Testgerät	Android-Version	Jahr
Samsung Galaxy S4	4.2.2	2013
Sony Xperia Z3	4.4.4	2014

Tabelle 3.5 Übersicht der Testgeräte

3.8.2 Testszenarien

T1 VR-App

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

T2 Controller

- T2.1 Verbindungsabbruch
- T2.2 Reichweite

Tabelle 3.6 Testszenarien

4

Analyse vorhandener Controller

Aus Nutzersicht wäre ein markenloses Outside-In-verfahren natürlich wünschenswert, da hierbei die Einschränkungen für den Nutzer am geringsten sind. Nutzer müssen nichts in den Händen halten, brauchen keine Markierungen auf der Kleidung und können sich frei bewegen und auch frei durch den Raum gehen. In der Praxis zeigt sich allerdings, dass markenlose Tracking-Systeme gegenüber markenbasierten zum einen anfälliger gegenüber Störungen (z. B. weitere Personen im Raum oder sich wechselnde Lichtverhältnisse) sind und zum anderen, dass die Genauigkeit bei markenbasierten Systemen höher ist.

Auch weisen Outside-In-verfahren den Nachteil auf, dass die Interaktionsfläche durch die Kamerapositionen begrenzt ist. Der Interaktionsbereich kann zwar durch den Einsatz von zusätzlichen Kameras vergrößert werden, dennoch bedeutet ein größerer Interaktionsraum entweder Mehrkosten (für die zusätzlichen Kameras) oder durch den größeren Abstand zu den Kameras eine größere Ungenauigkeit.

4.1 Verfügbare Controller

Vorstellung von Controllern Verschiedene Arten

Kinect, tango, analog, stem, cardboard,

4.1.1 Veränderung des Magnetfeldes

4.1.2 Gyroskop

4.1.3 Gamepad

4.1.4 Kamera die Bewegungen aufnimmt kinect und tango

4.1.5 ODT Virtuix Omni

5 Bewertung

Bewertungsskala erstellen, Punktesystem aufstellen danach dann Controller auswählen

Bewertungsskala anhand der Grundlagen aus [Dör13], Freiheitsgrad etc.

5.1 Mögliche Ansätze

5.2 Theoretische Ansätze

5.3 Auswahl

6 Implementation

6.1 GUI

6.2 Verbindung des Motion Controllers

6.3 Auslesen der Daten

6.4 Szene

6.5 Spiellogik

7 Tests

7.1 Schnelle Bewegungen

7.2 Bewegungsräume bei falscher Handhabung

7.3 Erfahrungen

8 Fazit und Ausblick

8.1 Bewertung

8.1.1 Controller

8.1.2 Applikation

asdasd

8.2 Ausblick

8.2.1 Prototypen von Controllern, was kommt ist in Arbeit

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
- [Fin13] T. Finnegan: „Unity Android Game Development by Example Beginner’s Guide“, 1. Auflage, Packt Publishing, Birmingham, Dezember 2013
- [Rup14] 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>
- [@Ext] Git Extensions Manual, <https://git-extensions-documentation.readthedocs.org/en/latest/>
- [@Gear] Gear VR, <http://www.samsung.com/global/microsite/gearvr/index.html>
- [@Git] Git Documentation, <http://git-scm.com/doc>
- [@Hub] GitHub, <https://github.com/>
- [@Leap] Leap Motion Forum, <https://forums.leapmotion.com/>
- [@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>
- [@Rud] 3DRudder, <http://www.3drudder.com/>
- [@Sph] Sphero, <http://www.gosphero.com/de/>
- [@Tan] Project Tango, <https://www.google.com/atap/projecttango/#project>

[@Uni] Unity Documentation, <http://unity3d.com/learn/documentation>

[@Six] Sixense STEM System, <http://sixense.com/wireless>

Bildquellen

[Com] Wikimedia Commons: Eine freie Sammlung von Bildern, www.commons.wikimedia.org

[Bus] Bilder von Tobias Busch

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
\Quellen	*.pdf	benutzte Internetseiten
\Bilder	*.jpg, *.png	verwendete Bilder in größerem Format
\unity-project	*.* \builds*.apk	enthält Testapplikation, die mit Unity erstellt wurde
\Videos	*.mp4	Videos der getesteten Controller

Tabelle B.1 Inhalt der CD

Erklärung

Hiermit versichere ich, dass ich meinen Projektbericht selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Datum:
.....

(Unterschrift)