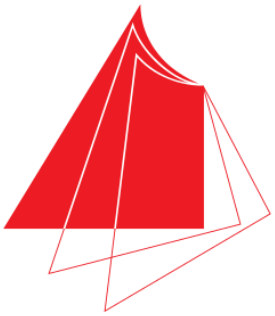

Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Fakultät für Informationsmanagement und Medien
Studiengang Kommunikation und Medienmanagement

Bachelor-Thesis

Konzeption und Umsetzung einer Virtual-Reality-Anwendung als interaktives Informationsmedium über den Einfluss des Fleischkonsums auf den Klimawandel

---

Bearbeitungszeitraum:
26.04.2019 – 26.08.2019

Erstprüfer:	Prof. Dipl.-Ing. Martin Schober
Zweitprüferin:	Prof. Dipl.-Inf. Sissi Closs

Verfasser:
Maurice Daum

Abstract

Im Rahmen dieser Bachelorthesis wurde eine Virtual-Reality-Anwendung entwickelt, die als interaktives Aufklärungsspiel auf die Folgen übermäßigen Fleischkonsums auf den Klimawandel aufmerksam machen soll. In der Anwendung „Climeat Change“ taucht der Nutzer in eine virtuelle Welt ein, die ihm die Auswirkungen seines Fleischkonsums auf das Klima sowohl interaktiv als auch informativ näherbringen soll. Unter dem Leitmotiv Präsenz wurde eine Anwendung konzipiert, deren Anspruch es ist, mittels Empathie und Emotionen ein Nutzererlebnis zu schaffen, das nachhaltig im Gedächtnis bestehen bleibt und im besten Fall Einfluss auf die Konsumgewohnheiten des Nutzers nimmt.

Der theoretische Teil dieser Arbeit gliedert sich in drei Bestandteile. Zunächst werden alle relevanten theoretischen Hintergründe zum Thema aufbereitet und vorgestellt. Auf Basis der Theorie, insbesondere unter der Berücksichtigung des Leitmotivs Präsenz, wird anschließend der konzeptionelle Entscheidungsfindungsprozess näher erläutert und die daraus folgenden Entscheidungen begründet. Abschließend wird das konkrete Vorgehen zur Umsetzung des Konzepts eingehend beschrieben.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Kontext

1.1.1 Fleischkonsum und Klimawandel

„Vier Fünftel der weltweit landwirtschaftlich genutzten Flächen werden heute allein von der Tierhaltung beansprucht. Das ist rund ein Drittel der gesamten Landoberfläche der Erde.“ (WWF Deutschland 2014a:5). Hintergrund dieser massiven Landnahme ist der hohe Konsum an tierischen Produkten in Industrie- und Schwellenländern. Der Großteil der genutzten Flächen entfällt dabei auf Ackerflächen für die Futtermittelproduktion. Gerade in den Tropen muss aus diesem Grund immer mehr Bewaldung für die Erschließung neuer Agrarflächen weichen (vgl. WWF Deutschland 2014b:9). Aus klimatischer Sicht führt dies zu einem unumkehrbaren Teufelskreis. Durch Brandrodung werden die nährstoffarmen Regenwaldböden zu fruchtbaren Ackerflächen für die Futtermittelproduktion umgewandelt (vgl. WWF International 2007:4f). Zu Gunsten der Konsumgewohnheiten unserer Gesellschaft wird die Umwelt also ihrer natürlichen CO^2-Speicher beraubt, bei gleichzeitiger Freisetzung klimaschädlicher Treibhausgase. Nach einigen Jahren der Nutzung sind die Nährstoffreservoirs der Böden schließlich verbraucht, es verbleibt eine nicht mehr nutzbare Brache (vgl. Ulin 2010).
Durch den Transport um die ganze Welt wächst der CO^2-Fußabdruck der Futtermittel weiter an, ehe sie unter anderem in deutschen Viehzuchtbetrieben landen (vgl. Müller-Lindenlauf 2012:29). Vor allem bei der Verfütterung an Rinder wird durch deren Verdauung Methan frei - ein weiteres Treibhausgas, das die Umwelt belastet (vgl. Umweltbundesamt 2019). Die Konversion von naturbelassenen Flächen in Agrarflächen und das bei der Verdauung freigesetzte Methan stellen jedoch nur zwei klimaschädliche Aspekte einer langen Kette dar. Emissionen entstehen beispielsweise auch beim Düngen der Futtermittel sowie bei Produktion, Verarbeitung, Verpackung, Lagerung, Transport, Zubereitung und Entsorgung von tierischen Nahrungsmitteln (vgl. WWF Deutschland 2012:18). Die Erzeugerbetriebe scheinen hauptverantwortlich für die fehlende Nachhaltigkeit in der Fleischproduktion zu sein. Jeder Einzelne bestimmt jedoch über seinen Konsum, die Nachfrage nach Fleisch und damit indirekt auch über die Art der Produktion von Fleischwaren. Ein gesamtgesellschaftliches Umdenken in Belangen des Konsums tierischer Produkte könnte langfristig also zu einer nachhaltigeren Produktion ebendieser führen.

1.1.2 Virtual Reality und Aufklärungsarbeit

Virtual Reality (VR) findet von Jahr zu Jahr neue Anwendungsszenarien. Auch wenn der Hauptabsatzmarkt für die Branche die Gaming-Industrie ist, hat sich die Technologie mittlerweile auf verschiedene andere Bereiche ausgeweitet. So auch als Medium für Aufklärungsarbeit und Fundraising. Eine Vielzahl an Projekten von Hochschulen und Non-Profit-Organisationen, die VR gezielt zur Aufklärungsarbeit nutzen, bestätigt dies (vgl. Wohlert 2018). Die Besonderheit von Virtual Reality liegt darin, den Nutzer in andere Welten eintauchen zu lassen. Der Prozess des Eintauchens in die virtuelle Welt nennt sich fachsprachlich Immersion. Gelingt die Immersion in besonderem Maße, so spricht man von der Präsenz des Nutzers in der virtuellen Welt. Der Nutzer kann dann,

zumindest, was seine Wahrnehmung betrifft, nicht mehr zwischen Realität und virtueller Realität unterscheiden. (vgl. Dörner et al. 2014:46). Der Zustand der Präsenz in einer virtuellen Umgebung schafft Nutzererfahrungen, die bei gelungenen VR-Projekten sogar Emotionen und Empathie auslösen können. Das Erleben reiner Fiktion kann so eine Wirkung bis in die Realität des Nutzers hinein entfalten (vgl. Bailenson 2018). Im Kontext zum Thema Fleischkonsum und dessen Auswirkungen auf den Klimawandel wäre eine nachhaltige Einflussnahme auf das Konsumverhalten des Nutzers erstrebenswert.

1.2 Zielsetzung

Ziel des praktischen Teils dieser Bachelorarbeit ist die Konzeption und Umsetzung einer interaktiven Virtual-Reality-Anwendung für die Aufklärungsarbeit zum Thema „Auswirkungen des Fleischkonsums auf den Klimawandel“. Unter dem Leitmotiv Präsenz soll dabei ein realistisches und emotionales Erlebnis für den Nutzer geschaffen werden. Je besser dessen Präsenzgefühl nach dem Spielen der Anwendung ist, desto eindrücklicher verbleibt die Nutzererfahrung im Gedächtnis. Über die Präsenz kann neben einer rein informativen auch eine empathische Basis geschaffen werden, die den Nutzer nachhaltig in seinem Denken und Handeln beeinflussen kann. Vor allem virtuelle Menschen und damit verbundene soziale Interaktionen sind dazu in der Lage. Virtuellen Menschen kommt bei der Konzeption und Umsetzung daher eine besondere Bedeutung zu. Daneben spielen aber auch zwei weitere Aspekte eine zentrale Rolle: Zum einen wird ein zum Thema passendes Narrativ benötigt, das den Nutzer auf eine informative, spannende und in sich schlüssige Reise mitnimmt, zum anderen müssen bei der Konzeption jene Interaktions- und Fortbewegungstechniken ausgewählt werden, die einen möglichst natürlichen und damit präsenzsteigernden Charakter besitzen. Über die Präsenz hinaus gilt es bei der Konzeption einer Virtual-Reality-Anwendung aber auch, die Möglichkeit zur technischen Umsetzung der geforderten Inhalte zu berücksichtigen. Die Auswahl von Software und Hardware muss daher in den konzeptionellen Rahmen miteinbezogen und bei allen Folgeentscheidungen berücksichtigt werden.
Prognosen über den Umfang der am Ende umgesetzten Inhalte zu treffen, ist unter den gegebenen Umständen schwer. Es besteht kaum Vorerfahrung mit Unity und keinerlei Erfahrung in der Entwicklung von Virtual-Reality-Anwendungen. Zentraler Anspruch soll zumindest die Umsetzung der Techniken zur Interaktion und Fortbewegung sowie die Implementierung von realistischen virtuellen Menschen sein. Alle weiteren Inhalte ergeben sich aus der schrittweisen Implementierung des konzipierten Spielverlaufs innerhalb des zeitlich gegebenen Rahmens.

2 Hintergründe und Begriffsdefinition

Im Folgenden werden Begriff und historische Hintergründe von Virtual Reality näher erläutert.

2.1 Virtual Reality

2.1.1 Geschichte

Virtual Reality ist eine Technologie, die ihre Präsenz im Bewusstsein der Gesellschaft erst in den letzten fünf bis zehn Jahre erlangt hat. Die Geschichte dieser Technologie reicht jedoch wesentlich weiter in die Vergangenheit:

1935

1935 liefert Stanley G. Weinbaum eine erste Beschreibung des heutigen Konzepts von Virtual Reality in seiner Science-Fiction-Kurzgeschichte „Pygmalion's Spectacles". Darin wird unter anderem ein System mit Brillen beschrieben, das holografische Aufzeichnungen von fiktiven Ereignissen inklusive Tast-, Geruchs- und Geschmackssinn wiedergibt (vgl. Neeb 2017).

1962

1962 stellt Morton Heilig den ersten Virtual-Reality-Automaten der Welt vor. „Sensorama" nennt er sein etwa telefonzellengroßes Gerät, das mittels Rüttelmechanik, stereoskopischen Bildern sowie einem Geruchs- und Windsystem bis zu fünf verschiedene Szenarien erlebbar macht. Unter anderem eine Motorradfahrt durch die Straßen von Brooklyn (vgl. Steinicke 2016:27).

1968

1968, sechs Jahre später, folgt die Erfindung des ersten Head-Mounted-Displays durch Ivan Edward Sutherland und seinen Studenten Bob Sproul (vgl. Steinicke 2016:27):

> *„Aufgrund seines enormen Gewichts und der Notwendigkeit Kopfbewegungen zu verfolgen, musste es an einem mechanischen Arm befestigt werden, der an der Decke von Sutherlands Labor hing. Das bedrohliche Aussehen inspirierte auch seinen Namen: „The Sword of Damocles".* (Neeb 2017)

> *„Ein im Raum schwebender Drahtgitterwürfel mit etwa fünf Centimeter [sic] Kantenlänge war das erste Objekt, welches in der virtuellen Realität dargestellt wurde."* (Hertel 2017)

1970er

Bedingt durch leistungsschwache Computer stagniert die Entwicklung der Virtual-Reality-Technologie in den Folgejahren jedoch. Erst Mitte der 1970er Jahre bringt der amerikanische Computerkünstler Myron Krueger mit seinem Labor „Videoplace" neue Fortschritte (vgl. Steinicke 2016:28). „Der Videoplace war ein großer Raum voller Bildschirme, deren Videos auf die Handlungen des Nutzers reagieren, ohne dass dieser selbst eine Brille oder andere Eingabegeräte benötigt." (Neeb 2017).

1982

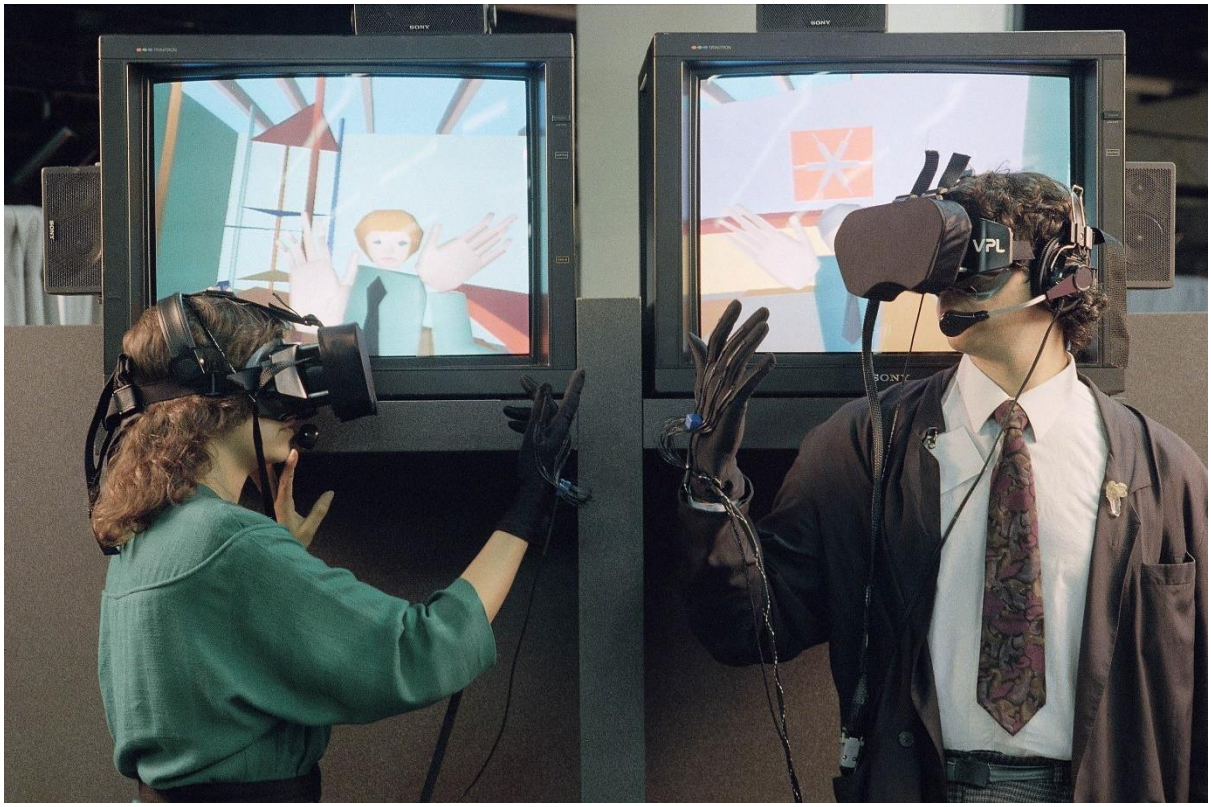

Abb. 1: *Showcase des EyePhone inklusive DataGlove auf der Texpo Telecommunications Show, 1989 in San Francisco. (Quelle: Sorene 2014)*

1982 wird dann „der Begriff Virtual Reality in seiner heutigen Bedeutung zum ersten Mal in dem Roman „The Judas Mandala“ von Damien Francis Broderick erwähnt.“ (Neeb 2017). Im selben Jahr eröffnet das Videospielunternehmen Atari Inc. ein Forschungslabor für Virtual Reality, muss das Projekt aber aufgrund der damaligen Krise in der Videospielindustrie nach zwei Jahren wieder abbrechen (vgl. Johnson o.J.). Der am Projekt beteiligte Jaron Lanier gründet daraufhin das Unternehmen VPL Research zur Entwicklung und Vermarktung von Virtual-Reality-Anwendungen. In dieser Zeit entwickelt er im Auftrag der NASA unter anderem den „DataGlove“ (vgl. *Abb. 1*), „ein Gerät, das man über die Hand zieht, um damit Aktionen im Computer auszulösen und nach virtuellen Gegenständen zu greifen.“ (Neeb 2017). In den Folgejahren wird Lanier zum Pionier auf dem Gebiet der Virtual Reality und trägt deutlich zur Popularisierung der Technologie bei (vgl. Steinicke 2016:29).

1990er

In den 1990er Jahren kommt Virtual Reality schließlich in der Videospielbranche an. Gleich mehrere Virtual-Reality-Systeme wie beispielsweise die Headsets „Sega VR“ und „Virtuality“ machen in diesem Jahrzehnt auf sich aufmerksam, scheitern jedoch alle an hohen Anschaffungskosten und dem Phänomen, das heute als sogenannte Cybersickness bekannt ist. Niedrige Auflösungen und Bildwiederholungsraten rufen bei den damaligen Nutzern immer wieder Kopfschmerzen und Übelkeit hervor (vgl. Neeb 2017).
Neues auf dem Gebiet Virtual Reality liefert damals auch das System „Cave Automatic Virtual Environment“, das von dem Kunstprofessor Daniel Sandin und den Informatikern

Tom DeFanti und Carolina Cruz-Neira entwickelt wurde. In dem System werden auf Wände, Decke und Boden eines Raumes 3D-Bilder projiziert. Auf diese Weise kann eine virtuelle Umgebung mit hohem Immersionsgrad geschaffen werden (vgl. Hertel 2017).

2000er

Während der 2000er Jahre stagnieren die Innovationen in der Virtual-Reality-Technologie. Deutliche Fortschritte werden in dieser Zeit aber im Bereich der Eingabegeräte unabhängig der VR-Technologie erzielt. Microsoft und Sony liefern sich mit ihren Produkten „Kinect“ und „Playstation Eye“ bzw. dessen Folgeprodukt „Playstation Camera“ gegen Ende des Jahrzehnts einen Konkurrenzkampf um sensorbasierte Eingabesysteme. Beide Systeme sind dazu in der Lage Bewegungsabläufe und Stimmen zu erkennen. Die Sensoren der Playstation Camera finden auch heute noch Anwendung im Anwendertracking des VR-Headset „Playstation VR“ (vgl. Hertel 2017). Die Computertechnik im Allgemeinen macht in diesem Zeitraum außerdem rasante Fortschritte: Geräte werden kleiner und immer leistungsfähiger, die ersten Smartphones kommen auf den Markt, und die Digitalisierung der Welt nimmt rasant an Fahrt auf (vgl. Steinicke 2016:31).

2012-2015

2012 gelingt den Entwicklern John Carmack und Palmyer Luckey schließlich der entscheidende Durchbruch, der die VR-Technologie der breiten Gesellschaft zugänglich macht. Carmack entwickelt eine „selbstgebastelte VR-Brille, die bei einem Komponentenpreis von 500 Dollar die Performance von professionellen HMD’s erreichen soll, die sonst 10.000 Dollar oder mehr kosten.“ (Hertel 2017). Als er schließlich von der Gründung der Firma Oculus durch Palmyer Luckey erfährt, stellt Carmack sein eigenes Produkt ein und arbeitet künftig mit Luckey zusammen. Auf der Crowdfunding Plattform Kickstarter sammelt das Unternehmen 2,5 Millionen Dollar für die Entwicklung einer kostengünstigen aber hoch performanten VR-Brille für den PC. Noch im selben Jahr stellt das Unternehmen sein mit 300 Dollar erschwingliches „Development Kit“ vor. Durch den Erfolg der Oculus Crowdfunding-Kampagne steigen nun etliche Start-ups und Großkonzerne wie Google und Facebook in die VR-Branche ein und bereiten damit den Weg für die Renaissance der Virtual-Reality-Technologie. Facebook erwirbt im folgenden Jahr die Firma Oculus für 2,3 Milliarden Dollar. Sowohl Sony als auch Oculus präsentieren schließlich Prototypen ihrer eigenen Head-Mounted-Displays. 2015 trumpft der Hardwarehersteller HTC in Kooperation mit dem Steam-Betreiber Valve mit der Vorstellung ihrer eigenen Virtual-Reality-Brille der „HTC Vive“ auf. Mit der sogenannten Lighthouse-Technologie wird erstmals auch ein Tracking-System implementiert, das die Position des Nutzers im Raum bis auf wenige Millimeter genau berechnen kann. Außerdem enthält das System Virtual-Reality-Controller, die fortan eine Interaktion mit der virtuellen Welt ermöglichen (vgl. Hertel 2017).

Der VR-Technologie geht eine lange Geschichte an konzeptionellen Ideen und technologischen Entwicklungen voraus, die die Technologie zu dem gemacht haben, was sie heute ist. Durch den Einstieg von Großkonzernen wie Facebook, Google, Sony, HTC und Microsoft erlebt die Branche heute eine unglaublich rasante Entwicklung und fortwährende Innovation. Es bleibt abzuwarten, wohin und wie schnell sich Virtual Reality in den nächsten Jahrzehnten entwickeln wird.

2.1.2 Begriffsdefinition

Wie die geschichtliche Entwicklung der VR-Technologie zeigt, ist Virtual Reality ein noch relativ „junges Wissenschaftsgebiet, dessen Weiterentwicklung u.a. stark von rasanten Fortschritten bei der zugrundeliegenden Hardware getrieben wird." (Dörner et al. 2014:12) Die Wissenschaft ist daher aktuell nicht in der Lage, sich auf eine einheitliche Definition des Begriffs zu einigen. Konsens besteht aber in einer technologieorientierten Charakterisierung der Virtual Reality (vgl. Dörner et al. 2014:12):

> *„The ultimate display would, of course, be a room within which the computer can control the existence of matter. A chair displayed in such a room would be good enough to sit in. Handcuffs displayed in such a room would be confining, and a bullet displayed in such a room would be fatal. With appropriate programming such a display could literally be the Wonderland into which Alice walked."*
> (Sutherland 1965 zit. nach Dörner et al. 2014:12)

Einer der ersten Pioniere der VR-Technologie beschrieb Virtual Reality also schon anhand seiner technologischen Aspekte. Die damals noch vage Vorstellung von Virtual Reality forderte schon „computer" und „display" als Ausgabegeräte zur Darstellung der virtuellen Umgebung. Auch nachfolgende Definitionsversuche beschränkten sich auf eine Charakterisierung von VR nach technologischen Maßstäben:

> „*Virtual Reality is electronic simulations of environments experienced via head mounted eye goggles and wired clothing enabling the end user to interact in realistic three-dimensional situations."* (Coates 1992, zit. nach Steuer 1992:5)

> *"Virtual Reality is an alternate world filled with computer-generated images that respond to human movements. These simulated environments are usually visited with the aid of an expensive data suit which features stereophonic video goggles and fiber-optic data gloves."* (Greenbaum 1992, zit. nach Steuer 1992:5)

> *"The terms virtual worlds, virtual cockpits, and virtual workstations were used to describe specific projects.... In 1989, Jaron Lanier, CEO of VPL, coined the term virtual reality to bring all of the virtual projects under a single rubric. The term therefore typically refers to three-dimensional realities implemented with stereo viewing goggles and reality gloves."* (Krueger 1991, zit. nach Steuer 1992:5)

Steuer und Dörner erkennen jedoch unabhängig voneinander, dass bei technologiebasierten Definitionsansätzen die Gefahr besteht, „dass sich die Definitionen der VR zu sehr auf konkrete Ein- und Ausgabegeräte […] beziehen, welche durch technologischen Fortschritt rasch überholt werden." (Dörner et al. 2014:13). Dörner fordert in Bezug auf technologieorientierte Begriffsdefinitionen von VR daher zukunftssichere Definitionen, die auch mit visionären Vorstellungen wie Sutherlands „Ultimate Display" oder dem „Holodeck" aus Star Trek vereinbar seien (vgl. Dörner et al. 2014:13). Als Beispiele führt er folgende Definitionen aus den frühen Jahren der Virtual Reality an, die auch auf heutige Systeme noch zutreffen:

> *„Virtual Reality (VR) refers to the use of three-dimensional displays and interaction devices to explore real-time computer-generated environments."*
> (Bryson 1993, zit. nach Dörner et al. 2014:13)

> *„Virtual Reality refers to immersive, interactive, multi-sensory, viewer-centered, three-dimensional computer generated environments and the combination of technologies required to build these environments."*
> (Cruz-Neira 1993, zit. nach Dörner et al. 2014:13)

Sowohl Steuer als auch Dörner liefern deswegen eine andere Herangehensweisen an die Begriffsdefinition von Virtual Reality. Es geht dabei um die Betrachtung von Virtual Reality als Erfahrung oder Erlebnis und deren Definition darüber:

> *„Im Zentrum der VR steht eine Erfahrung – die Erfahrung in einer Virtuellen Welt oder an einem fremden Ort zu sein."*
> (Rheingold 1992 zit. nach Dörner et al. 2014:17)

Bei dieser Betrachtungsweise misst sich Virtual Reality nicht mehr an der zugrundeliegenden Technologie, sondern vielmehr an den Sinneseindrücken ihrer Nutzer. In der perfekten virtuellen Realität nähmen diese ein Niveau an, das in Qualität und Quantität dem aus der realen Welt entspräche. Der Nutzer wäre nicht mehr dazu in der Lage zwischen Realität und virtueller Realität zu unterscheiden (vgl. Dörner 2014:18). Präsenz ist dabei das zentrale Konzept zur Beschreibung der mentalen Aspekte der VR-Erfahrung:

> *„Presence can be thought of as the experience of one's physical environment; it refers not to one's surroundings as they exist in the physical world, but to the perception of those surroundings as mediated by both automatic and controlled mental processes.*"
> (Steuer 1992:5)

Virtual Reality definiert sich in diesem Ansatz also über die Wahrnehmung der Nutzer und lässt sich über das Messinstrument Präsenz qualitativ und quantitativ bewerten.

Im Kontext dieser Thesis wird Virtual Reality aus beiden der oben vorgestellten Perspektiven betrachtet. Bei der Konzeption der Anwendung aus narrativer und interaktiver Sicht stand stets die Nutzererfahrung und damit die Präsenz im Vordergrund. Bei der Entwicklung einer VR-Anwendung für ein spezifisches Hardwaresystem spielt aber auch die technische Betrachtung von Hardware und Software eine entscheidende Rolle. Das Präsenzgefühl des Nutzers steht außerdem in unmittelbarem Kontext zu verschiedenen Hardwareparametern – Diese Fälle heben die Bedeutung der Betrachtung beider Ansätze im gegenseitigen Kontext nochmals hervor.

3 Menschliche Wahrnehmung und Virtual Reality

Die Präsenz spielt eine entscheidende Rolle bei der Definition von Virtual Reality. An der menschlichen Wahrnehmung lassen sich Präsenz und Immersion am besten messen und bewerten. Sowohl in der VR-Hardware- als auch in der VR-Softwareentwicklung ist daher ein Verständnis der menschlichen Informationsverarbeitung und der Funktionsweise der unterschiedlichen Sinneswahrnehmungen notwendig. Für Virtual Reality von zentraler Bedeutung sind dabei die visuelle, die akustische und die haptische Wahrnehmung des Menschen. Darüber hinaus existieren VR-typische Probleme und Phänomene, die sich nur mit dem Wissen um die menschliche Wahrnehmung erklären, diskutieren und lösen lassen (vgl. Dörner et al. 2014:33).
In den folgenden Kapiteln findet eine wissenschaftliche Betrachtung der menschlichen Informationsverarbeitung und Sinneswahrnehmung statt. Auf Grundlage dieser Betrachtung werden anschließend VR-typische Probleme und Phänomene wie Doppelbilder oder Cybersickness vorgestellt und diskutiert.

3.1 Menschliche Informationsverarbeitung

Menschen nehmen Informationen wahr und verarbeiten diese. Auch beim Eintauchen in eine virtuelle Welt ist der Mensch nicht dazu in der Lage, sich dem zu entziehen. Er ist unumgänglich dazu gezwungen seine Umgebung zu konsumieren, sei es im Virtuellen oder Realen (vgl. Dörner et al. 2014:34).

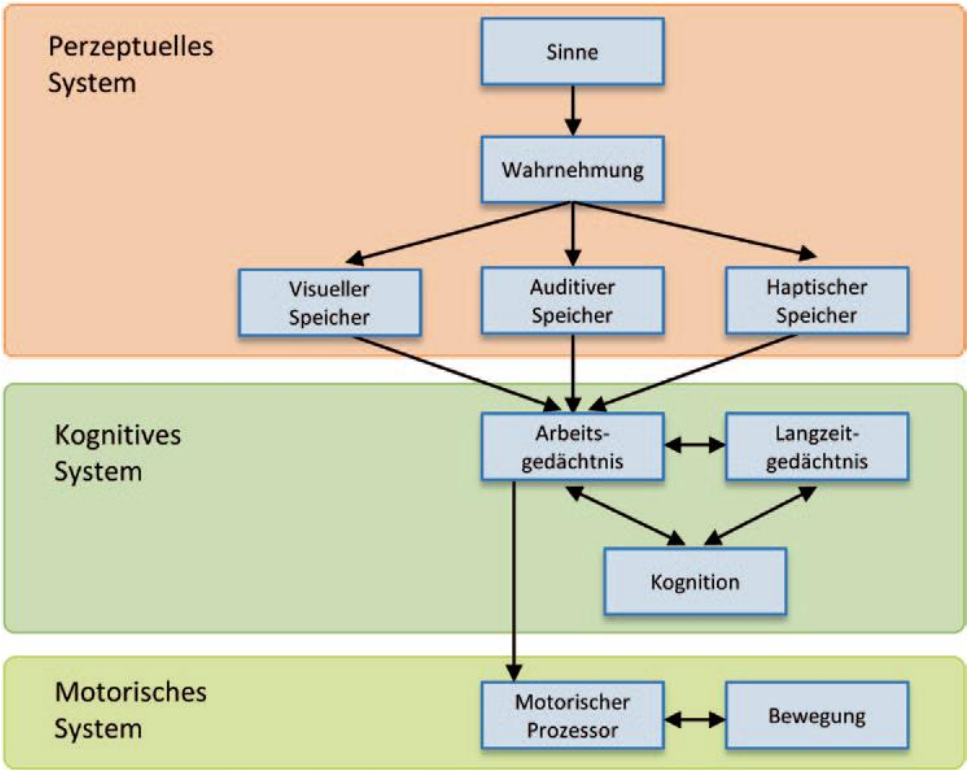

Abb. 2: *Modell der Menschlichen Informationsverarbeitung. (Quelle: Card et al. 1986 zit. nach Dörner et al. 2014:35)*

Um die komplexen Vorgänge bei der menschlichen Informationsverarbeitung besser verstehen zu können, ist es hilfreich, sich den Menschen als informationsverarbeitendes

System vorzustellen. Analog zur Datenverarbeitung eines Computers lässt sich dieses System beschreiben durch eine Eingabe, eine Verarbeitung und eine Ausgabe von Daten. Beim Mensch entspricht dies der Wahrnehmung durch die Sinnesorgane, der Verarbeitung durch Bewusstsein und Gedächtnis sowie der Motorik als Komponente zur Reaktion auf die verarbeiteten Reize. Zunächst gelangt ein Reiz über die Sinnesorgane in das sogenannte Perzeptuelle System, in dem die Wahrnehmung stattfindet. Abhängig vom angesprochenen Sinn werden die Informationen vorgefiltert und zwischengespeichert. Der wahrgenommene Reiz wird schließlich an das Kognitive System zur Verarbeitung weitergegeben. In diesem System können bereits bestehende Informationen aus dem Arbeitsgedächtnis und dem Langzeitgedächtnis abgerufen werden, um den neuen Reiz korrekt zu interpretieren und entsprechendes Handeln zu planen. Die tatsächliche Handlung wird schließlich im motorischen System durch den motorischen Prozessor umgesetzt und in Form von Bewegungen ausgegeben. (vgl. Dörner et al. 2014:34f & *Abb. 2*).

3.2 Visuelle Wahrnehmung

Der für die Informationsverarbeitung des Menschen relevanteste Parameter ist die visuelle Wahrnehmung. Bei der visuellen Wahrnehmung werden Lichtimpulse, die auf die Netzhaut im menschlichen Auge treffen in Nervensignale umgewandelt und im Gehirn schließlich zu einem Bild zusammengesetzt (vgl. Heinicke 2012:54).

3.2.1 Funktionsweise des Auges

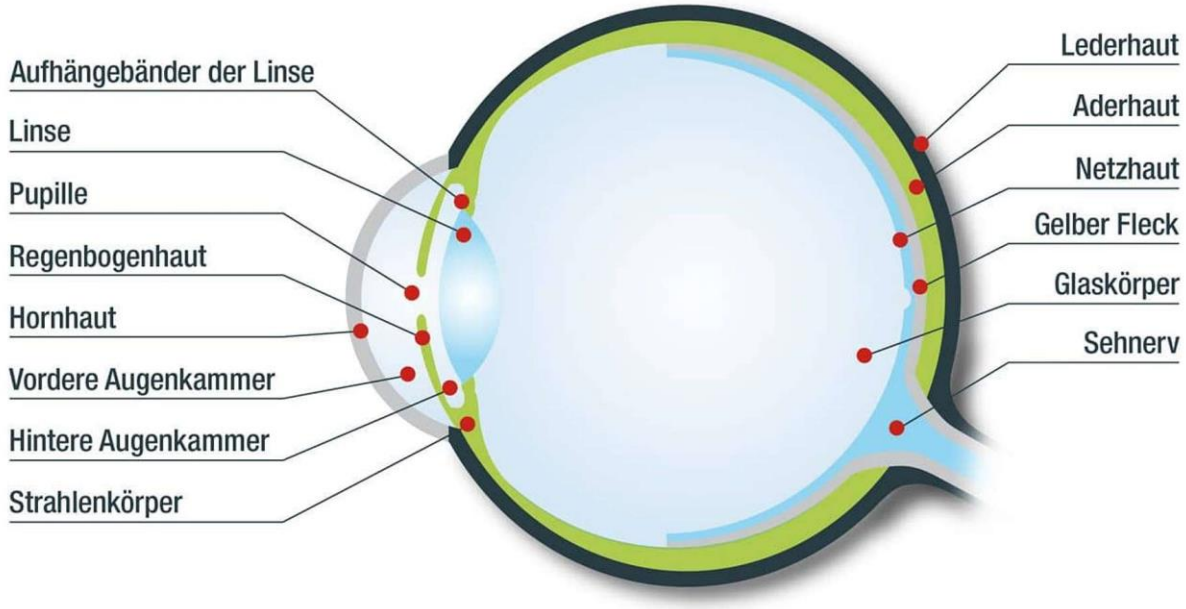

Abb. 3: *Das Auge – die Anatomie des Auges. (Quelle: Kuratorium Gutes Sehen e.V. o.J.)*

Um mit unseren Augen Objekte wahrnehmen zu können, müssen die Objekte entweder sichtbares Licht reflektieren oder selbst eine Lichtquelle sein. Das über die Linse einfallende Licht wird dann auf unsere Netzhaut projiziert. Auf der menschlichen Netzhaut befinden sich etwa 120 Millionen Sehzellen. Diese unterteilen sich in sogenannte Stäbchen und Zapfen (vgl. Dörner et al. 2014:35f). Die Stäbchen sind für die

Hell-Dunkel-Wahrnehmung verantwortlich. Die Zapfen übernehmen die Funktion der Farbwahrnehmung und unterteilen sich nochmals in drei Subtypen zur Wahrnehmung der Farbtöne Rot, Grün und Blau. Weitere Farbeindrücke entstehen durch unterschiedliche Intensitäten des einfallenden Lichts und damit durch eine unterschiedliche Reizung der einzelnen Zapfentypen. Das Gehirn ermittelt aus den empfangenen Reizen schließlich den jeweiligen Farbeindruck. In ihrer Gesamtheit sind die Zapfen dazu in der Lage auf Licht in einem Wellenlängenbereich von etwa 400 bis etwa 700 Nanometer zu reagieren. Dies entspricht dem Farbspektrum des sichtbaren Lichts (vgl. Heinicke 2012:55). Das Licht, das über die Linse auf die Netzhaut fällt erzeugt dort ein auf dem Kopf stehendes und gespiegeltes Bild. Damit dieses Bild scharf auf der Netzhaut ankommt, muss die Linse mithilfe des sogenannten Ziliarmuskels abhängig von der Entfernung eines betrachteten Objekts richtig eingestellt werden. Dieser Vorgang wird Akkommodation genannt (vgl. Grasnick 2016:26). Auf der Netzhaut befindet sich eine Stelle der höchsten Abbildungsschärfe und der größten Dichte an Sehzellen. Diese Stelle misst in ihrer Größe lediglich die Hälfte eines Stecknadelkopfs und wird Fovea genannt. Trotz eines Öffnungswinkels des Auges von rund 150° werden nur zwei bis drei Grad des Sichtfeldes auf die Fovea projiziert. Damit der Mensch dennoch dazu in der Lage ist ein vollständig hochaufgelöstes Bild wahrzunehmen, verweilt unser Auge nie länger als 250 ms bis eine Sekunde auf einem einzelnen Punkt. Es treten vielmehr sogenannte Sakkaden auf, während derer das Auge rasche, ruckartige Bewegungen vollzieht. Diese Bewegungen ergänzen die periphere Wahrnehmung, in der die Auflösung lediglich ein Vierzigstel der fovealen Auflösung entspricht (vgl. Dörner et al. 2014:36).
Die bisherige Betrachtungsweise beschränkt sich lediglich auf die Wahrnehmung einzelner Bilder, das sogenannte statische Sehen. Das menschliche Auge ist aber auch dazu in der Lage, Bewegungen wahrzunehmen. Voraussetzung für die Wahrnehmung einer Bewegung ist die aufeinanderfolgende Stimulation mehrerer Sehzellen. Dieses Phänomen nennt sich retinale Bildverschiebung. Ähnlich wie beim Farbensehen entsteht der Bewegungseindruck schließlich im Gehirn. Dort sitzen spezielle Nervenzellen, die dazu in der Lage sind, die Bewegungsinformationen, die vom Auge kommen, zu interpretieren. Bei der Analyse des Bewegungsreizes findet eine Berücksichtigung körpereigener Bewegungen statt, damit retinale Bildverschiebungen, die beispielsweise durch Körperbewegungen bedingt sind, nicht als Bewegung wahrgenommen werden. (vgl. Groß 2017).

3.2.2 Stereosehen

Das Stereosehen (Stereopsis) ist ein wichtiger Bestandteil der menschlichen Wahrnehmung und von besonderer Relevanz für die Entwicklung von Virtual-Reality-Systemen. Mithilfe des Stereosehens ist der Mensch dazu in der Lage, einen dreidimensionalen Eindruck von seiner Umgebung zu erhalten, obwohl die Lichtreize nur im Zweidimensionalen auf der Netzhaut abgebildet werden. Außerdem sorgt das Stereosehen für den Eindruck eines einzelnen wahrgenommenen Bildes, statt eigentlich zweier Bilder wie sie jeweils von den Augen an das Gehirn übermittelt werden (vgl. Dörner et al. 2014:36).

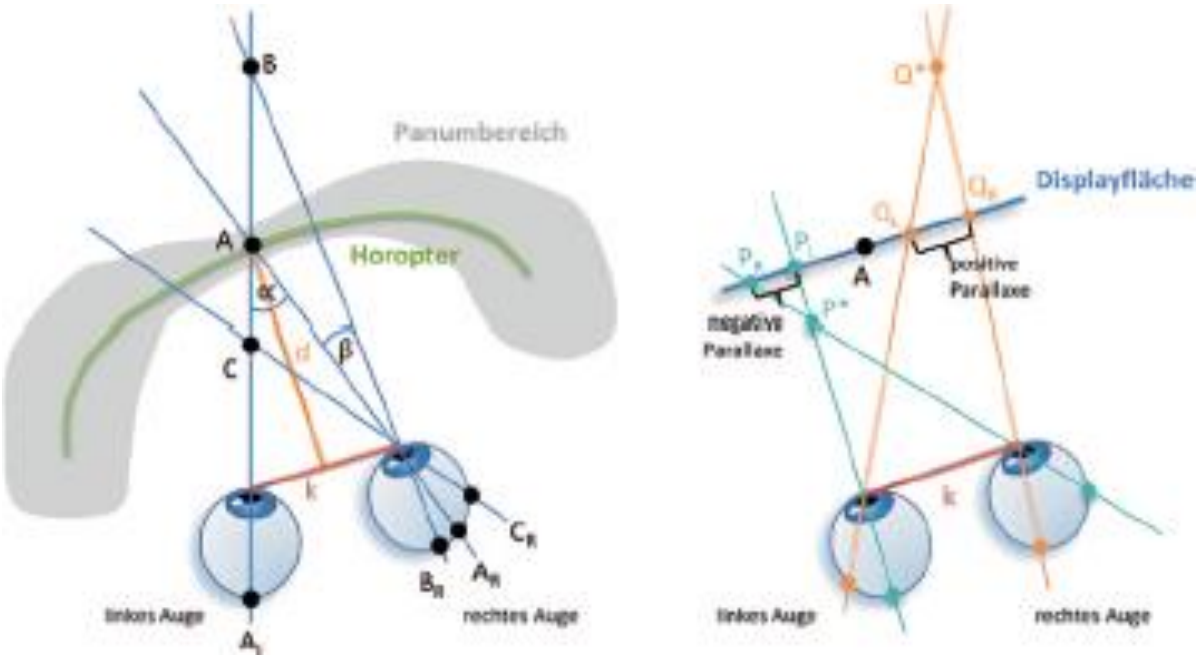

Abb. 4: *links: Stereopsis, rechts: Manipulation der Stereopsis mit einem Stereodisplay (Quelle: Dörner et al. 2014:37)*

In *Abb. 4* fixieren die Augen den Punkt A, was bedeutet, dass das Licht dieses Punktes sowohl auf die Fovea des rechten (A_R) als auch auf die des linken Auges (A_L) fällt. Dazu müssen die Augen zunächst entsprechend ausgerichtet und die Linse entsprechend eingestellt werden. Je näher der fixierte Punkt am Betrachter liegt, desto stärker werden die Augen nach innen zur Nase gedreht. Die Bewegung, die die Augen dabei vollziehen, nennt man Konvergenz. Mithilfe der Konvergenz sowie des konstant bleibenden Augenabstands ist das Gehirn dazu in der Lage, die Entfernung zum Punkt A zu ermitteln. Bei einer gedachten Überlagerung beider Augen, lägen die Punkte A_L und A_R an der gleichen Stelle, sie korrespondieren. Die Fläche, auf der alle Punkte liegen, die bei der Fixierung eines Punktes A auf korrespondierenden Punkten auf der Netzhaut abgebildet werden, nennt man Horopter. Der Horopter ist also eine gekrümmte Fläche, auf der alle Punkte die gleiche Entfernung zum Auge besitzen. (vgl. Dörner et al. 2014:36). Die Punkte B und C in *Abb. 4* liegen nicht auf dem Horopter. Das Licht dieser Punkte trifft an nicht korrespondierenden Punkten auf die Netzhaut. Die Abweichung, die ein Punkt im Auge zum jeweils gedachten korrespondierenden Punkt annimmt, nennt man Disparität. Mithilfe dieser Disparität ist das Gehirn dazu in der Lage, Entfernungen von Punkten zu bestimmen, die im Gegensatz zum Punkt A nicht fokussiert werden und auch nicht auf dem Horopter liegen, sondern davor oder dahinter. Im menschlichen Gesichtsfeld existieren darüber hinaus Punkte, bei denen die Disparität so groß wird, dass das visuelle System nicht mehr dazu in der Lage ist, die Bildeindrücke beider Augen zu einem einzelnen Bildeindruck zu fusionieren. In diesem Fall entstehen sogenannte Doppelbilder. Alle Punkte im menschlichen Gesichtsfeld, bei denen keine Doppelbilder entstehen, liegen auf dem sogenannten Panumbereich. (vgl. Dörner et al. 2014:37f).

Stereodisplays

Trotz ihrer zweidimensionalen Darstellungsweise ist es auch mit Displays möglich, einen dreidimensionalen Eindruck zu erzeugen. Wie *Abb. 4* zeigt, fokussiert der Betrachter

dabei einen Punkt A, der auf der Displayfläche liegt. Auf dem Display werden weiterhin die Punkte P_L und P_R dargestellt. Bestimmte technische Vorkehrungen sorgen dafür, dass das Licht von P_R nur in das rechte Auge und das Licht von P_L nur in das linke fällt. Durch diesen besonderen technischen Aufbau wird das Gehirn getäuscht. Es setzt die Informationen des rechten und des linken Auges zusammen und geht von einem einzigen Punkt P* aus, der vor der Displayfläche liegen muss. Invertiert man den beschriebenen Aufbau, wie es bei den Punkten Q_R und Q_L der Fall ist, so lässt sich der scheinbare Punkt Q* hinter der Displayfläche erzeugen. Beim Betrachter entstehen so dreidimensionale Sinneseindrücke. (vgl. Dörner et al. 2014:38f).

3.2.3 Raumwahrnehmung

Bei etwa einem Fünftel der Weltbevölkerung ist das Gehirn zum Auswerten der Informationen, die sich aus den Disparitäten ergeben, gar nicht in der Lage. Diesen Teil der Bevölkerung bezeichnet man als stereoblind. Der betroffene Teil der Bevölkerung ist dennoch dazu fähig Räumlichkeit und Tiefe wahrzunehmen. Neben dem Stereosehen gibt es also noch andere Parameter, sogenannte Tiefenhinweise, die eine Raumwahrnehmung ermöglichen. Einer dieser Tiefenhinweise ist die sogenannte Verdeckung. Verdeckt ein Gegenstand A einen anderen Gegenstand B, so kann das visuelle System daraus ableiten, dass sich Gegenstand A vor Gegenstand B und näher am Betrachter befinden muss. Tiefenhinweise wie die Verdeckung, die aus der Information eines einzelnen Auges ermittelt werden können, nennt man monokulare Tiefenhinweise. Analog dazu stellt beispielsweise das Stereosehen einen binokularen Tiefenhinweis dar. Tab.: 1 zeigt eine Auflistung aller bekannten visuellen Tiefenhinweise und deren Klassifizierung als monokulare, binokulare oder dynamische Tiefenhinweise. Dynamisch bedeutet, dass sich der Tiefenhinweis aus einer Bewegung ergibt (vgl. Dörner et al. 2014:40f).

Tiefenhinweis	Wirkungsbereich	Klassifizierung
Verdeckung	Kompletter Bereich	Monokular
Disparität	Bis 10 m	Binokular
Konvergenz	Bis 2 m	Binokular
Akkomodation	Bis 2 m	Monokular
Image Blur	Kompletter Bereich	Monokular
Linearperspektive	Kompletter Bereich	Monokular
Texturgradient	Kompletter Bereich	Monokular
Relative Größe	Kompletter Bereich	Monokular
Bekannte Größe	Kompletter Bereich	Monokular
Atmosphärische Perspektive	Über 30 m	Monokular
Höhe im Gesichtsfeld	Über 30 m	Monokular
Shape from Shading	Kompletter Bereich	Monokular
Schattenwurf	Kompletter Bereich	Monokular
Bewegungsparallaxe	Über 20 m	Dynamisch
Accretion	Kompletter Bereich	Dynamisch

Tab.: 1 *Tiefenhinweise*

Tiefenhinweise sollten nie als unabhängige Parameter der Raumwahrnehmung betrachtet werden. Der Eindruck von Räumlichkeit entsteht durch das Zusammenspiel

aller existierenden Tiefenhinweise. Die Gewichtung eines Tiefenhinweises ist primär an die Entfernung gekoppelt und findet bei der Interpretation durch das Gehirn flexibel statt (vgl. Dörner et al. 2014:40). Sekundär fließt in die Gewichtung von Tiefenhinweisen aber auch die jeweilige Aufgabe mit ein, mit der der Betrachter befasst ist. Geht es beispielsweise darum Entfernungen einzuschätzen, so greift das Gehirn verstärkt auf die Tiefenhinweise Bewegungsparallaxe, Linearperspektive, Texturgradient und Schattenwurf zurück. Besteht die Aufgabe hingegen darin, ein Objekt zu greifen, dann nehmen Disparität, Konvergenz und Akkommodation den höheren Stellenwert bei der Gewichtung der Tiefenhinweise ein (vgl. Wanger/Ferwerda/Greenberg 1992:54f). Das menschliche Gehirn arbeitet also nicht mit einem einzelnen Modell zur Raumwahrnehmung, sondern mit mehreren unterschiedlichen aufgabenspezifischen Modellen. Ist die Raumwahrnehmung von besonderer Relevanz für eine VR-Anwendung, so gilt es dies im Kontext der Anwendungsentwicklung zu berücksichtigen (vgl. Dörner et al. 2014:42).

3.3 Auditive Wahrnehmung

Das menschliche Ohr ist dazu in der Lage, Schwingungen wahrzunehmen, die sich durch ein Medium wie Luft oder Wasser fortbewegen. Die Schwingungen erzeugen dabei Druckschwankungen im Medium, die durch die Ohren aufgenommen und in Nervensignale umgewandelt werden. (vgl. Dörner et al. 2014:43)

3.3.1 Funktionsweise des Ohrs

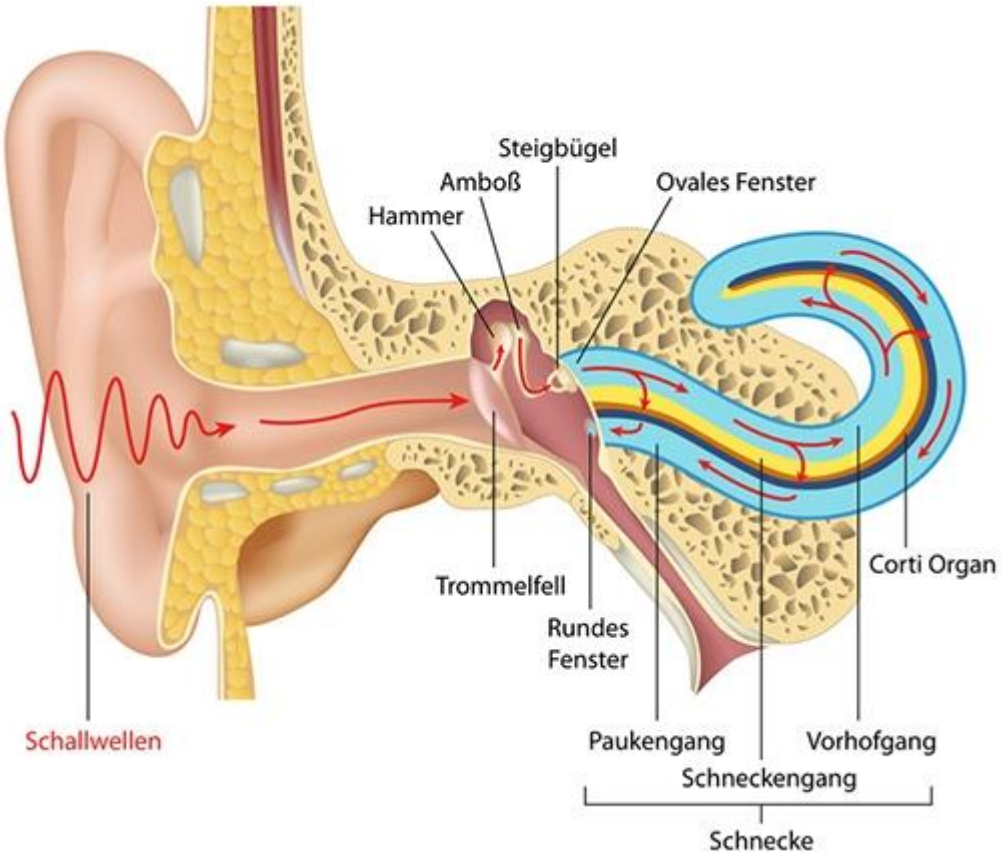

Abb. 5: *Aufbau des Ohrs (Quelle: Spektrum Hören o.J.)*

Die Ohrmuschel fängt Schwingungen in der Luft zunächst ein und leitet diese an das Mittelohr weiter. Im Mittelohr versetzen die Druckschwankungen das sogenannte Trommelfell in Schwingung, welches die Schwingungen schließlich mechanisch über die Gehörknöchelchen Hammer, Amboss und Steigbügel an das Innenohr abgibt. Das Innenohr besteht aus einem schneckenförmigen, flüssigkeitsgefüllten Organ, auch Cochlea genannt, das die Schwingungen vom Steigbügel aufnimmt. Diese breiten sich durch das flüssige Medium aus und stimulieren feine Haarzellen, die im Innern der Cochlea sitzen. Die Härchen erzeugen schließlich Nervensignale, die über den Hörnerv an das Gehirn weitergeleitet werden. Das menschliche Ohr ist dazu in der Lage Frequenzen zwischen etwa 0,016 kHz und 18 kHz wahrzunehmen (vgl. Dörner et al. 2014:43 & *Abb. 5).*

3.3.2 Richtungshören

Auch der menschliche Hörsinn ermöglicht die Wahrnehmung von Räumlichkeit durch sogenanntes Richtungshören oder auch räumliches Hören. Dabei nutzt das Gehirn drei

verschiedene Parameter zur Bestimmung der Richtung einer Schallquelle. Befindet sich ein Schallereignis beispielsweise rechts des Hörers, so erreicht dieser Schall das rechte Ohr des Hörers schneller, als das linke. Gleichzeitig verringert sich der Schalldruck bei der Passage des Kopfes vom rechten zum linken Ohr deutlich und zuletzt findet eine unterschiedliche Dämpfung von Tönen unterschiedlicher Frequenz statt, was dazu führt, dass sich auch die Klangfarbe zwischen der Wahrnehmung des rechten und des linken Ohres verändert. Die Differenzen in Klangfarbe, Schalldruck und Zeit nutzt das Gehirn, um die Richtung eines Schallereignisses zu interpretieren (vgl. Hanser 2005).

3.4 Haptische Wahrnehmung

> „*Haptik oder haptische Wahrnehmung beschreibt die sensorische und/oder motorische Aktivität, die das Erfühlen von Objekteigenschaften, wie beispielsweise Größe, Konturen, Oberflächentextur und Gewicht, durch Integration der in der Haut, in den Muskeln, Gelenken und Sehnen empfundenen Sinneseindrücke.*“
> (vgl. Hayward et al. 2004 zit. nach Dörner et al. 2014:44)

Die Sinne, die dabei zur haptischen Wahrnehmung beitragen gliedern sich in die taktile Wahrnehmung und die sogenannte Tiefensensibilität (vgl. Dörner et al. 2014:44).

3.4.1 Taktile Wahrnehmung

Die taktile Wahrnehmung ist verantwortlich für die Wahrnehmung von Berührungen, Wärme und Schmerz. Für jede dieser Wahrnehmungen existieren eigene Rezeptortypen, die auf und unter der Haut sitzen. Die Rezeptoren verteilen sich in unterschiedlicher Anzahl über den menschliche Körper. Am häufigsten finden sie sich an den Händen, den Lippen und auf der Zunge. Die für die taktile Wahrnehmung relevantesten Rezeptortypen sind die Mechanorezeptoren zur Wahrnehmung von Druck, Berührung und Vibration, die Thermorezeptoren zur Wahrnehmung von Wärme und Kälte, sowie die Nozizrezeptoren, die für das Schmerzempfinden verantwortlich sind (vgl. Dörner et al. 2014:44).

3.4.2 Tiefensensibilität

Die Tiefensensibilität gliedert sich in Propriozeption und Kinästhesie. Unter Propriozeption versteht man,

> *„alle Empfindungen, die mit der menschlichen Körperposition - sowohl in Ruhestellung als auch in Bewegung – zusammenhängen […] Propriozeption gibt uns also Informationen über die Position des Körpers im Raum und die Stellung der Gelenke und des Kopfes (Lagesinn), sowie Informationen über den Spannungszustand von Muskeln und Sehnen (Kraftsinn).“*
> (Dörner et al. 2014:44f).

Die Kinästhesie ermöglicht die körpereigene Bewegungsempfindung wie beispielsweise das Fühlen der Bewegungsrichtung der menschlichen Extremitäten oder der Gesamtbewegungsrichtung des Körpers (vgl. Dörner et al. 2014:45). Im Kontext von Virtual Reality ist die Tiefensensibilität von großer Bedeutung für die Interaktion in der virtuellen Welt. Eine Stimulation der Tiefensensibilität findet mithilfe von Controllern, Exoskeletten oder Bewegungsplattformen statt, die Bewegungen in der Realität in die virtuelle Realität transformieren (vgl. Dörner et al. 2014:45).

3.5 Multisensorische Wahrnehmung

Die zuvor separat vorgestellten Wahrnehmungsaspekte visuelle, auditive und haptische Wahrnehmung sind keineswegs als unabhängig arbeitende Systeme zu verstehen. Bei der Informationsverarbeitung greift das Gehirn zur Interpretation eines Ereignisses vielmehr auf alle ihm zur Verfügung stehenden Informationen zurück (vgl. *3.1 Menschliche Informationsverarbeitung*). Aus diesem Grund kann übergreifend auch von multisensorischer Wahrnehmung gesprochen werden (vgl. Dörner et al 2014:43).

3.5.1 Bewegungswahrnehmung

Die Bewegungswahrnehmung des Menschen stellt einen Aspekt von multisensorischer Wahrnehmung dar. Die visuelle Wahrnehmung nimmt dabei zwar die größte Rolle ein, das menschliche Gehirn greift aber auch auf den Lagesinn im Innenohr (vestibulärer Sinn) und die Tiefenwahrnehmung zurück, um neben der Bewegung anderer Objekte auch die Eigenbewegung des Körpers bei der Interpretation von Bewegungen zu berücksichtigen (vgl. Dörner et al. 2014:46).

3.5.2 Präsenz und Immersion

Wie schon im Kapitel *1.1.2 Virtual Reality und Aufklärungsarbeit* beschrieben, versteht man unter Immersion das Eintauchen eines Nutzers in die virtuelle Welt. Immersion beruht größtenteils auf der menschlichen Wahrnehmung und ist daher ein quantifizierbarer Prozess. Unter Präsenz versteht man den Zustand eines Nutzers in der virtuellen Welt, in dem er bezogen auf seine Wahrnehmung nicht mehr dazu in der Lage ist, zwischen Realität und virtueller Realität zu unterscheiden. Erreicht ein Nutzer den Zustand der Präsenz, so ist damit stets ein sehr hoher Immersionsgrad verbunden. In Bezug zur menschlichen Wahrnehmung lässt sich der Grad der Immersion daran messen, wie gut die verschiedenen Wahrnehmungsaspekte im Einzelnen aber auch multimodal, also in Form von multisensorischer Wahrnehmung, stimuliert werden (vgl. Dörner et al. 2014:46). Spezielle Fragebögen und das Erheben physiologischer Daten sind dabei gängige Instrumente zur Ermittlung des Immersionsgrads. Eine in der virtuellen Realität auftretende Gefahrensituation kann bei einem hohen Immersionsgrad beispielsweise zu entsprechenden messbaren körperlichen Reaktionen führen, wie einer erhöhten Hautleitfähigkeit oder einem Anstieg der Herzfrequenz (vgl. Slater/Usoh/Steed 1994:8ff).

Immersion und Präsenz als ganzheitliche Konzepte

Immersion und Präsenz werden im Kontext von Virtual Reality primär aus der wahrnehmungsbezogenen Perspektive betrachtet. Bücher und Filme beweisen jedoch regelmäßig, dass das Eintauchen in eine fiktive Welt auch ohne eine allumfassende Stimulation der Sinne möglich ist. 2004 führt Adams im Kontext der Spieleentwicklung drei weitere Typen von Immersion ein:

- Strategische Immersion
- Taktische Immersion
- Narrative Immersion

Unter strategischer Immersion versteht er die aktive Beteiligung des Nutzers am Spielgeschehen. Beispielsweise, indem man dem Spieler ein konkretes Ziel setzt, das er lösen muss, oder indem das Spiel und seine Umgebung subtile Anreize schaffen, die den Nutzer dazu bewegen, sich strategisch damit auseinanderzusetzen. Als taktische Immersion bezeichnet Adams das unmittelbare und intuitive Eintauchen in ein bestimmtes Spielgeschehen. Er beschreibt taktische Immersion als einen Zustand, in dem die Augen unmittelbar mit den Händen und Fingern kommunizieren, ohne dass sich der Nutzer Gedanken über sein Handeln macht. Taktische Immersion findet also ohne Reflexion statt – eine intuitive Spielumgebung, die schnelles Agieren und Reagieren erlaubt ist nach Adams die Basis für gelungene taktische Immersion. Unter narrativer Immersion versteht Adams zuletzt das Eintauchen des Nutzers in die Geschichte, wie es vergleichsweise auch beim Lesen von Büchern stattfindet. Durch ein gutes Narrativ solidarisiert sich der Leser mit seiner Umwelt und interagiert mit den Charakteren (vgl. Adams 2004).

3.6 Wahrnehmungsbezogene Phänomene und Probleme in Virtual Reality

Die Komplexität der menschlichen Wahrnehmung bedingt verschiedene Phänomene und Probleme, die beim Einsatz von Virtual-Reality-Technologien und beim Erleben virtueller Welten auftreten können. Die Relevantesten davon werden nachfolgend erläutert.

3.6.1 Abweichende Betrachtungsparameter

Es sei eine reale Umgebung in einer virtuellen Welt rekonstruiert und mit einer virtuellen Kamera auf einem Stereodisplay abgebildet. Im Idealfall ist der dabei erzeugte Bildeindruck bei einem Betrachter identisch zu dem, den die Realität erzeugen würde. Tatsächlich fließen aber immer sogenannte abweichende Betrachtungsparameter in den entstehenden Bildeindruck mit ein. Das Display erzeugt eine planare Bildebene, der Bildeindruck des Betrachters entsteht aber auf einer gekrümmten Netzhaut. Außerdem nimmt der Betrachter in den seltensten Fällen exakt die gleiche Position zum Bild ein, die der Kameraposition in der virtuellen Welt entspräche. In der Regel ist er näher oder weiter entfernt, schaut nicht senkrecht auf das Bild, sondern von der Seite. In der Folge entstehen Vergrößerungen, Verkleinerungen und Verzerrungen des Bildeindrucks. Dadurch wird die Entfernungseinschätzung oder auch die Wahrnehmung der Neigung von Objekten beeinflusst (vgl. Kuhl et al. 2005 zit. nach Dörner et al. 2014:47). Die beschriebenen Verzerrungen werden interessanterweise jedoch als wenig störend wahrgenommen. Man spricht auch von der Robustheit der linearen Perspektive (vgl. Kubovy 1986:41). Auch in einem Kino nehmen Menschen verschiedene Perspektiven zur Bildebene ein, stören sich aber nicht an verzerrten Bildeindrücken. Grund dafür ist, dass das visuelle System des Menschen dazu in der Lage ist, die abweichenden Betrachtungsparameter, mithilfe der Abweichung der Blickrichtung von der Normalen des betrachteten Punkts, zu korrigieren (vgl. Vishwanath/Girshick/Banks 2005:8).
Trotz des geringen Einflusses der abweichenden Betrachtungsparameter auf die Wahrnehmung, sollten sie im Kontext von Virtual Reality berücksichtigt werden. Je nach Szenario, wenn es beispielsweise um die Einschätzung von Entfernungen geht oder um die Aufgabe mit der Umgebung zu interagieren, können abweichende

Betrachtungsparameter den Nutzer in seiner Wahrnehmung stören. (vgl. Dörner et al. 2014:48).

3.6.2 Doppelbilder

Bei der Betrachtung eines Stereodisplays kann sogenannte Diplopie, das Sehen von Doppelbildern auftreten. Dies geschieht dann, wenn das visuelle System des Menschen nicht dazu in der Lage ist, die beiden separaten Bildeindrücke des rechten und linken Auges zu einem einzelnen Bildeindruck zu fusionieren. Da Diplopie den Immersionsgrad stark beeinträchtigt, gilt es, dieses Phänomen im Kontext von Virtual Reality zu vermeiden (vgl. Dörner et al. 2014:49).
Beim Tragen eines Head-Mounted-Displays fokussiert der Betrachter stets Punkte, die auf der Displayfläche des Stereodisplays liegen. Wie in Kapitel *3.2.1 Funktionsweise des Auges* bereits vorgestellt, berührt der sogenannte Horopter an diesen Punkten die Displayfläche. In bestimmten Abständen um die Fläche des Horopters herum liegt der Panumbereich. Dieser zeichnet sich dadurch aus, dass das Auge bei allen Punkten, die innerhalb dieses Bereichs liegen, noch dazu in der Lage ist, einen einzelnen Bildeindruck zu erzeugen. Bei allen Punkten außerhalb des Panumbereichs werden die Disparitäten so groß, dass zwei getrennte Bildeindrücke entstehen: Der Nutzer sieht Doppelbilder. Soll also eine virtuelle Welt mit Hilfe eines Stereodisplays dargestellt werden, so steht dafür nur ein begrenzter Bereich zur Verfügung, in dem die virtuellen Objekte vor oder hinter dem Display angeordnet werden können, ohne dass Diplopie auftritt (vgl. Dörner et al. 2014:49).

3.6.3 Frame Cancellation

Stereodisplays nutzen die Fähigkeit des menschlichen Stereosehens aus, um Objekte vor oder hinter die Displayfläche zu projizieren. Aus technischer Sicht weisen Stereodisplays aber noch Unvollkommenheiten auf, was zu Problemen bei der visuellen Wahrnehmung wie dem Problem der sogenannten Frame Cancellation führen kann. Projiziert ein Stereodisplay ein Objekt vor die Displayfläche, so rückt dieses Objekt, je größer der scheinbare Abstand zur Displayfläche wird, immer weiter an den Rand. Berührt dieses Objekt schließlich den Rand, so geht die Illusion, dass sich das Objekt vor der Displayfläche befindet, schlagartig verloren. In dem Moment, in dem das Objekt den Rand des Displays berührt, treten widersprüchliche Tiefenhinweise auf. Die Disparität sorgt dafür, dass das Objekt vor dem Display wahrgenommen wird, die Verdeckung des Objekts durch den Displayrand suggeriert dem Gehirn jedoch das Gegenteil. Die Verdeckung durch den Displayrand ist in diesem Fall der stärkere Tiefenhinweis – das Objekt wird nun also hinter dem Display wahrgenommen. Um das Phänomen der Frame Cancellation zu vermeiden, sollten Objekte, die vor die Displayfläche projiziert werden, entweder vom Rand ferngehalten oder am Rand abgedunkelt werden, um einen fließenden Übergang zu schaffen und den widersprüchlichen Tiefenhinweis der Verdeckung zu vermeiden (vgl. Mendiburu 2009 zit. nach Dörner et al. 2014:50f).

3.6.4 Vergenz-Fokus-Konflikt

Fokussiert das Auge einen bestimmten Punkt, so ermöglicht das Stereosehen mithilfe von Disparitäten, Objekte als vordergründig oder hintergründig zu diesem Punkt wahrzunehmen. In der Realität ist es einfach, den Fokus mittels Konvergenz der Augen

auf eines der im Vordergrund liegenden Objekte zu richten. Wird dieser Versuch aber beim Betrachten eines Stereodisplays unternommen, scheint das Objekt plötzlich unerwartet verschwommen. Der Fokus der Augen wandert in diesem Fall weg von der Displayfläche hin zum im Vordergrund projizierten Objekt. Die Bildinformationen werden jedoch weiterhin auf der Displayfläche erzeugt, was zu einer verschwommenen Wahrnehmung des Objekts führt. Konvergenz und Fokusinformationen stehen bei diesem Phänomen im Konflikt. Bei längerer Betrachtung von virtuellen Welten kann dieser Konflikt schließlich zu Ermüdung und Kopfschmerzen führen (vgl. Mon-Williams/Wann 1998:42ff). Durch das Einhalten eines geringen Abstands der projizierten Objekte zur Displayfläche lässt sich dieser Konflikt jedoch lösen (vgl. Hoffmann et al. 2008:18ff).

3.6.5 Diskrepanzen in der Raumwahrnehmung

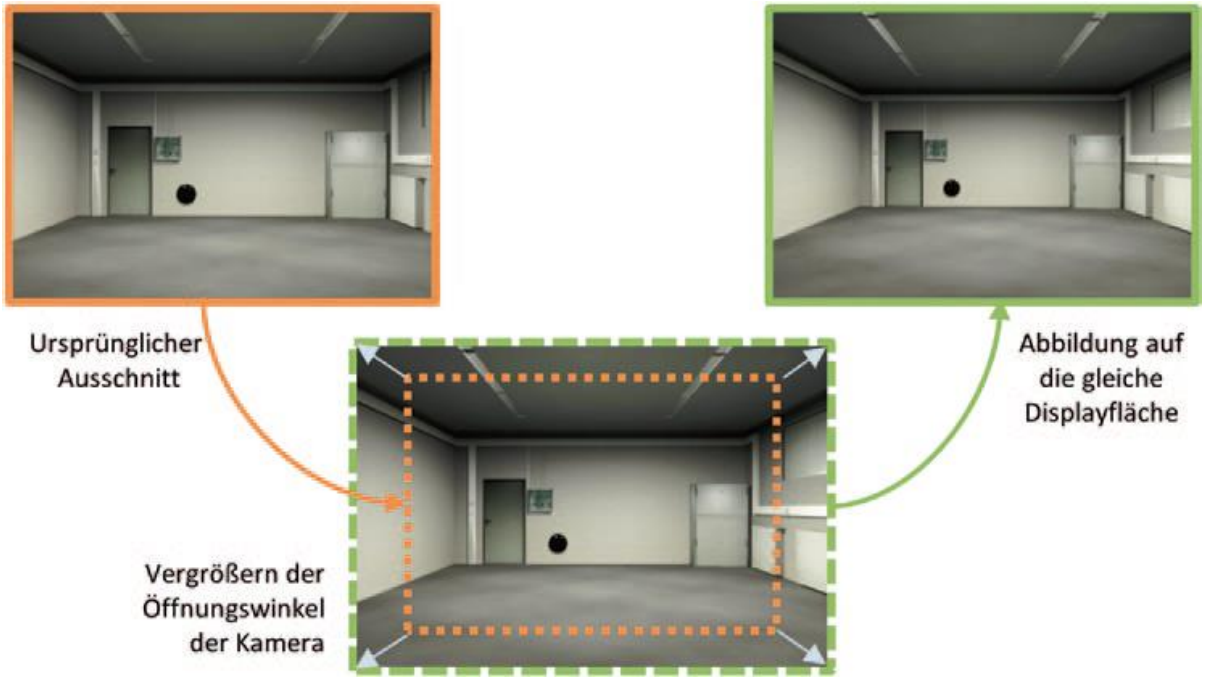

Abb. 6: *Darstellung des gleichen virtuellen Raums mit (links) kleinem und (rechts) großem geometrischen Sichtbereich (Quelle: Steinicke et al. 2009 zit. nach Dörner et al. 2014:54)*

Für viele Virtual-Reality-Anwendungen wie Flugsimulationen oder medizinische Trainings ist eine korrekte Einschätzungen von Distanzen durch den Nutzer unabdingbar. Trotzdem treten bei Nutzern virtueller Welten regelmäßig Diskrepanzen in der Raumwahrnehmung auf. Die Nutzer sind dann nicht mehr dazu in der Lage, Distanzen korrekt einzuschätzen. Dieses Phänomen wird zwar mit einigen Faktoren wie der stereoskopischen Darstellung, der Beschränkung des Sichtbereichs oder dem Realismus von Beleuchtung und Schattierung in Verbindung gebracht, wissenschaftlich ist das Phänomen aber noch nicht abschließend erklärbar. Versuche zu Diskrepanzen in der Raumwahrnehmung belegen jedoch, dass Nutzer in ihrer Distanzeinschätzung in virtuellen Welten um bis zu 50% abweichen können (vgl. Dörner et al. 2014:52f). Mithilfe verschiedener Ansätze lässt sich der Fehleinschätzung von Distanzen in virtuellen Welten aber entgegenwirken. Durch eine Vergrößerung des geometrischen Sichtbereichs, der durch den horizontalen und vertikalen Öffnungswinkel der virtuellen

Kamera definiert wird, sieht der Betrachter je nach Veränderung der Öffnungswinkel einen größeren oder kleineren Bereich der virtuellen Welt. Da sich die Größe des Displays dabei nicht verändert, werden auch die dargestellten Objekte als größer bzw. kleiner wahrgenommen. Dies wirkt sich unmittelbar auf die Distanzeinschätzung des Betrachters aus. (vgl. Dörner et al. 2014:53 & *Abb. 6*). Eine allgemeine Verbesserung des Präsenzgefühls des Nutzers trägt darüber hinaus signifikant zur Qualität der Distanzeinschätzung in virtuellen Welten bei (vgl. Interrante/Anderson/Ries 2006).

3.6.6 Diskrepanzen in der Bewegungswahrnehmung

Vergleichbar mit den Diskrepanzen der Raumwahrnehmung treten in virtuellen Welten häufig auch Diskrepanzen in der Bewegungswahrnehmung auf. Vor allem bei der Vorwärtsbewegung und Rotation des eigenen Körpers ist dies der Fall. Der Nutzer hat dann Probleme, seine eigene Geschwindigkeit korrekt zu interpretieren. Die gängigste Lösung für Diskrepanzen in der Bewegungswahrnehmung ist es, die Geschwindigkeit der getrackten Bewegungen in der virtuellen Welt zu erhöhen bzw. zu senken, um einen möglichst natürlichen Bewegungseindruck zu erzeugen. Durch die beschriebene Anpassung werden die in Kapitel *3.2.3 Raumwahrnehmung* vorgestellten dynamischen Tiefenhinweise jedoch verzerrt. In bestimmten Situationen, in denen der Nutzer auf diese Tiefenhinweise zurückgreift, können so wiederum Diskrepanzen in der Raumwahrnehmung auftreten. Vor der Anpassung der Geschwindigkeiten in der virtuellen Umgebung sollte daher stets eine Prüfung auf etwaige Nebeneffekte wie Diskrepanzen in der Raumwahrnehmung stattfinden. (vgl. Dörner et al. 2014:55f).

3.6.7 Cybersickness

Bei einer Vielzahl der Nutzer von Virtual-Reality-Anwendungen treten Symptome wie Übelkeit, erhöhter Speichelfluss, Benommenheit, Schwindelgefühle und im Extremfall sogar erbrechen auf. Das Phänomen in seiner Gesamtheit nennt sich Cybersickness. Heute geht man von drei unterschiedlichen Faktoren aus, die Cybersickness hervorrufen können (vgl. Steinicke 2016:48):

- Sensorische Konflikttheorie
- Gifttheorie
- Theorie der Haltungsinstabilität

Die sensorische Konflikttheorie erklärt Cybersickness durch einen Konflikt verschiedener Sinneswahrnehmungen: Wenn Informationen des visuellen Systems mit denen des vestibulären Systems (Lagesinn) in Widerspruch geraten, so reagiert der Körper mit Cybersickness. Aus der Realität ist dieses Phänomen auch bekannt als Reisekrankheit. In der virtuellen Welt kann ein derartiger Konflikt auf zwei unterschiedliche Weisen entstehen. Zum einen, wenn kein Headtracking erfolgt, die virtuelle Umgebung aber trotzdem ihre Blickrichtung verändert oder sich bewegt. In diesem Fall liefert das visuelle System einen Bewegungseindruck, während das vestibuläre System dieser Information widerspricht. Es liegt ein sensorischer Konflikt vor und der Körper reagiert mit den oben genannten Symptomen der Cybersickness. Im zweiten Fall werden die Kopfbewegungen zwar mittels Headtracking überwacht, die Latenz zwischen den unmittelbaren Eindrücken des vestibulären Systems und den

nachfolgenden Bildeindrücken des visuellen Systems ist aber zu groß. Die Hardware benötigt in diesem Fall zu lange, die getrackten Bewegungen zu interpretieren und auf das Display der VR-Brille zu rendern. Beim heutigen Stand der Technik tritt dieses Problem aber nur noch äußerst selten auf. Sensorische Konflikte lassen sich weitestgehend verhindern, indem die Diskrepanzen zwischen den simulierten und tatsächlich empfundenen Bewegungen möglichst gering gehalten werden (vgl. Steinicke 2016:48).

Die Gifttheorie erklärt die Symptome der Cybersickness damit, dass ein im Körper vorhandener Überlebensmechanismus dafür sorgt, dass etwaige giftige Substanzen mittels Übelkeit und anschließenden Erbrechen aus dem Körper befördert werden. Die Theorie nimmt an, dass die visuellen und vestibulären Reize versehentlich Rezeptoren im Gehirn aktivieren, die für die Gifterkennung verantwortlich sind. Diese Rezeptoren leiten schließlich die beschriebenen Symptome ein. Der Gifttheorie mangelt es jedoch an fundierter wissenschaftlicher Evidenz (vgl. Davis/Nesbitt/Nalivaiko 2015:7).

Die Theorie der Haltungsinstabilität bezieht sich auf die Idee, dass ein primäres Ziel des menschlichen Organismus darin besteht Haltungsstabilität zu wahren. Aus diesem Grund führt anhaltende Haltungsinstabilität zu den Symptomen der Cybersickness. Die Theorie basiert auf der Annahme, dass plötzliche Veränderungen in einer Umgebung, in der Haltungsstabilität nicht erlernt wurde, zu Haltungsinstabilität führen. Dies trifft auf viele virtuelle Umgebungen zu, in denen Veränderungen der visuellen Wahrnehmung keinen Bezug zu den normalen Körperbewegungen haben. Im Körper entsteht dann ein Konflikt zwischen den Mechanismen zur Wahrung der Haltungsstabilität, was letztlich die Symptome der Cybersickness hervorruft (vgl. Davis/Nesbitt/Nalivaiko 2015:7).

4 Virtual-Reality-Technologien

In den nachfolgenden Kapiteln werden die relevantesten Ein- und Ausgabetechnologien von Virtual Reality vorgestellt.

4.1 Eingabetechnologien

Die Eingabetechnologien dienen der sensorischen Erfassung von Nutzerinteraktionen und deren Transfer in die virtuelle Umgebung. Dazu werden Position und Rotation der Eingabegeräte kontinuierlich erfasst und in der virtuellen Umgebung als sogenannte Metaphern (virtuelle Hände, 3D-Modelle der Controller, etc.) dargestellt. Tastendrücke lösen schließlich ortsabhängige Aktionen im virtuellen Raum aus. Der Prozess ist vergleichbar mit einer Maus, die auf dem Desktop in Abhängigkeit des Ortes ein bestimmtes Programm durch Klicken auf ein Icon aufruft. Im Folgenden soll daher nur das sogenannte Tracking, also die Positions- und Rotationsüberwachung der Eingabegeräte betrachtet werden. Das Auslösen von Aktionen durch Tastendrücke verhält sich analog zu etablierten Technologien und ist daher nicht Gegenstand dieses Kapitels.

4.1.1 Positionstracking

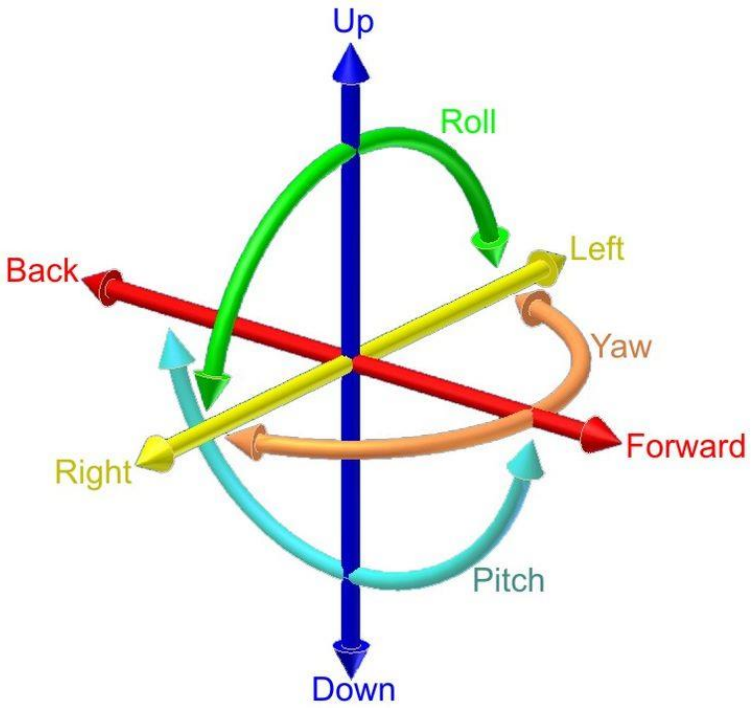

Abb. 7: *Die sechs Freiheitsgrade eines Körpers im freien Raum (Quelle: Wikipedia 2019d)*

Positionstracking bezeichnet das kontinuierliche Überwachen der Position und Orientierung eines Eingabegeräts. Dabei verfolgen Sensoren die Verschiebung des Eingabegeräts im Raum (Translation) und die Drehung des Eingabegeräts um drei senkrecht aufeinander stehende Achsen (Rotation) (vgl. *Abb. 7*). Durch die Überwachung dieser sechs unterschiedlichen Werte, in Form dreier Raumkoordinaten und dreier Winkel, können Ort und Bewegung des Eingabegeräts zu jedem Zeitpunkt eindeutig spezifiziert werden. Diese sechs voneinander unabhängigen Parameter nennt man auch

Freiheitsgrade. Jedes Trackingsystem besitzt zur Überwachung dieser Freiheitsgrade ein eigenes Bezugssystem, das die Rotation und Translation des Eingabegeräts zu jedem Zeitpunkt eindeutig in diesem Bezugssystem verortet (vgl. Dörner et al. 2014:99). Bei der Konzeption einer Virtual-Reality-Anwendung müssen zur Auswahl des passenden VR-Systems stets die folgenden auf das Trackingsystem bezogenen Kriterien berücksichtigt werden (vgl. Dörner et al. 2014:100ff):

- Anzahl der Freiheitsgrade pro verfolgtem Körper
- Anzahl der gleichzeitig verfolgten Körper
- Größe der überwachten Fläche / des überwachten Volumens
- Genauigkeit des Trackingsystems
- Wiederholrate
- Latenz
- Drift
- Empfindlichkeit gegenüber äußeren Rahmenbedingungen
- Kalibrierung
- Usability

Klassifikation nach physikalischem Prinzip

Trackingsysteme basieren auf unterschiedlichen physikalischen Prinzipien. Dazu zählen unter anderem:

- Elektromagnetisches Tracking
- Akustisches Tracking
- Inertial-Tracking
- Optisches Tracking

Beim elektromagnetischen Tracking befinden sich in den Eingabegeräten stromdurchflossene Spulen, die elektromagnetische Felder erzeugen. Die Sensoren zur Überwachung von Translation und Rotation der Eingabegeräte bestehen ebenfalls aus Spulen. In Abhängigkeit von Distanz und Rotation der Eingabegeräte zu den Sensoren, werden unterschiedliche Ströme in den Empfängerspulen induziert. Damit lassen sich die sechs Freiheitsgrade der Eingabegeräte eindeutig ermitteln (vgl. Dörner et al. 2014:112). Eingabegeräte, die auf akustischem Tracking beruhen, nutzen Unterschiede in der Laufzeit oder in der Phase von Schallwellen. Dabei kommt für den Menschen nicht hörbarer Ultraschall zum Einsatz, der eine Frequenz >20.000 Hz besitzt. Mittels eines Senders und eines Empfängers lässt sich die Position eines Eingabegeräts so schon auf die Oberfläche einer Kugel genau reduzieren. Treten weitere Empfänger hinzu wird die Position genauer bestimmbar. Ab drei Empfängern ist schließlich eine exakte Positionsbestimmung möglich. Zusätzliche Sender im Eingabegerät ermöglichen weiterhin eine Bestimmung der Rotation des Eingabegeräts. So lassen sich auch mit

akustischem Tracking alle sechs Freiheitsgrade eindeutig bestimmen. Im Vergleich zu anderen 3D-Tracking-Systemen sind die akustischen Trackingsysteme zwar relativ günstig, dafür aber auch äußerst empfindlich in Bezug auf Temperatur- oder Luftdruckänderungen. Die Systeme müssen daher regelmäßig kalibriert werden (vgl. Dörner et al. 2014:111f).
Das Inertial-Tracking basiert auf dem physikalischen Phänomen der Trägheit. Spezielle Inertialsensoren messen die lineare Beschleunigung zur Bestimmung der Translation. Mithilfe von Gyroskopen (Beschleunigungssensoren für Drehbewegungen) findet die Bestimmung der Rotation statt. Relativ zu einem bestimmten Ausgangspunkt lassen sich auch mit diesem physikalischen Grundprinzip alle sechs möglichen Freiheitsgrade bestimmen (vgl. Dörner et al. 2014:112).
Beim optischen Tracking fungieren Infrarotleuchten im Eingabegerät als sogenannte Marker. Mindestens zwei Kameras beobachten die Bewegungen der Marker im Raum und ermitteln so die Positionsinformationen des Eingabegeräts. Auch in diesem Fall gilt: Werden drei statt eines einzelnen Markers im Eingabegerät verbaut, so wird zusätzlich eine Ermittlung der drei Freiheitsgrade zur Rotation des Eingabegeräts möglich. Um die Marker an einem Gerät unterscheiden und die Freiheitsgrade der Rotation ermitteln zu können, folgen die unterschiedlichen Marker eines Eingabegeräts alle einem unterschiedlichen geometrischen Aufbau (vgl. Dörner et al. 2014:106).

Eingabelösungen

Es existieren drei verschiedene Arten von Eingabelösungen. Sie sind abhängig von der Anzahl der überwachten Freiheitsgrade durch das Trackingsystem. Den Typ der Eingabelösung gilt es bei der Konzeption einer Virtual-Reality-Anwendung zu spezifizieren, da etliche Parameter in der Anwendungsentwicklung davon abhängen. Die drei Eingabelösungen bezeichnet man als:

- Seated
- Standing
- Room Scale

Unter „Seated“ versteht man VR-Anwendungen, die im Sitzen erlebt werden. Anwendungen, die mit der Eingabelösung „Seated“ arbeiten, überwachen in den meisten Fällen nur die drei Freiheitsgrade der Rotation. In der Regel bewegt sich der Nutzer in diesem Szenario nicht fort, weshalb die gesamte Spieleumgebung auf den Bereich um den Nutzer herum konzipiert wird (vgl. Korgel 2017:25f).
Die Eingabelösung „Standing“ beschreibt eine Nutzung der Anwendung im Stehen. Sofern die Freiheitsgrade der Translation hardwareseitig überwacht werden können, ist es dem Nutzer bei dieser Eingabelösung möglich, kleine Schritte zu den Seiten zu machen, die Anwendung wird jedoch nicht auf eine freie Bewegung des Nutzers ausgelegt. Fortbewegung im virtuellen Raum findet bei dieser Eingabelösung häufig über spezielle Fortbewegungstechniken wie „Gehen auf der Stelle“ oder „Zeigen und Teleportieren“ statt (vgl. Korgel 2017:26f).
Unter der Eingabelösung „Room Scale“ versteht man Virtual-Reality-Anwendungen, die darauf ausgelegt sind, dass sich der Nutzer in einem fest definierten Bereich (Spielbereich) frei bewegen kann. Fortbewegung, die über diesen Bereich hinaus geht

wird wie bei der „Standing“-Eingabelösung durch spezielle Fortbewegungstechniken umgesetzt. Bei der Überwachung aller sechs Freiheitsgrade ist der Übergang von „Standing“ zu „Room Scale“ fließend und ergibt sich aus der Größe des Spielbereichs (vgl. Korgel 2017:26f).

4.1.2 Eye-Tracking

Unter Eye-Tracking versteht man die Verfolgung der Blickrichtung des menschlichen Auges. Spezielle Infrarotkameras erfassen die Augen dabei kontinuierlich und ermitteln über die Objektkonturen der Augen das Pupillenzentrum. Ähnlich der Freiheitsgrade beim Positionstracking, bilden die Kameras den Pupillenmittelpunkt in einem zweidimensionalen Bezugssystem nach seiner horizontalen und vertikalen Ausrichtung ab. Da die Augen beim Eye-Tracking in der Regel mit Infrarotlicht beleuchtet werden, können neben der Pupillenposition auch Hornhautreflexionen ausgewertet werden. Ein Differenzvektor zwischen dem Pupillenmittelpunkt und dem Mittelpunkt der Hornhautreflexion, erlaubt eine Ermittlung des Punktes, auf den der Nutzer fokussiert. In Head-Mounted-Displays wird die Eye-Tracking-Technologie häufig als Eingabetechnologie eingesetzt, die eine Steuerung und Interaktion in der virtuellen Welt ermöglicht. (vgl. Dörner et al. 2014:121ff).

4.2 Ausgabetechnologien

Virtual-Reality-Ausgabetechnologien sind das Fenster zur virtuellen Welt. Ziel der VR-Ausgabetechnologien ist es, unter Berücksichtigung der in Kapitel *3 Menschliche Wahrnehmung und Virtual Reality* vorgestellten Wahrnehmungsaspekte des Menschen, einen möglichst hohen Immersionsgrad des Nutzers in die virtuelle Welt zu erreichen:

4.2.1 Head-Mounted-Displays

Abb. 8: *Explosionsdarstellung einer VR-Brille (Quelle: VRNerds o.J.b)*

Als Head-Mounted-Displays (HMD) bezeichnet man mobile Visualisierungs- und Interaktionssysteme, die man in Form eines Helmes oder einer Datenbrille am Kopf trägt.

Über eine Schnittstelle zu einem Rechner werden Videodaten zunächst an das HMD übertragen. Ein miniaturisiertes Stereodisplay erzeugt aus dem Videosignal schließlich zwei Bilder, die durch perspektivische Unterschiede räumliches Sehen ermöglichen. Da das Stereodisplay nur etwa drei bis sieben Zentimeter von den Augen des Nutzers entfernt ist, das menschliche Auge auf diese Distanz jedoch kaum bis gar nicht fokussieren kann, sorgt eine spezielle Linsenoptik für eine Vergrößerung und scharfe Darstellung des Bildes. Neben Display und Linsenoptik sind HMDs in der Regel auch mit einem Trackingsystem zur Positionsüberwachung des Kopfes ausgestattet. (vgl. Dörner et al. 2014:142).

Ausgabeparameter

Aus technischer Sicht lassen sich HMDs anhand dreier zentraler Ausgabeparameter klassifizieren und bewerten, die allesamt signifikanten Einfluss auf das Präsenzgefühl des Nutzers in der virtuellen Welt haben:

- Field of View
- Framerate
- Latenz

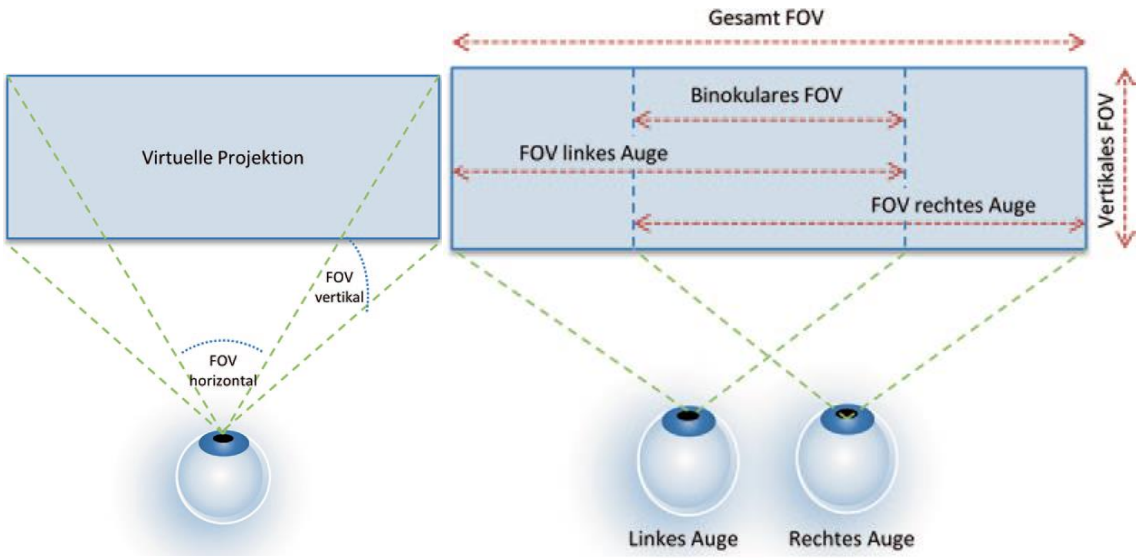

Abb. 9: *links: Prinzipskizze des FoV ausgehend vom Nutzerauge, rechts: FoV bei binokularen HMDs (Quelle: Dörner et al. 2014:143f)*

Unter dem Field of View (FoV) versteht man das Sichtfeld eines Lebewesens – in der Regel das des Menschen. Es charakterisiert sich durch die vom Auge aus horizontal und vertikal aufgespannten Winkel, in denen Licht und damit visuelle Informationen auf die Netzhaut fallen. Das Sichtfeld des Menschen konstruiert sich aus den Sichtfeldern der einzelnen Augen (monokulare Sichtfelder). Im Überlappungsbereich dieser monokularen Sichtfelder entsteht ein binokulares Sichtfeld, also ein Bereich, der von beiden Augen perzipiert wird und der mithilfe des Stereosehens eine dreidimensionale Raumwahrnehmung ermöglicht (vgl. Herman 2016:763 & *Abb. 9*). Im Kontext der Displaytechnologie kann das Field of View vom natürlichen Sichtfeld des Menschen abweichen. Für Displays gilt daher: „Je größer das FOV ist, desto stärker ist die Wahrnehmung des Nutzers, in eine Virtuelle Welt versetzt zu werden, da diese umso

weiträumiger angezeigt wird.“ (Dörner et al. 2014:143).
Die Framerate oder auch Bildfrequenz ist ein weiterer hardwareseitiger Parameter, der entscheidende Auswirkungen auf das Präsenzgefühl der Nutzer von Virtual-Reality-Systemen hat. Als Framerate bezeichnet man die Wiederholrate unterschiedlicher Bilder in einer Sekunde. Zwar ist das menschliche Auge schon ab einer Framerate von 16 Bildern in der Sekunde (Frames per Second = fps) dazu in der Lage fortlaufende Bewegungen wahrzunehmen, um von einer flüssigen Bewegung sprechen zu können, sind je nach Medium aber wesentlich höhere Framerates notwendig (vgl. Vötter 2012). Das Kino beispielsweise arbeitet mit Framerates von 24 fps, wiederholt ein einzelnes Bild aber zwei Mal hintereinander, was einer Bildwiederholfrequenz von 48 fps entspricht. In diesem Frequenzbereich liegt auch die kritische Wiederholrate, bei der eine Einzelbildabfolge nicht mehr direkt wahrnehmbar ist. Erst ab einer Bildwiederholfrequenz von 100Hz gilt ein Bild aber als wirklich flimmerfrei (vgl. Bauer et al. 2009:1666f). Studien zum Thema belegen weiterhin, dass Framerates unter 90 Frames per Second (fps) bei Virtual-Reality-Systemen zu Symptomen wie Orientierungslosigkeit und Übelkeit führen können. Aus diesem Grund muss bei der VR-Softwareentwicklung darauf geachtet werden, dass die Framerate nicht unter 90 fps fällt und dass hardwareseitig hohe Verarbeitungs- und Renderkapazitäten in Form einer leistungsstarken Grafikkarte (GPU) und eines leistungsstarken Prozessors zur Verfügung stehen (vgl. IRISVR o.J.).
Unter Latenz versteht man die Zeitspanne, die ein System benötigt, um auf eine Eingabe zu reagieren. Nutzer erwarten eine unmittelbare Reaktion auf etwaige Eingaben, andernfalls ließe sich die Reaktion des Systems nicht mehr eindeutig auf die Nutzerinteraktion zurückführen. Als Analogie seien hier etwa Energiesparlampen der ersten Generation angeführt, die eine noch recht große Einschaltverzögerung besaßen. Nutzern dieser Lampen passierte es häufig, dass sie nach dem Betätigen des Lichtschalters und der vermeintlich ausbleibenden Reaktion, den Schalter nochmals aus- und wieder einschalteten. Die Wartezeit auf das Licht vergrößerte sich dadurch deutlich (vgl. Dörner et al. 2014:196). Auch in virtuellen Welten bringen hohe Latenzen ähnliche Probleme mit sich. Am Beispiel eines Werkzeuges, das über ein Trackingsystem an die Handbewegungen des Nutzers gekoppelt ist, lässt sich dies veranschaulichen. Werden die Latenzen hier zu groß, „so wird das Werkzeug nicht direkt mit der Hand mitgeführt, sondern, insbesondere bei schnellen Bewegungen, in mehr oder minder großem Abstand nachgezogen“ (Dörner et al. 2014:196). Auch hier stört die verzögerte Reaktion auf etwaige Nutzereingaben die Wahrnehmung des Nutzers. Bezüglich der Auswirkungen von Latenz auf die Wahrnehmung lässt sich also schlussfolgern:

> *„Je größer die Latenz des Systems, desto größer ist der zeitliche Abstand der wahrnehmbaren Konsequenz einer Handlung und desto dissoziierter ist die Wirkung auf den Nutzer.“* (Dörner et al. 2014:196)

Die Latenz eines VR-Systems sollte daher stets unter der menschlichen Wahrnehmungsschwelle liegen. Untersuchungen zum Thema konnten zeigen, dass eine Erhöhung der Latenz von 50ms auf 90ms mit einem signifikanten Anstieg von Symptomen der Cybersickness bei den Testpersonen einherging (vgl. Meehan et al.

2003). Für HMDs wird daher eine Latenz von unter 50ms empfohlen (vgl. Brooks 1999:18f).

4.2.2 Akustische und Haptische Ausgabetechnologien

Neben der visuellen Ausgabe, die primär über Head-Mounted-Displays stattfindet, sprechen Virtual-Reality-Systeme in der Regel auch die akustische und die haptische Wahrnehmung an. Für die akustische Wahrnehmung werden Audiosysteme meist direkt in den HMDs verbaut. Eine zentrale Anforderung zur Unterstützung der Raumwahrnehmung ist dabei die Fähigkeit der Audiosysteme zur Erzeugung von Raumklang (vgl. Dörner et al. 2014:154).
Das Ansprechen der haptischen Wahrnehmung geschieht über zusätzliche Eingabegeräte wie beispielsweise Controllern, die in den Händen gehalten werden und der Interaktion in virtuellen Welten dienen. Haptisches Feedback wird dort durch spezifisch ausgelöste Vibrationen erzeugt (vgl. Dörner et al. 2014:154f).

5 Interaktion in virtuellen Welten

Virtual Reality ist ein Medium, das im Gegensatz zu Büchern oder Filmen nicht nur auf den Konsum von, sondern auch die Interaktion mit Inhalten ausgelegt ist. Der Nutzer tritt unter Echtzeitbedingungen in Wechselwirkung mit der virtuellen Umgebung. Es findet ein Informationsaustausch zwischen dem Nutzer und dem Computer, der die virtuelle Welt steuert, statt. Aus diesem Grund handelt es sich bei der Nutzung des Mediums Virtual Reality immer auch um eine Form der Mensch-Computer-Interaktion (MCI) (vgl. Dörner et al. 2014:158). Während bei klassischen Schnittstellen der Mensch-Computer-Interaktion wie der Interaktion mit einem Laptop oder Smartphone seit einigen Jahren etablierte Vorgehensweisen zur Entwicklung effektiver MCI existieren, fehlen diese Erkenntnisse im Kontext von Virtual Reality jedoch noch (vgl. Dörner et al 2014:159). Usability und User Experience stellen daher die entscheidenden Konzepte zur Bewertung verschiedener Interaktionstechniken in virtuellen Welten dar. Die Auswahl für ein VR-Projekt sollte daher stets auf Basis des Nutzungskontexts, der Nutzererfahrung und des Grads an Natürlichkeit erfolgen (Bowman et al. 2004:331). Im Folgenden werden verschiedene Interaktionstechniken für das Medium Virtual Reality vorgestellt und erläutert.

5.1 Navigation

Die Navigation durch die virtuelle Welt und die Art der Fortbewegungen (Lokomotion) stellen zentrale Herausforderungen für eine gelungene Immersion des Nutzers in die virtuelle Welt dar. Der Nutzer muss sich möglichst einfach, intuitiv und natürlich fortbewegen können. Gerade bei der Bewertung des Grades an Natürlichkeit enttäuschen die existierenden Lokomotionstechniken, bleibt doch der Bewegungsspielraum des Nutzers im Realen stets begrenzt und damit auch die natürlichste Form der Lokomotion: Das Gehen. (vgl. Bozgeyikli 2016:4).

Bezüglich der Navigation ist zwischen den Teilbereichen Wegfindung und Bewegungskontrolle zu unterscheiden. Das Konzept der Wegfindung beschreibt den kognitiven Prozess, bei dem das menschliche Gehirn die Umgebung mental kartographiert, um sich in der Umgebung intuitiv zurechtzufinden. Die Bewegungskontrolle beschreibt die motorische Komponente der Navigation, also die grundlegenden Lokomotionstechniken, mit denen Position und Orientierung des virtuellen Kameraausschnitts passend verändert werden können (vgl. Dörner et al. 2014:169f). Bowman et al. (2005) definieren drei Aufgabenstellungen an Nutzer virtueller Welten, bei denen Wegfindung und Bewegungskontrolle eine entscheidende Rolle spielt:

- Exploration
- Suche
- Manövrieren

Bei der Exploration folgt der Nutzer keinem konkreten Ziel, er erkundet die virtuelle Umgebung frei. Die Aufgabestellungen Suche und Manövrieren ähneln sich: Bei der Suche verfolgt der Nutzer das Ziel, zu einer bestimmten Position zu gelangen. Beim

Manövrieren hingegen, geht es um die Ermittlung einer exakten Position in der unmittelbaren Nähe des Nutzers, was häufig an eine Folgeaufgabe wie die Manipulation eines Objekts geknüpft ist. Die Aufgabenstellungen ergeben sich aus der VR-Anwendung, weshalb die Auswahl passender Lokomotionstechniken stets anwendungsspezifisch erfolgen sollte (vgl. Bowman 2014:185f). Im Folgenden werden verschiedene Steuerungstechniken zur Bewegungskontrolle vorgestellt.

5.1.1 Steuerungstechniken zur Bewegungskontrolle

Reales Gehen

Reales Gehen gilt als die intuitivste Lokomotionstechnik. Es erreicht den bestmöglichen Immersionsgrad, sowohl was die Bewegungskontrolle in virtuellen Welten als auch die Wegfindung betrifft und ist anderen Lokomotionstechniken diesbezüglich deutlich überlegen. Dennoch unterliegt das Reale Gehen starken Beschränkungen, was vor allem auf die Raumbegrenzungen des überwachten Bereichs zurückzuführen ist. Sobald die virtuelle Umgebung größere Dimensionen als die des abgedeckten Trackingbereichs annimmt, läuft der Nutzer Gefahr, den Trackingbereich zu verlassen und seine Präsenz in der virtuellen Welt plötzlich zu verlieren (vgl. Bozgeyikli 2016:16).
Um die Grenzen des realen Gehens zu überwinden existieren mittlerweile sowohl software- als auch hardwareseitige Lösungen. Vielversprechend aus Sicht der softwareseitigen Lösungen ist das sogenannte „Redirected Walking“. Bewegt sich ein Nutzer beispielsweise entlang einer Geraden, so führt eine nicht bemerkbare Rotation der virtuellen Kamera dazu, dass der Nutzer diese Bewegung unbewusst kompensiert. Um in der virtuellen Welt weiterhin geradeaus laufen zu können, muss der Nutzer in der Realität also eine Kurvenbahn vollziehen. Er erlebt die Illusion des Geradeauslaufens, bewegt sich aber stets im Kreis, ohne den Trackingbereich je zu verlassen. Untersuchungen konnten zeigen, dass der Nutzer bei einem Radius >20 Meter nicht mehr dazu in der Lage ist diese Manipulation zu erkennen (vgl. Steinicke et al. 2010:18ff). Aufgrund der notwendigen Größe des Trackingbereichs ist „Redirected Walking“ stark vom Anwendungskontext abhängig und in den seltensten Fällen realisierbar.
Zu den aussichtsreichsten hardwareseitigen Lösungen zählen omnidirektionale Laufbänder. Die Laufbänder passen sich an die Gehgeschwindigkeit des Nutzers an und erlauben ihm eine uneingeschränkte Bewegung in alle Richtungen. Omnidirektionale Laufbänder sind jedoch sehr teuer und daher ebenso unpassend für die meisten Anwendungsszenarien von Virtual-Reality-Anwendungen (vgl. Bozgeyikli 2016:31f).

Gehen auf der Stelle

Gehen auf der Stelle ist eine gestengesteuerte Lokomotionstechnik. Bei dieser Lokomotionstechnik läuft der Nutzer auf der Stelle. Die Geste wird durch das Trackingsystem erkannt und als Bewegung entlang der Blickrichtung des Nutzers in der virtuellen Welt umgesetzt. Gehen auf der Stelle wahrt das Präsenzgefühl des Nutzers: Sowohl Tiefenwahrnehmung als auch visuelles System erhalten Informationen, dass sich der eigene Körper gerade gehend fortbewegt. Bisher lässt sich die Geschwindigkeit aus der stehenden Bewegung des Nutzers heraus aber noch nicht zufriedenstellend interpolieren. An dieser Stelle entstehen Widersprüche zwischen dem visuellen System und der Tiefenwahrnehmung. Der Nutzer empfindet seine Fortbewegung in der virtuellen Welt entweder als zu schnell oder als zu langsam. Auch zwischen Lagesinn und

dem visuellen System kommt es zu Widersprüchen. Während der Lagesinn die Information vermittelt, dass sich der Körper aktuell in Ruhe befindet, liefert das visuelle System die Information, dass sich der Körper bewegt. Nach der in Kapitel *3.6.7 Cybersickness* vorgestellten sensorischen Konflikttheorie kann dies ein Auslöser für Cybersickness sein (vgl. Bozgeyikli 2016:23ff).

Fliegen

Beim Fliegen dienen im Vergleich zum Realen Gehen beziehungsweise dem Gehen auf der Stelle andere Eingabeformen wie z.B. Tastendrücke oder Joystickbewegungen als Auslöser der Fortbewegung in der virtuellen Umgebung. Auch hier korreliert die Bewegungsrichtung mit der Blickrichtung des Nutzers. Die Lokomotionstechnik Fliegen ist zwar leicht umzusetzen, erzielt jedoch schwächere Präsenzgefühle als die Lokomotionstechniken Reales Gehen und Gehen auf der Stelle. Der Nutzer hat bei dieser Technik keine natürliche Möglichkeit zur Einflussnahme auf die Geschwindigkeit, was den Grad der Natürlichkeit dieser Technik deutlich einschränkt und zu einem sensorischen Konflikt zwischen dem visuellen System und dem Lagesinn führen kann (vgl. Bozgeyikli 2016:27ff).

Neigen

Beim Neigen löst der Nutzer die Bewegungsreaktion aus, indem er sich leicht nach vorne lehnt. Aus technischer Sicht reagiert das Trackingsystem also auf bestimmte Rotationswerte des Head-Mounted-Displays. Die Richtung der Fortbewegung wird durch die Blickrichtung des Nutzers bestimmt. Der Nutzer ist bei dieser Lokomotionstechnik dazu in der Lage, die Geschwindigkeit der Fortbewegung durch die Stärke seiner Neigung zu steuern. In einer Vergleichsstudie der Techniken „Neigen“ und „Gehen auf der Stelle“ traten bei der Neigen-Technik geringere Latenzen auf. Darüber hinaus war die Technik weniger fehleranfällig. Die Bewertung der Wegfindung fiel zwischen beiden Techniken ähnlich aus, die Probanden sprachen sich jedoch mehrheitlich für die Lokomotionstechnik „Neigen“ aus (vgl. Bozgeyikli 2016:27ff).

Controllerbasierte Fortbewegung

Die controllerbasierte Fortbewegung mittels Joysticks, Touchpads oder Tastatur gehört zu den am häufigsten genutzten Lokomotionstechniken. Sie ist billig, einfach zu implementieren und die Nutzer sind an Interaktionen mit diesen Eingabegeräten in der Regel gewohnt. Bei der controllerbasierten Steuerung dienen Tastendrücke als Auslöser der Bewegung. Werden Joysticks verwendet, so tritt die Möglichkeit der Geschwindigkeitssteuerung hinzu. Die Geschwindigkeit korreliert dabei mit der Stärke der Neigung des Joysticks. Auch die Richtungssteuerung läuft über Controllereingaben. Nachteilig erweisen sich diese Lokomotionstechniken jedoch bezüglich ihres Grads an Natürlichkeit und des Präsenzgefühls bei der Nutzung. Bewegungen im Virtuellen ergeben sich nicht aus den realen Bewegungen des Nutzers, was zu einem sensorischen Konflikt und den Symptomen der Cybersickness führen kann (vgl. Bozgeyikli 2016:35ff).

Zeigen und Teleportieren

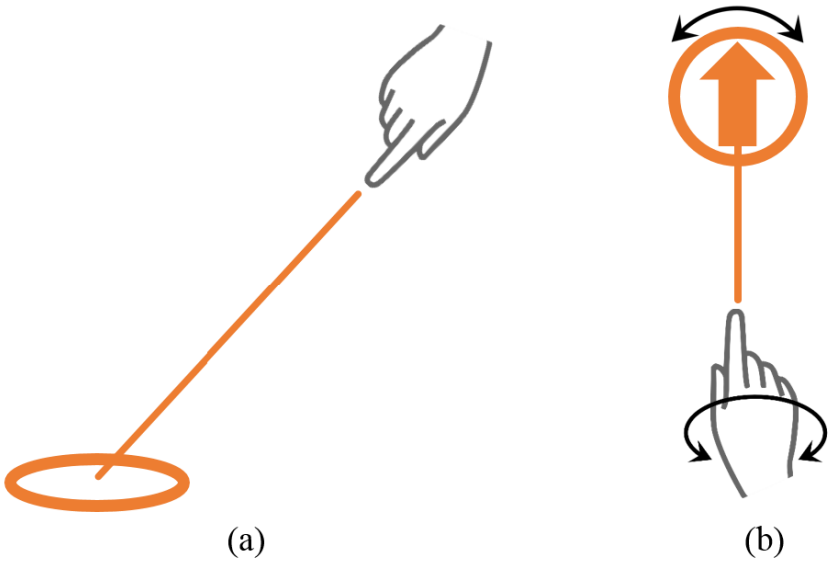

Abb. 10: *Illustration von (a) Zeigen und Teleportieren und (b) Zeigen und Teleportieren mit Spezifikation der Blickrichtung (Quelle: Bozgeyikli 2016:125)*

Zeigen und Teleportieren gehört zu den unmittelbaren Lokomotionstechniken. Das bedeutet, dass sich der Nutzer nicht mit einer Geschwindigkeit entlang eines Vektors fortbewegt, sondern instantan von Punkt zu Punkt teleportiert wird. Teleportieren stellt eine einfache, aber dennoch leistungsstarke Alternative zu den bisher vorgestellten Steuerungstechniken zur Bewegungskontrolle dar (vgl. Bozgeyikli 2016:124).
Bei dieser Lokomotionstechnik greift der Nutzer auf Controller zurück, über die er in der virtuellen Welt einen Zeiger in Form eines Strahls oder einer Parabel steuern kann. Der Punkt, an dem der Zeiger auf die Umgebung trifft, stellt das Ziel der Teleportation dar (vgl. *Abb. 10*). Durch Tätigen einer Eingabe, beispielsweise durch einen Tastendruck, wird die Teleportation ausgelöst. Der Nutzer landet augenblicklich an der definierten Zielposition (vgl. Bozgeyikli 2016:124f).
Erweitern lässt sich die Technik „Zeigen und Teleportieren“ durch die Möglichkeit, die Blickrichtung nach erfolgter Teleportation festlegen zu können. Mithilfe eines Joysticks oder über die Rotationswerte des Eingabegeräts ist dies auch technisch umsetzbar. Durch die Bestimmung der Blickrichtung kann die Wegfindung des Nutzers verbessert werden (vgl. Bozgeyikli 2016:125). Neben der Erweiterung um die Möglichkeit zur Vorgabe einer Blickrichtung können auch Einschränkungen der Technik Vorteile mit sich bringen. Mithilfe von fest definierten Zielpunkten können lineare Pfade erstellt werden, die die Bewegungsfreiheit des Nutzers deutlich einschränken, seine Wegfindung damit jedoch verbessern. (vgl. Bozgeyikli 2016:125f).
Der Lokomotionstechnik „Zeigen und Teleportieren“ mangelt es jedoch im Vergleich zu anderen Lokomotionstechniken aufgrund der instantanen Fortbewegung an Natürlichkeit. Gerade die Wegfindung kann bei dieser Form der unmittelbaren Fortbewegung gestört werden, da der Nutzer sich nach erfolgter Teleportation jedes Mal von neuem orientieren muss. Es liegt daher die Vermutung nahe, dass dies das Präsenzgefühl der Nutzer einschränken könnte (vgl. Bozgeyikli 2016:175f). Buttussi und Chittaro (2019) konnten in einer vergleichenden Analyse zwischen den

Lokomotionstechniken „Zeigen und Teleportieren“ und „Neigen“ jedoch keine Auswirkungen auf das Präsenzgefühl der Probanden feststellen. Was die Performanz betrifft, stellte sich in diesem Vergleich vielmehr heraus, dass die Nutzer der Technik „Zeigen und Teleportieren“ Aufgaben wesentlich schneller lösen konnten, als die Nutzer der Lokomotionstechnik „Neigen“. Weiterhin zeigte sich, dass Symptome der Cybersickness, darunter unter anderem Übelkeit, beim Neigen sogar marginal häufiger auftraten als beim Teleportieren. Die Autoren der Analyse führten dies auf mögliche sensorische Konflikte zurück, die beim Neigen auftreten können – beim Teleportieren jedoch nicht (vgl. Buttussi / Chittaro (2019). Aus dem Vergleich acht verschiedener Lokomotionstechniken durch Bozgeyikli geht „Zeigen und Teleportieren“ in Belangen der Nutzerpräferenz sogar als Sieger hervor. Teilnehmer seiner Untersuchung präferierten „Zeigen und Teleportieren“ vor allen anderen untersuchten Lokomotionstechniken – trotz ihres mangelnden Grads an Natürlichkeit (vgl. Bozgeyikli 2016:175f).

5.2 Selektion

Einer der zentralen Aspekte zur Interaktion in einer virtuellen Umgebung ist die Selektion:

> „*Selektion bedeutet, dass der Nutzer einen Punkt, eine Fläche oder ein Volumen in der Virtuellen Welt bestimmt (z.B. um dort ein Objekt einzufügen) oder eine für ihn semantisch bedeutsame Teilmenge dieser Welt auswählt (z.B. ein bestimmtes virtuelles Objekt oder Teilobjekt, um es zu bewegen).“* (Dörner et al. 2014:160)

Die Selektion von Objekten dient deren anschließender Manipulation. Bei der Auswahl von Interaktionstechniken müssen Selektion und Manipulation daher stets im Verbund und nicht isoliert betrachtet werden. Im Folgenden werden gängige Selektionstechniken vorgestellt:

5.2.1 Selektionstechniken

Ray-Casting

Ray-Casting ist die wohl am häufigsten genutzte Selektionsmethode. Beim Ray-Casting steuert der Nutzer über das Eingabegerät einen Strahl in der virtuellen Welt. Alle durch den Strahl geschnittenen Objekte bilden mögliche Selektionskandidaten, von denen das dem Nutzer am nächsten liegende Objekt ausgewählt wird. Mit steigender Distanz nimmt die Handhabbarkeit dieser Methode jedoch ab, weil der mit der Hand einzustellende Winkel immer kleiner und die Selektion von Objekten daher immer schwerer wird. Außerdem erweist sich die Manipulation eines Objekts, das durch den Strahl ausgewählt wurde, auf große Distanzen als problematisch. (vgl. Bowman / Hodges 1999).

Flashlight-Technik

Die Flashlight-Technik ähnelt dem Ray-Casting stark. Statt eines Strahls kommt bei dieser Technik jedoch ein Kegel ähnlich dem einer leuchtenden Taschenlampe zum Einsatz. Bei der Flashlight-Technik kommen alle vom Kegel getroffenen Objekte als Selektionskandidaten in Frage. Zusätzliches Auswahlkriterium ist hier die Entfernung vom Zentrum des Kegels. Da der Kegel mit steigender Distanz immer breiter wird, ist es einfacher, Objekte aus großer Distanz zu selektieren. (vgl. Dörner et al. 2014:164).

Go-Go-Technik

Bei der Go-Go-Technik findet eine Verlängerung des virtuellen Arms statt. Innerhalb der normalen Interaktionsdistanz von etwa 50cm verhält sich die virtuelle Hand analog zur realen Hand. Die getrackten Bewegungen skalieren linear. Außerhalb dieser Distanz wird die Bewegung der Hand nicht-linear skaliert. Mit steigender Entfernung vom Nutzer können so immer größere Distanzen mit der gleichen Handbewegung zurückgelegt werden. Die Go-Go-Technik weist jedoch die gleiche Winkelabhängigkeit wie das Ray-Casting auf, weshalb sie auf große Distanzen nur schwer handhabbar ist (vgl. Bowman / Hodges 1999).

HOMER-Technik

HOMER steht für „Hand-centered Object Manipulation Extending Ray-casting". Auch bei dieser Technik wird ein Strahl aus der aktuellen Handposition extrapoliert. Im Unterschied zum normalen Ray-Casting wird die virtuelle Hand jedoch an dem getroffenen Objekt befestigt, was die Manipulation des Objekts auf große Distanzen deutlich vereinfacht. Da der Selektionsmechanismus identisch zum Ray-Casting ist, entsteht auch hier eine Winkelabhängigkeit, die eine Selektion auf große Distanzen erschwert (vgl. Bowman / Hodges 1999).

World-in-Miniature-Technik

Bei der World-in-Miniature-Technik verlässt der Nutzer seine egozentrische Perspektive. Er trägt eine Miniatur als Modell der virtuellen Umgebung bei sich, über die er Selektionen mit seinen virtuellen Händen durchführen kann (vgl. Dörner et al 2014:164).

5.3 Manipulation

Unter Manipulation versteht man das Verändern von Objekteigenschaften wie Position, Orientierung im Raum (Rotation), Größe, Form, Gewicht, Geschwindigkeit, Erscheinung etc. (vgl. Dörner et al. 2014:166). Im Folgenden werden die gängigsten Manipulationstechniken vorgestellt:

5.3.1 Manipulationstechniken

Arcball

Die Arcball-Technik dient der Manipulation der Orientierung eines Objekts im Raum (Rotation). Dabei wird das Objekt in eine Kugel gehüllt und die Interaktionen des Nutzers als Rotationen der Kugel um deren Mittelpunkt übertragen (vgl. Dörner et al. 2014:166).

Virtuelle Hand

Die Manipulationstechnik der virtuellen Hand verfolgt eine möglichst realitätsnahe Manipulation von Objekten in der virtuellen Umgebung. Sie besitzt einen hohen Grad der Natürlichkeit und steigert damit das Präsenzgefühl des Nutzers im Virtuellen Raum. Bei dieser Manipulationstechnik kontrolliert der Nutzer virtuelle Hände, die sich über getrackte Eingabegeräte wie Controller oder Handschuhe steuern lassen. Virtuelle Hände sind für realitätsnahe Anwendungsszenarien konzipiert. Sie schränken den Nutzer aber auch in seiner Greifreichweite ein, solange sie nicht durch Selektionstechniken wie Ray-Casting oder die Flashlight-Technik (Zeigegesten) erweitert werden. Mittels Gestensteuerung oder Eingaben via Tasten und Joysticks am Eingabegerät lassen sich

selektierte Objekte mithilfe der virtuellen Hände manipulieren. Dies geschieht beispielsweise durch Übertragung der Handbewegungen, die die Rotation und Translation eines selektierten Objekts erlauben (vgl. Dörner et al. 2014:167f).

Voodoo-Dolls

Die Manipulationstechnik Vodoo-Dolls ähnelt stark der World-in-Miniature-Selektionstechnik. Im Unterschied zur WiM-Technik trägt der Nutzer jedoch kein miniaturisiertes Modell seiner Umgebung mit sich, sondern ein Modell des selektierten Objekts. Der Nutzer manipuliert dieses Modell und die Manipulationen werden auf das Originalmodell übertragen. (vgl. Dörner et al. 2014:168).

6 Software und Hardware

Mit dem Aufschwung von Virtual Reality als massentaugliche Technologie haben sich verschiedene Hardware- und Softwaresysteme für die Entwicklung von Virtual-Reality-Anwendungen etabliert. Die Gängigsten werden im Folgenden vorgestellt:

6.1 Software

Im Bereich der VR-Softwareentwicklung haben sich die in der Videospielbranche führenden Entwicklungsplattformen Unity und Unreal Engine festgesetzt. Beide Plattformen stellen Entwicklungsumgebungen für Spiele und Grafikanwendungen bereit und erweiterten ihr Portfolio im Kontext des Aufschwungs der VR-Technologie um Schnittstellen zu den etablierten Virtual-Reality-Systemen.

6.1.1 Unity

Unity ist eine Laufzeit- und Entwicklungsumgebung für Spiele und andere Grafikanwendungen des Unternehmens Unity Technologies.

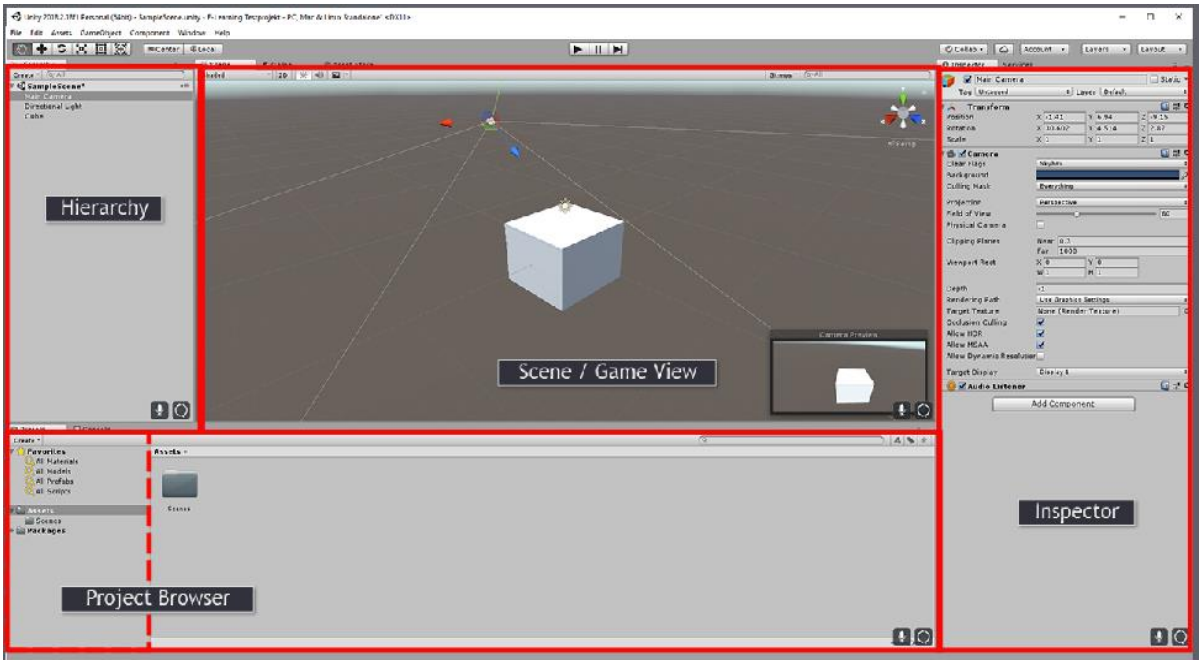

Abb. 11: *Aufbau der Nutzeroberfläche von Unity (Quelle: Eigene Darstellung)*

Der Unity-Editor ist gängigen 3D-Animationsprogrammen nachempfunden (vgl. *Abb. 11*). Er besteht aus einem Hauptfenster (Scene View), über das die Szene dargestellt und bearbeitet werden kann. Über Mauseingaben können Objekte skaliert, verschoben und gedreht werden. Ein Objekt ist dabei zunächst ein Punkt im Koordinatensystem der Spielumgebung. Die Funktion eines Objekts wird durch dessen Typ definiert. Einfache Objekte wie Lichter oder simple geometrische Formen können im Editor direkt erzeugt und in der Szene platziert werden. Komplexe 3D-Modelle oder ganze Objekthierarchien („Prefabs“) werden in der Regel importiert. In einer objektorientierten Struktur („Hierarchy“) können die Objekte einer Szene hierarchisch organisiert werden. Jedes Objekt kann über die Struktur außerdem gezielt angewählt und über das Inspector-Fenster anschließend konfiguriert und um objektspezifische Komponenten

(Audioquellen, Collider, Rigidbodies, benutzerdefinierte Skripte, etc.) erweitert werden. (vgl. Wikipedia 2019c).
Unity stellt außerdem Schnittstellen zu den gängigen Virtual-Reality-Systemen wie Oculus Rift oder HTC Vive bereit. Die Anbieter dieser Systeme liefern ihrerseits sogenannte Software Development Kits (SDKs) aus, die die Softwareentwicklung von Virtual-Reality-Anwendungen durch eine Sammlung von Programmierwerkzeugen und Programmbibliotheken erleichtern sollen (vgl. Unity 2018 & Wikipedia 2019b). Bei Unity gilt es besonders dessen Nutzerfreundlichkeit auch für Anfänger hervorzuheben. Weiterhin ist die Entwicklungsumgebung Branchenführer im Multiplattform-Support. Aus einer Quelle lassen sich mit Unity Anwendungen für über 25 Plattformen erzeugen, darunter unter anderem (vgl. Viscircle 2018):

- Windows
- Linux
- Mac
- Playstation 4
- Xbox One
- Wii U
- Nintendo Switch
- iOS
- Android
- WebGL

Virtual Reality Toolkit

Das Virtual Reality Toolkit (VRTK) ist ein Framework für die VR-Entwicklung mit Unity, das aus einer großen Sammlung von Skripten und Prefabs besteht, die dabei helfen sollen, Virtual-Reality-Lösungen schnell und einfach zu implementieren. Als Open-Source-Projekt leistet die Community hinter VRTK einen großen Anteil an der Weiterentwicklung des Frameworks durch eine stetige Implementierung neuer Features. Das Virtual Reality Toolkit unterstützt u.a. die SDKs von Oculus und SteamVR und bietet seinen Nutzern verschiedene Lösungen zu Lokomotion und Interaktion in virtuellen Umgebungen (vgl. VRTK o.J. & Heanley 2019).

6.1.2 Unreal Engine

Die Unreal Engine ist ein Framework zur Entwicklung von Spielen und anderen Grafikanwendungen. Im Vergleich zu Unity vermarktet sich die Unreal Engine als Sammlung von einzelnen Werkzeugen zur Spieleentwicklung. Im Kern ähnelt die Unreal Engine jedoch der Funktionsweise von Unity: Sie setzt sich aus einer Grafik-Engine, einer Skriptsprache (UnrealScript) und Hilfsprogrammen zusammen. Mit Hilfsprogrammen sind einzelne Editoren gemeint, die verschiedene Aufgaben in der Spieleentwicklung übernehmen (vgl. Unreal Engine 2019b). Auch Unreal Engine ermöglicht die Entwicklung von Virtual-Reality-Anwendungen über Schnittstellen zu Virtual-Reality-

Systemen. Wie für Unity, stellen die Hersteller der Virtual-Reality-Systeme auch für Unreal Engine Software Development Kits bereit (vgl. Unreal Engine 2019a). Die Unreal Engine überzeugt besonders durch ihre leistungsstarke Grafik-Engine, die sich mit Features wie komplexen Partikelsimulationssystemen oder fortschrittlicher dynamischer Beleuchtung auszeichnet (vgl. Viscircle 2018).

6.2 Hardware

Virtual Reality erlebte in den vergangenen Jahren einen rasanten technologischen Aufschwung. Im Zuge dieser Entwicklung etablierten sich verschiedene Hersteller von Virtual-Reality-Systemen auf dem Markt. Für die Entwicklung der Virtual-Reality-Anwendung „Climeat Change“ standen jedoch nur die Oculus Rift sowie die HTC Vive zur Verfügung. Beide Systeme zählten lange Zeit zu den leistungsstärksten Systemen auf dem Markt. Mittlerweile sind sie durch ihre eigenen Nachfolgermodelle die Oculus Rift S sowie die HTC Vive Pro abgelöst worden. Da die Nachfolgermodelle jedoch erst im Zuge dieser Thesis veröffentlicht wurden und daher auch nicht zur Verfügung standen, werden im Folgenden noch die Vorgängermodelle Oculus Rift und HTC Vive vorgestellt.

6.2.1 Head-Mounted-Displays

Oculus Rift

Abb. 12: *Oculus-Rift-Headset (Quelle: vrbrillen.net o.J.)*

Die Oculus Rift ist ein Virtual-Reality-Headset vom gleichnamigen Hersteller Oculus. Das Headset wiegt rund 470g und wird über ein Kabel mit dem PC verbunden. Es skaliert mit einer Auflösung von 2160 x 1200 Pixeln auf zwei 90mm OLED Displays, die mit einer Framerate von 90 fps arbeiten. Beide Displays spannen insgesamt ein Gesichtsfeld von 100° in der Diagonalen auf (vgl. Nield 2016).
Im Headset ist ein sogenannter „Adjacent Reality Tracker“ verbaut, der aus einem Magnetometer, einem Beschleunigungssensor und einem Gyroskop besteht und die Headtracking-Funktionalität unterstützt. Darüber hinaus besteht das Trackingsystem aus einem stationären Sensor, der die Position im Raum anhand zweier am Headset befestigter Infrarot-LEDs ermittelt. Das Gesamtsystem ermöglicht so die Überwachung aller sechs Freiheitsgrade, bei einer sehr geringen Latenz von nur 1-4 ms. Die maximale Größe des Trackingbereichs der Oculus Rift wird mit 4m x 4m (2 Sensoren) angegeben. Über ein integriertes Audiosystem ist die Oculus Rift außerdem dazu in der Lage 360°-Raumklang zu erzeugen (vgl. Nield 2016). Dadurch schafft das Audiosystem die

Möglichkeit, auch das Richtungshören anzusprechen und damit das Präsenzgefühl der Nutzer zu erhöhen.

HTC Vive

Abb. 13: *HTC Vive (Quelle: Glatz 2016)*

Die HTC Vive wurde in einer Kooperation der Unternehmen HTC und Valve entwickelt. Sie gilt als größter Konkurrent zur Rift und liefert hardwareseitig sehr ähnliche Spezifikationen: Die Vive ist mit zwei OLED Displays ausgestattet und löst in einem Verhältnis von 2160 x 1200 auf. Die Framerate liegt bei 90 Hz, das Sichtfeld erstreckt sich diagonal über einen Winkel von 110°. Damit ist die HTC Vive in ihren Hardwarespezifikationen nahezu identisch zur Oculus Rift. Sie kann lediglich mit einem leicht vergrößerten Sichtfeld punkten (vgl. VRNerds o.J.a).
Das Unternehmen Valve steuerte zur Entwicklung das einzigartige Trackingsystem Lighthouse bei. Dabei sendet eine Basisstation Laserstrahlen aus, die von Fotosensoren auf dem Headset erkannt werden. Basierend auf der zeitlichen Differenz zwischen dem Auftreffen der Laserstrahlen auf den jeweiligen Sensoren, kann der Computer die exakte Position und Orientierung des Geräts errechnen. Die Lighthouse-Technologie ist damit in der Lage alle sechs Freiheitsgrade zu überwachen. Um ein fehlerfreies 360°-Tracking zu gewährleisten, werden zwei Basisstationen mitgeliefert, die an gegenüberliegenden Ecken eines Raumes platziert werden. Die Lighthouse-Technologie ist so dazu in der Lage, einen Trackingbereich von bis zu 5 m x 5 m zu überwachen. Dabei benötigen die Basisstationen ausschließlich Strom, sie müssen nicht mit dem Headset oder dem Computer kommunizieren. Im Vergleich zur Oculus Rift besitzt die HTC Vive kein integriertes Audiosystem, kann aber um eine solche Komponente durch Zukauf erweitert werden (vgl. Wikipedia 2019a).

6.2.2 Peripherie

Die vorgestellten Virtual-Reality-Systeme besitzen ergänzende Peripheriegeräte, die die Grundlage für Interaktionen in virtuellen Welten schaffen. Außerdem lassen sich die VR-Systeme um Peripheriegeräte von Drittanbietern erweitern, die weitere Eingabeformen wie Gestensteuerung und uneingeschränktes Gehen ermöglichen.

Oculus Touch

Abb. 14: *Oculus Touch Controller (Quelle: Peix 2019)*

Beim Release der Oculus Rift waren noch keine Controller für das Virtual-Reality-System verfügbar. Erst sechs Monate später veröffentlichte Oculus mit Oculus Touch die zum System gehörende Peripherie. Die Touch-Controller sind daher nicht standardmäßig im Lieferumfang der Oculus Rift enthalten (vgl. Jannsen 2016).
Oculus Touch umfasst zwei Tracking-Controller und einen Sensor. Der Sensor erhöht die Zahl der Sensoren des Gesamtsystems, also inklusive Brille, auf zwei. Damit vergrößert sich automatisch auch der Trackingbereich. Die Controller sind batteriebetrieben und vereinen die traditionellen Funktionalitäten von normalen Controllern mit dem Positionstracking des Oculus-Rift-Systems. Über verschiedene Tasten sind die Nutzer dazu in der Lage, Aktionen in der virtuellen Umgebung auszulösen. Kapazitive Sensoren, die auf den Tasten verbaut sind, überwachen außerdem die Position der Finger auf den Controllern. Standardmäßig werden die Controller in der virtuellen Umgebung durch virtuelle Hände repräsentiert. Mithilfe der kapazitiven Sensoren lassen sich so verschiedene Gesten der virtuellen Hände realisieren. Ruht der Daumen beispielsweise nicht auf einer der für ihn vorgesehenen Tasten, so interpretiert das System den Zustand als Geste und formt die virtuelle Hand zu einer Faust mit abgespreiztem Daumen (vgl. *Abb. 15*).

Abb. 15: *Gestenbasierte Steuerung der virtuellen Hände mit Oculus Touch (Quelle: in Anlehnung an Oculus 2019)*

Das Positionstracking der Touch-Controller ermöglicht außerdem die Überwachung aller sechs Freiheitsgrade. Damit können sowohl Rotation als auch Translation der Eingabegeräte in den virtuellen Raum transferiert werden. (vgl. Laukkonen 2019).

HTC Vive Controller

Abb. 16: *HTC Vive Controller (Quelle: VRPlayground o.J.)*

Die HTC Vive Controller sind standardmäßig im Lieferumfang der HTC Vive enthalten. Sie bestehen aus verschiedenen Tasten und Triggern sowie einem Touchpad, die Aktionen in der virtuellen Welt auslösen können (vgl. Wikipedia 2019a). Die Controller sind kompatibel mit dem Lighthouse-Trackingsystem und können so in allen sechs Freiheitsgraden überwacht werden (vgl. *6.2.1 Einleitung*). Im Gegensatz zu den Touch-Controllern von Oculus werden die HTC Vive Controller jedoch nicht als virtuelle Hände, sondern als Abbilder ihrer selbst in der virtuellen Umgebung dargestellt (vgl. Janssen 2016). Ohne die virtuellen Hände ist eine Umsetzung von Handgesten in den virtuellen Raum daher nicht möglich.

Leap Motion

Leap Motion ist ein auf Handgesten basierendes Eingabesystem für PCs und Virtual-Reality-Systeme. Bei der Nutzung in Verbindung mit Virtual-Reality-Systemen wird der Leap-Motion-Controller am jeweiligen Headset befestigt und über USB mit dem Computer verbunden. Der Controller besteht aus zwei monochromatischen Infrarotkameras sowie drei Infrarot-LEDs. Seine Reichweite beträgt etwa einen Meter. Innerhalb dieses Bereichs erkennen die Kameras das durch die Hände reflektierte Infrarotlicht und leiten die Informationen an den PC weiter. Dort analysiert und interpretiert die Leap-Motion-Software die Informationen und erzeugt Abbilder virtueller Hände. Das Handtracking der Leap-Motion-Technologie erkennt Hand- und Fingerbewegungen auf 0.7 mm genau (vgl. Weichert et al. 2013:6382f). Damit ermöglicht die Technologie, die standardmäßigen, auf Tastendrücken basierenden Controller durch controllerlose, gestenbasierte Eingaben zu ersetzen.

7 Virtuelle Welten

Als Virtuelle Welten werden die Inhalte von Virtual-Reality-Anwendungen bezeichnet. Virtuelle Welten bestehen aus statischen und dynamischen 3D-Objekten, die auf Nutzereingaben reagieren können. Darüber hinaus enthalten Virtuelle Welten aber auch Klang- und Lichtquellen, virtuelle Kameras sowie Ersatzkörper für die Kollisionsprüfung. Alle genannten Aspekte werden unter Echtzeitbedingungen durch die Grafik- und die Physik-Engine der Entwicklungsumgebung prozessiert. Dazu zählen die Erfassung und Verarbeitung von Nutzereingaben, die Physiksimulation, das Rendering sowie die anschließende Ausgabe auf dem Display (vgl. Dörner et al. 2014:66). Hardwareseitig benötigen Virtual-Reality-Anwendungen leistungsstarke Prozessoren (CPUs) und Grafikkarten (GPU) (vgl. *4.2.1 Head-Mounted-Displays*). Neben der Hardwarespezifikation, müssen stets auch Anforderungen an die virtuelle Welt selbst berücksichtigt werden. Nur dadurch lässt sich eine Überlastung der Hardware verhindern, die zu hohen Latenzen und niedrigen Framerates führen kann. Im Folgenden werden verschiedene Bestandteile von Spieleumgebungen vorgestellt und im Kontext der Anforderungen an virtuelle Welten betrachtet.

7.1 3D-Objekte

3D-Objekte füllen die anfangs leere virtuelle Umgebung mit Inhalten. Sie bilden den Grundstein der Architektur einer Spieleumgebung und haben daher auch den größten Einfluss auf die Performanz einer Virtual-Reality-Anwendung. Je nach Budget und Personal eines VR-Projekts können die 3D-Objekte entweder selbst modelliert oder über Datenbanken im Internet erworben werden.

Objektgeometrie

Für den Einsatz in Virtual-Reality-Anwendungen unterliegen 3D-Objekte bestimmten Auswahlkriterien. Von zentraler Bedeutung ist dabei eine vereinfachte Objektgeometrie, man spricht auch von sogenannten „LowPoly-Modellen“. Der Aufbau eines 3D-Objekts lässt sich als Netz beschreiben, das aus vielen Ecken und Kanten besteht. Die Vereinfachung der Objektgeometrie zielt darauf ab, die Zahl der Ecken (Polygone) eines 3D-Objekts zu reduzieren, damit die Objekte durch die Hardware schneller verarbeitet und dargestellt werden können. Als intelligente Lösung erweisen sich hierbei spezielle 3D-Modelle, die mehrere Varianten des 3D-Objekts in unterschiedlicher Auflösung bereitstellen und diese Varianten in Abhängigkeit der Entfernung des Nutzers zum Objekt darstellen (Level of Detail). Objekte können so auf große Entfernungen stark vereinfacht und die Performanz der Anwendung damit deutlich verbessert werden (vgl. Dörner et al. 2014:67f).

Kollisionsgeometrie

Auch im Bereich der Kollisionserkennung von Objekten wirken sich große Objektgeometrien negativ auf die Performanz aus. Es hat sich daher als sinnvoll erwiesen, 3D-Objekte mit sogenannten Kollisionsgeometrien auszustatten, die sehr einfache geometrische Formen wie beispielsweise die eines Quaders oder einer Kugel annehmen (vgl. Dörner et al. 2014:68).

Materialien und Texturen

Damit 3D-Objekte realitätsnah dargestellt werden können, muss neben der Form des Objekts auch sein Erscheinungsbild angepasst werden. Materialien und Texturen steuern das Erscheinungsbild von 3D-Modellen. Materialien nehmen dabei Einfluss auf die Emissions-, Reflexions- und Transparenzeigenschaften von 3D-Objekten. Sie schreiben der Grafik-Engine also vor, wie Licht, das auf das Objekt fällt, zu verarbeiten ist (vgl. Dörner et al. 2014:75f). Texturen verleihen der Oberfläche von 3D-Objekten ihre Struktur, ohne jedes Detail in der Objektgeometrie modellieren zu müssen.

> *„Texturen sind Rasterbilder, die auf die Objektoberfläche gelegt werden. Die genaue Abbildung von Pixeln der Textur auf Punkte der Objektoberfläche erfolgt über die Zuordnung von normierten Texturkoordinaten (also Rasterkoordinaten) zu den Eckpunktkoordinaten der Polygone einer Oberfläche.“* (Dörner et al. 2014:76f)

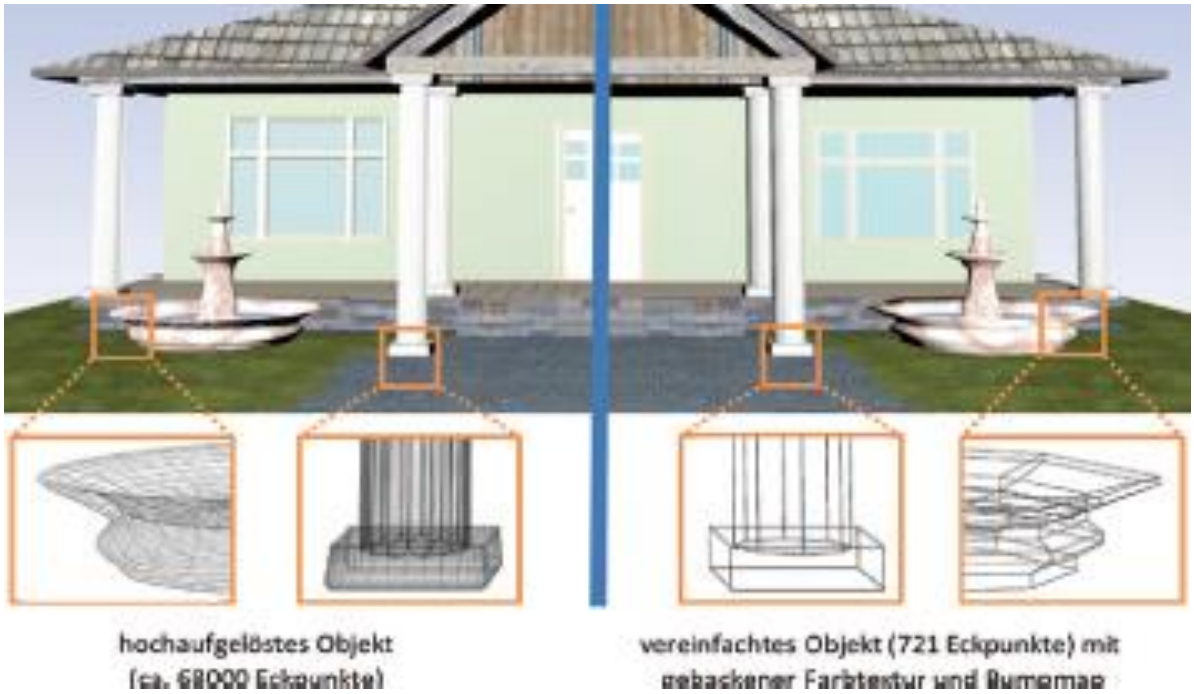

Abb. 17: *Beispiel für Texture Baking. Links: hoch aufgelöste Originalszene. Rechts: Szene mit vereinfachter Geometrie und gebackenen Texturen (Quelle: Dörner et al. 2014:79)*

Mithilfe des sogenannten „Texture Baking“ lassen sich auch LowPoly-Modelle scheinbar detailreich darstellen. Dabei werden aus einem Modell mit vielen Polygonen Texturen erzeugt, die anschließend über das vereinfachte LowPoly-Modell gelegt werden (vgl. Dörner et al. 2014:78f & *Abb. 17*).

7.2 Animation und Objektverhalten

Wenn sich die Eigenschaften von Objekten der virtuellen Welt mit der Zeit ändern, spricht man von animierten Objekten. Dabei spielt es keine Rolle, welche Eigenschaften genau verändert werden. (vgl. Dörner et al. 2014:80).

Physikbasierte Animationen

Die in Kapitel *6.1 Software* vorgestellten Entwicklungsumgebungen besitzen Physik-Engines, die dazu in der Lage sind, Objekte auf Basis physikalischer Parameter zu

animieren. 3D-Objekte müssen dazu mit einer Komponente ausgestattet werden, die sie durch die Physik-Engine verarbeitbar macht. Man nennt diese Komponente Rigidbody (starrer Körper). Die Physik-Engine führt ihre Simulationen unabhängig der Grafik-Engine durch und betrachtet dabei nur Objekte, die entweder mit einem Rigidbody oder einer Kollisionsgeometrie ausgestattet sind. Die physikalischen Eigenschaften von Objekten lassen sich sowohl individuell über die Rigidbody-Komponenten als auch global über die Physik-Engine konfigurieren. Zu den Parametern eines Rigidbodies gehören etwa seine Masse, um die Beschleunigungen bei der Einwirkung von Kräften ermitteln zu können, aber auch materialbezogene Dämpfungsparameter, die über das Reibungsverhalten eines Objekts bestimmen (vgl. Dörner et al. 2014:80f).

Keyframe-Animation

Bei der sogenannten Keyframe-Animation „werden bestimmte Schlüsselzeitpunkte (Keyframes) festgelegt, zu denen eine Eigenschaft des zu animierenden Objektes, beispielsweise seine Position jeweils einen bestimmten Wert (Schlüsselwert) aufweist." (Dörner et al. 2014:80). Die Werte zwischen zwei Keyframes werden durch Interpolation der beiden Schlüsselwerte errechnet (vgl. Dörner et al. 2014:80).

Ereignisbasiertes Objektverhalten

Das Verhalten von 3D-Objekten kann an Ereignisse gekoppelt werden. Wenn ein bestimmtes Ereignis eintritt, führt dies zu einer Zustandsänderung definierter Objekteigenschaften wie Position, Größe, Farbe, Form etc. Ereignisse können dabei durch Kollisionen, Nutzereingaben oder Distanzunterschreitungen ausgelöst werden (vgl. Dörner et al. 2014:82f)

7.3 Beleuchtung und Sound

7.3.1 Beleuchtung

Analog zur Realität sorgen Licht und Beleuchtung auch in der virtuellen Welt dafür, dass die Umgebung für den Nutzer sichtbar wird. In der Regel werden in der Entwicklung von Spieleumgebungen drei verschiedene Typen von Beleuchtung unterschieden, die alle bezüglich Farbe und Intensität angepasst werden können (vgl. Dörner et al. 2014:83):

- Direktionales Licht (Directional Light)
- Punktlicht (Point Light)
- Scheinwerferlicht (Spot Light)

Direktionales Licht simuliert eine unendlich weit entfernte Lichtquelle. Das Licht einer direktionalen Lichtquelle trifft, vergleichbar mit dem Licht der Sonne, überall im gleichen Winkel auf die Umgebung. Da direktionales Licht als eine unendlich weit entfernte Lichtquelle definiert ist, nimmt die Intensität des Lichts mit steigender Distanz nicht ab. Eine Punktlichtquelle strahlt ihr Licht hingegen ähnlich dem einer Glühlampe kugelförmig in alle Richtungen ab. Dabei nimmt die Intensität des Lichts mit steigender Entfernung ab, da es sich auf eine immer größere Kugeloberfläche verteilt. Das Licht einer Scheinwerferlichtquelle strahlt vergleichbar mit dem Licht einer Taschenlampe kegelförmig ab. Bei dieser Lichtquelle nimmt die Intensität sowohl mit steigender Entfernung als auch zum Rand des Lichtkegels hin ab (vgl. Dörner et al. 2014:84).

Komplexe Echtzeitsimulationen der Beleuchtung einer Umgebung wie beispielsweise die Absorption und Reflexion von Licht an Oberflächen stellen hohe Anforderungen an die Performanz eines Systems. Wenn sich die Beleuchtung einer Umgebung kaum bis gar nicht verändert, kann es daher hilfreich sein auf sogenannte „Lightmaps“ zurückzugreifen. Dabei werden Licht, Reflexionen und Schatten vorberechnet und als Texturen gespeichert. Beim Laden einer neuen Spielumgebung werden die Lightmap-Texturen schließlich auf die Umgebung angewendet. Ein Großteil der Berechnungen von Lichteigenschaften, die ansonsten dynamisch zur Laufzeit erfolgen würden, kann so umgangen und die Performanz der Anwendung damit deutlich verbessert werden (vgl. Wikipedia 2018).

7.3.2 Sound

Audioquellen sind häufige Bestandteile von virtuellen Umgebungen. Dabei können die Audioquellen mit der gesamten Umgebung, aber auch in Form von Komponenten mit Objekten verknüpft werden. Über Audioquellen lassen sich Parameter wie Raumklang und Dämpfungseigenschaften des Schalls steuern. Auf Softwareebene wird so die Basis für Richtungshören in der virtuellen Umgebung geschaffen. Problematisch ist jedoch eine physikalisch exakte, echtzeitfähige Berechnung von Schallabsorption, Schallreflexion und Schallbeugung durch Hindernisse. Ähnlich der dynamischen Berechnung des Lichts ergeben sich hierbei hohe Anforderungen an die Performanz des Systems. Die meisten Entwicklungsumgebungen unterstützen daher zwar Raumklang, jedoch keine komplexen Berechnungen zum Verhalten des Schalls im Raum (vgl. Dörner et al. 2014:84f).

7.4 Spezialsysteme

Unter Spezialsystemen sind besondere 3D-Objekte zu verstehen, deren Modellierung und Animation besondere Herausforderungen mit sich bringen. Die wichtigsten dieser 3D-Modelle werden im Folgenden vorgestellt.

7.4.1 Virtuelle Menschen

Virtuelle Menschen sind komplexe 3D-Objekte, die je nach Anwendungsszenario verschiedene Funktionen in virtuellen Welten übernehmen. Häufig fungieren sie als Gegen- oder Mitspieler (Non-Player Character, NPC) und bilden so zentrale Elemente der Interaktion in virtuellen Umgebungen (vgl. Dörner et al 2014:87f). Untersuchungen hierzu belegen einen positiven Einfluss von sozialen Interaktionen mit virtuellen Menschen auf das Präsenzgefühl der menschlichen Nutzer (vgl. Morina et al. 2014:5). Das Zukunftsbild von virtuellen Menschen beschreiben Magnenat-Thalmann und Thalmann (2004:2) wie folgt:

> *„The ultimate research objective is the simulation of Virtual Worlds inhabited by a Virtual Human Society, where Virtual Humans will co-operate, negotiate, make friends, communicate, group and break up, depending on their likes, moods, emotions, goals, fears, etc.“*

Um diesem idealisierten Ziel näher zu kommen, ist die Glaubwürdigkeit virtueller Menschen von zentraler Bedeutung. Bei der Konzeption und Umsetzung gilt es daher vier zentrale Punkte zu beachten (vgl. Magnenat-Thalmann / Thalmann 2004:2):

- Natürliches Verhalten, was Entscheidungsfindung, Intelligenz, Motivation und Sozialverhalten betrifft
- Wahrnehmung: Ausstattung mit Sensorik sowohl für die virtuelle als auch die reale Welt, um auf andere virtuelle Menschen und reale Nutzer reagieren zu können.
- Animation: Natürliche und flexible Bewegungskontrolle
- Grafik: Realistische Darstellung von Körper, Haut, Haaren und Kleidern

7.4.2 Partikelsysteme

Partikelsysteme sind Teil der Grafik-Engine und ermöglichen die Darstellung von Spezialeffekten wie bspw. Feuer, Rauch, Explosionen, Wassertropfen oder Schneeflocken in Virtuellen Welten. Im Unterschied zu 3D-Modellen, denen eine feste Geometrie zugrunde liegt, stellen Partikel unscharfe, sich fortlaufend verändernde Bestandteile virtueller Welten dar. Die Modellierung und Darstellung von Partikeln unterscheidet sich dementsprechend von der Darstellung normaler 3D-Objekte, weshalb eigens dafür entwickelte Systeme in Grafik-Engines implementiert werden (vgl. Reeves 1983:1). Partikelsysteme bestehen aus einer Vielzahl einzelner Partikel, die zur Echtzeit verarbeitet werden. Jedes Partikel ist dabei ein einzelner massebehafteter Punkt, dessen Position im Raum auf Basis einfacher physikalischer Simulationen der auf die Partikel einwirkenden Kräfte aktualisiert wird (vgl. Dörner et al. 2014:89). Zu jedem Zeitpunkt werden dabei:

- *„neue Partikel über einen sog. „Emitter" in das Partikelsystem eingefügt sowie Partikel aus dem Partikelsystem entfernt, deren Lebensdauer abgelaufen ist, oder die einen vordefinierten Aufenthaltsbereich verlassen,*
- *die auf jeden Partikel einwirkenden Kräfte (z.B. Schwerkraft, Wind, Dämpfung) berechnet und daraus die neue Position und Geschwindigkeit der Partikel berechnet,*
- *Visualisierungsattribute wie Farbe und Textur für jeden Partikel aktualisiert und die Partikel visuell dargestellt."* (Dörner et al. 2014:89)

Bezüglich der Anforderungen an virtuelle Welten können große Partikelzahlen oder komplexe Partikelgeometrien einen Mehraufwand für das Rendering erzeugen, der die Echtzeitfähigkeit der Anwendung beeinträchtigen kann (vgl. Dörner et al. 2014:90). Das kann zu hohen Latenzen und niedrigen Framerates führen, die das Präsenzgefühl des Nutzers einschränken oder im schlimmsten Fall zerstören können.

7.4.3 Vegetation

Bäume und Pflanzen kommen in der Realität in sehr komplexen Formen und großen Zahlen vor. Um virtuelle Umgebungen mit Bäumen und Pflanzen auszustatten kommen daher prozedurale Generierungsmethoden zum Einsatz. Mit einer gewissen Varianz können so die unterschiedlichsten Bäume generiert werden. Dabei werden die Bäume schrittweise automatisiert modelliert: Startend mit dem Stamm und groben Ästen, gefolgt von feinen Ästen und den Blättern. Die Blätter stellen in der Regel keine dreidimensionalen polygonbasierten Objekte dar, sondern einfache, zweidimensionale geometrische Formen wie Rechtecke, die mit einer an den Rändern transparenten

Blatttextur ausgestattet werden. Bei polygonbasierten dreidimensionalen Blattmodellen würde die Echtzeitfähigkeit aufgrund fehlender Renderingkapazitäten stark eingeschränkt. Auch eine große Zahl von Bäumen, wie sie beispielsweise in Wäldern auftritt, kann die Performanz einer Anwendung stark beeinträchtigen. Aus diesem Grund werden automatische Optimierungstechniken verwendet, die weiter entfernte oder teilweise verdeckte Bäume als texturierte Vierecke oder vereinfachte Modelle darstellen (vgl. Dörner et al. 2014:92).

8 Konzeption der Anwendung „Climeat Change“

Ein Bestandteil des praktischen Teils dieser Bachelorthesis war die Konzeption einer Virtual-Reality-Anwendung als interaktives Informationsmedium über den Einfluss des Fleischkonsums auf den Klimawandel. Auf Basis der zuvor vorgestellten theoretischen Hintergründe sollen im Folgenden alle konzeptionellen Entscheidungen vorgestellt und erläutert werden.

8.1 Leitmotiv Präsenz

Nach Bailenson (2018) kann ein Erlebnis im virtuellen Raum bei erfüllter Präsenz in all ihren Facetten Emotionen beim Nutzer erzeugen und damit nachhaltig im Gedächtnis bleiben. Ziel der Anwendung „Climeat Change“ ist es, den Nutzer über die Auswirkungen seines Fleischkonsums auf den Klimawandel aufzuklären, möglicherweise sogar einen Wandel seiner Konsumgewohnheiten zu erreichen. Dazu müssen die Inhalte der Anwendung jedoch nachhaltig auf den Nutzer einwirken – er muss bei ihrer Nutzung Emotionen erfahren.

In den vorgestellten theoretischen Grundlagen nahmen Immersion und Präsenz häufig die Rolle eines Kriteriums an, nach dem Technologien, Techniken und Inhalte bewertet werden konnten. Als ein solches Bewertungskriterium stand Präsenz auch bei den konzeptionellen Entscheidungen während der Entwicklung der Anwendung im Vordergrund.

8.2 Auswahl von Hardware und Software

8.2.1 Hardware

Von Seiten der Hochschule standen HTC Vive und Oculus Rift als Virtual-Reality-Systeme für diese Thesis zur Auswahl. Der Vergleich beider Headsets offenbart nahezu identische technische Daten. Lediglich das Field of View der HTC Vive (110°) ist im Vergleich zu dem der Oculus Rift (100°) geringfügig größer (vgl. *6.2.1 Head-Mounted-Displays*). Da sich das Präsenzgefühl mit steigender Größe des Field of View verbessert, besitzt die HTC Vive hier also einen leichten Vorteil gegenüber der Oculus Rift (vgl. 4.2.1 *Head-Mounted-Displays*). Auch bezüglich der Leistung ihrer Trackingsysteme liegen HTC Vive und Oculus Rift nah beieinander: Beide lösen auf den Millimeter genau auf und erzeugen sehr geringe Latenzen. Das Trackingsystem der HTC Vive stellt jedoch mit einer Fläche von maximal $25m^2$ gegenüber $16m^2$ bei der Rift den größeren Trackingbereich zur Verfügung (vgl. *6.2.1 Head-Mounted-Displays*). Für Anwendungen, die auf die Eingabelösung „Room Scale“ und damit primär auf die Lokomotionstechnik „Reales Gehen“ setzen, erweist sich ein großer Trackingbereich als vorteilhaft. (vgl. *4.1.1 Positionstracking*). Der Blick auf die Peripherie beider Systeme offenbart erstmals deutlichere Unterschiede: Während die HTC Vive Controller standardmäßig als Abbilder ihrer selbst in der virtuellen Umgebung dargestellt werden, ermöglicht die Rift mittels kapazitiver Sensoren den Einsatz gestenbasierter virtueller Hände (vgl. *6.2.1 Head-Mounted-Displays*). Virtuelle Hände wirken sich positiv auf das Präsenzgefühl der Nutzer aus, weshalb Oculus an dieser Stelle einen deutlichen Vorteil gegenüber HTC besitzt (vgl. *5.3.1Manipulationstechniken*). Im Headset der Oculus Rift ist weiterhin ein Audiosystem verbaut, das dazu in der Lage ist Raumklang zu erzeugen. Für die HTC Vive existiert zwar ein Äquivalent, es ist jedoch

nicht im Lieferumfang enthalten und muss zusätzlich erworben werden (vgl. *6.2.1 Head-Mounted-Displays*). Damit fehlt bei der HTC Vive in ihrer Standardausführung eine Unterstützung der akustischen Wahrnehmung, was das Präsenzgefühl des Nutzers stören kann (vgl. *3.5.2 Präsenz und Immersion*).
Mit der Präsenz als Leitmotiv fällt die Entscheidung bei der Auswahl der Hardware klar auf das Virtual-Reality-System von Oculus. Sowohl der Einsatz von gestenbasierten virtuellen Händen als auch die Möglichkeit die akustische Wahrnehmung anzusprechen ermöglichen es, den Immersionsgrad zu erhöhen und folglich ein besseres Präsenzgefühl beim Nutzer zu erzeugen. Das kleinere Field of View bleibt aufgrund der Geringfügigkeit des Unterschieds zu vernachlässigen, ebenso wie der im Vergleich zur HTC Vive kleinere Trackingbereich. Letzterer wirkt sich nämlich nur auf Anwendungen aus, bei denen die Eingabelösung „Room Scale“ und damit die Lokomotionstechnik „Reales Gehen“ von zentraler Bedeutung sind. Bei der Anwendung „Climeat Change“ ist das nicht der Fall (vgl. *8.4 Eingabelösung*).

8.2.2 Software

Der Vergleich der vorgestellten Entwicklungsumgebungen Unity und Unreal Engine offenbart zwei Systeme, die sich in keinem der zentralen Elemente zur Anwendungsentwicklung für Virtual Reality nachstehen. Sie zeigen in einzelnen Bereichen ihre individuellen Stärken gegenüber dem Konkurrenten, unter dem Strich sind sie aber als gleichwertig zu betrachten. Eine Auswahl sollte daher auf individuellen und anwendungsspezifischen Präferenzen der Entwickler beruhen. Während sich Unreal Engine beispielsweise durch eine leistungsstarke Grafik-Engine auszeichnet, gilt Unity als besonders nutzerfreundlich und führend was den Multiplattform-Support betrifft. Aus persönlicher Sicht stach Unity gerade aufgrund seiner Nutzerfreundlichkeit auch für Anfänger als zu bevorzugende Entwicklungsumgebung hervor (vgl. *6.1 Software*). Im Verlauf der Anwendungsentwicklung stellte sich außerdem das Virtual-Reality-Framework „Virtual Reality Toolkit“ als leistungsstarke Erweiterung der Entwicklungsumgebung heraus. VRTK liefert objekt- und skriptbasierte Basismodule, die eine einfache Implementierung von vorprogrammierten Interaktionen und Fortbewegungstechniken erlauben (vgl. *6.1 Software*). Die Wahl der Entwicklungsumgebung fiel daher auf Unity.

8.3 Spielverlauf

Auf Basis der zu vermittelnden Inhalte sollte sich die Anwendung „Climeat Change“ in drei Spielumgebungen bzw. vier Szenen gliedern. Die Aufteilung orientierte sich dabei an den in Kapitel *1.1.1 Fleischkonsum und Klimawandel* vorgestellten Aspekten der Fleischerzeugung, die sich unmittelbar auf den Klimawandel auswirken. Da keine Vorerfahrung in der Planung und Entwicklung derartiger Anwendungen bestand, erwies sich das gesetzte Ziel von 3 Spielumgebungen leider als nicht einhaltbar. Im zeitlichen Rahmen von etwa zweieinhalb Monaten Entwicklungsarbeit konnten nur zwei der konzipierten Spielumgebungen bzw. auch nur zwei statt vier Szenen umgesetzt werden. Im Folgenden werden dennoch alle konzeptionell erarbeiteten Szenen vorgestellt:

8.3.1 Szene 1: Supermarkt (Tutorial)

Einführung

Die erste Szene der Anwendung „Climeat Change“ spielt in der Spielumgebung „Supermarkt“. Diese Szene nimmt die Funktion eines Tutorials ein und soll den Nutzer in den Interaktions- und Lokomotionstechniken der Anwendung schulen. Medium des Informationstransfers ist dabei ein virtueller Mensch. Er klärt den Nutzer zunächst über die Hintergründe der Anwendung „Climeat Change“ auf und führt ihn anschließend in die Interaktions- und Lokomotionstechniken ein. Da sowohl die Interaktions- als auch die Lokomotionstechniken unmittelbar mit Controllereingaben verbunden sind, erhält der Nutzer neben den akustischen Informationen auch visuelle Unterstützung durch animierte grafische Darstellungen der Controller. Hiermit wird der Kontext zwischen den Tastenbezeichnungen und den zugehörigen Tasten hergestellt. Die virtuellen Hände des Nutzers werden zu Beginn des Tutorials außerdem als virtuelle Abbilder der Controller dargestellt. Nutzer, die noch keine Erfahrung im Umgang mit den Touch Controllern von Oculus haben, können die notwendigen Tasteneingaben dadurch unmittelbar in der virtuellen Umgebung zuordnen, ohne das Headset absetzen zu müssen. Der Nutzer soll außerdem die Möglichkeit besitzen, im Falle von Unklarheiten, die einzelnen Ausführungen des virtuellen Menschen wiederholen zu können.

Aufgabe

Sobald der Nutzer den einleitenden Teil der Szene durchlaufen hat, gilt es die erlernten Fertigkeiten an einer konkreten Aufgabe zu üben: Der Nutzer soll sich durch den Supermarkt bewegen und bestimmte Einkäufe tätigen. Die Einkäufe platziert er dabei in einem Einkaufskorb, den er im einleitenden Teil der Szene erhalten hat. Für den Einkauf steht dem Nutzer außerdem ein Einkaufszettel zur Verfügung. Dieser kann jederzeit aufgerufen werden und liefert Informationen über die bereits vorhandenen und die noch im Korb zu platzierenden Waren. Da der virtuelle Mensch den Nutzer bei seiner Aufgabe nicht für Rückfragen zur Verfügung steht, kann der Nutzer neben dem Einkaufszettel, auch ein sogenanntes „Cheatsheet“ in seinen Händen aufrufen, auf dem alle für die Interaktion und Lokomotion notwendigen Tasteneingaben nochmals aufgeführt sind. Beim Lösen der Aufgabe wiederholt der Nutzer immer wieder verschiedene Routinen wie die Fortbewegung mittels Teleportation oder die Selektion und Manipulation von Objekten, indem er Einkäufe aus den Regalen nimmt und sie im Korb platziert. Die letzte Ware, die der Nutzer laut Einkaufszettel benötigt sind Steaks. Sobald der Nutzer die Steaks im Korb platziert, findet automatisch eine Überblendung zur zweiten Szene in den tropischen Regenwald statt.

8.3.2 Szene 2: Regenwald

Einführung

Diese Szene spielt in der Spielumgebung „Regenwald“. Hier soll der Nutzer erfahren, in welchem Zusammenhang die Regenwaldrodung mit seinem Fleischkonsum steht. In dieser Szene lernt der Nutzer, welche Relevanz die Agrarflächen im Regenwald für die Fleischproduktion besitzen und in welchem Maße das Klima und die Umwelt dadurch geschädigt werden. Auch in dieser Szene steht dem Nutzer ein virtueller Mensch zur Seite, der ihn in einem Gespräch über diese Zusammenhänge aufklärt. Der Nutzer tritt

dabei in die Rolle des Projektleiters eines Rodungsauftrags im südamerikanischen Regenwald. Der virtuelle Mensch nimmt die Rolle eines Vorarbeiters ein und informiert seinen Vorgesetzten über die Projektfortschritte und Projekthintergründe. Ehe der Nutzer im Spielverlauf fortfahren kann, muss er zunächst alle relevanten Informationen des Gesprächs erhalten haben.

Aufgabe

Nach dem informativen Gesprächsteil der Szene, fordert der Vorarbeiter den Nutzer dazu auf, die Rodung des aktuellen Sektors symbolisch abzuschließen, indem er den letzten verbliebenen Baum mit einer Kettensäge fällt und anschließend die Brandrodung einleitet. An diesem Punkt soll der Nutzer emotional in das Spielgeschehen integriert werden. In der virtuellen Umgebung kann er die unmittelbaren Auswirkungen seines Fleischkonsums erleben, die ihm in der Realität normalerweise verborgen bleiben.

Reflektion

Es folgt eine abschließende Gesprächssequenz, die dem Nutzer die Möglichkeit bieten soll, über die Auswirkungen seines Fleischkonsums zu reflektieren und für sich selbst Lösungen zu finden. Der virtuelle Mensch schlägt hierzu verschiedene Ansätze vor, wie der Nutzer seinen Fleischkonsum anpassen kann, um die Regenwälder und das Klima weniger zu belasten.

8.3.3 Szene 3: Viehzuchtbetrieb

Einführung

Die dritte Szene spielt in der Spielumgebung „Kuhstall“. In dieser Szene geht es darum, den Nutzer vor allem über die konkreten Auswirkungen der konventionellen Fleischerzeugung durch Massentierhaltung aufzuklären. Auch diese Szene basiert auf der sozialen Interaktion zwischen dem Nutzer und einem virtuellen Menschen. Als Praktikant im Viehzuchtbetrieb unterstützt der Nutzer einen Bauern bei der Stallarbeit. Im Gespräch erhält er Informationen über den Futtermittelbedarf, die Methanemissionen der Rinder und weitere Glieder der Wertschöpfungskette, die das Klima belasten. Vor allem das Methangas, das bei der Verdauung der Rinder entsteht und regelmäßig abgerülpst wird, spielt in dieser Szene eine entscheidende Rolle.

Aufgabe

Nachdem das Gespräch zwischen dem Bauern und dem Nutzer beendet ist, erhält der Nutzer die Aufgabe, die Rinder im Stall zu füttern. Dazu muss er verschiedene Futtertröge im Raum befüllen. Die Rinder beginnen aus den gefüllten Trögen zu fressen, im Zuge ihrer Verdauung entsteht Methan, das sie anschließend abrülpsen. Da Methan eigentlich ein unsichtbares und geruchloses Gas ist, wird es in dieser Szene farbig dargestellt. Während der Nutzer weiterhin mit seiner Aufgabe beschäftigt ist, breitet sich das Gas sichtbar im Raum aus und entweicht schließlich nach draußen in die Atmosphäre.

Reflektion

Nach Abschluss der Aufgabe wird der Nutzer erneut dazu eingeladen, über seinen eigenen Fleischkonsum zu reflektieren. Der Bauer steht ihm hierzu mit verschiedenen Vorschlägen zur Seite, wie der Nutzer seinen CO^2-Fußabdruck durch einen Konsumwandel verringern kann.

8.3.4 Szene 4: Supermarkt (Finale)

Nach Abschluss der dritten Szene landet der Nutzer wieder dort, wo seine Reise begann: Vor dem Fleischregal im Supermarkt. Er muss nun den Einkauf beenden, indem er die Waren im Korb an der Kasse bezahlt. An der Kasse trifft der Nutzer wieder auf den virtuellen Menschen aus der ersten Szene. Beim Bezahlen zeigt das Display der Kasse jedoch nicht den Preis der Waren an, sondern die Äquivalente an CO^2-Emissionen, die bei der Produktion der Waren entstanden sind. Hier erhält der Nutzer einen Vergleichswert zwischen Fleischwaren und den anderen Waren aus dem Korb, um die Drastik der Auswirkungen seines Fleischkonsums auf den Klimawandel auch im Vergleich mit anderen Konsumgewohnheiten nochmals zu verdeutlichen. Der virtuelle Mensch bietet dem Nutzer abschließend an, die Fleischwaren zurückzulassen – eine Entscheidung, die der Nutzer erstmals frei treffen kann. Mit der Verabschiedung durch den Kassierer ist das Spiel beendet.

8.4 Eingabelösung

Aus dem inhaltlichen Aufbau der Anwendung „Climeat Change“ folgen Spielumgebungen, die in jedem Fall größer als der maximale Trackingbereich der Oculus Rift sind (vgl. *8.3 Spielverlauf*). Bei der Eingabelösung „Seated“ wird die Spielumgebung in der Regel vollkommen auf den Bereich um den sitzenden Nutzer herum konzipiert, weshalb diese Lösung vorab ausgeschlossen wurde. Die Eingabelösungen „Standing“ und „Room Scale“ gehen in Abhängigkeit von der Größe des Trackingbereichs fließend ineinander über. Da die Oculus Rift einen relativ kleinen Trackingbereich zur Verfügung stellt, der sich nur bedingt für zufriedenstellende Room-Scale-Anwendungen eignet, fiel die Entscheidung daher auf die Eingabelösung „Standing“ (vgl. *4.1.1 Positionstracking*). Je nach Größe des Trackingbereichs in einer konkreten Spielsituation kann sich der Nutzer dabei trotzdem frei bewegen, die Wahl der Eingabelösung „Standing“ impliziert lediglich, dass alle Folgeentscheidungen so getroffen wurden, dass ein Nutzer die Anwendung auch bei einem sehr kleinen Trackingbereich im Stehen spielen kann.

8.5 Navigationskonzept

8.5.1 Lokomotion

Die Lokomotionstechnik „Reales Gehen“ ist bei der Eingabelösung „Standing“ stark eingeschränkt. Ohne vorhandene Hardware wie beispielsweise omnidirektionale Laufbänder, werden alternative Steuerungstechniken zur Bewegungskontrolle notwendig, um eine Fortbewegung durch die Spielumgebungen zu ermöglichen. Für die Eingabelösung „Standing“ passende Lokomotionstechniken sind (vgl. *5.1.1 Steuerungstechniken zur Bewegungskontrolle*):

- Neigen
- Fliegen
- Gehen auf der Stelle
- Controllerbasierte Fortbewegung
- Zeigen und Teleportieren

Sowohl die controllerbasierte Fortbewegung als auch das Fliegen weisen in ihrer Funktionsweise Mängel auf, die zu sensorischen Konflikten und damit zu Cybersickness führen können. Cybersickness schränkt das Präsenzgefühl der Nutzer stark ein. Aus diesem Grund entfielen diese Lokomotionstechniken als mögliche Lösungen für das konkrete Anwendungsszenario (vgl. *5.1.1 Steuerungstechniken zur Bewegungskontrolle*).
Die Lokomotionstechnik „Gehen auf der Stelle“ gilt als natürlichste Art der Fortbewegung unter den oben genannten Alternativen. Der aktuelle Stand der Technik offenbart jedoch Mängel beim Errechnen einer passenden Geschwindigkeit auf Basis der Bewegungsfrequenz der Füße. Bewegt sich der Nutzer nicht mit dem erwarteten Tempo durch die Spielumgebung, so kann sein Präsenzgefühl beeinträchtigt werden. (vgl. *5.1.1 Steuerungstechniken zur Bewegungskontrolle*). Da das Tracking von Beinen und Füßen bei der Oculus Rift nicht möglich ist, müssten die Trackinginformationen außerdem aus den überwachten Controllern, also aus den Handbewegungen beim Gehen abgeleitet werden. Das verwendete Virtual-Reality-Framework VRTK bietet eine derartige Lösung zwar an, nach mehreren persönlichen Tests wirkte sie jedoch sehr unnatürlich. „Gehen auf der Stelle“ wurde deswegen ebenso als mögliche Lokomotionstechnik ausgeschlossen.
Als Letzte der verbliebenen kontinuierlichen Lokomotionstechniken gilt es das „Neigen“ zu bewerten. Die Technik „Neigen“ besitzt einen hohen Grad an Natürlichkeit. Aus diesem Grund geht sie mit einem guten Präsenzgefühl einher. Beim Neigen steuert der Nutzer die Bewegungsrichtung durch seine Blickrichtung und die Geschwindigkeit mithilfe der Stärke seiner Neigung. Unter den kontinuierlichen Fortbewegungstechniken schneidet „Neigen“ beim aktuellen Stand der Technik am besten ab. (vgl. *5.1.1 Steuerungstechniken zur Bewegungskontrolle)*.
Die Lokomotionstechnik „Zeigen und Teleportieren“ gehört im Vergleich zum „Neigen“ zu den instantanen Lokomotionstechniken. Der Technik mangelt es vergleichsweise jedoch an Natürlichkeit, da sich der Nutzer nicht wie bei den kontinuierlichen Lokomotionstechniken entlang eines Vektors durch den Raum bewegt, sondern unmittelbar von einem Punkt im Raum zu einem anderen Punkt im Raum versetzt wird. Darüber hinaus verliert der Nutzer nach jeder Teleportation kurzzeitig die Orientierung – Er muss sich neu orientieren, was dessen Wegfindung beeinträchtigen kann. Auf das Präsenzgefühl des Nutzers wirkt sich diese Störung jedoch nicht aus. Auf der anderen Seite kann die Technik „Zeigen und Teleportieren“ die Wegfindung des Nutzers auch stark vereinfachen. Durch das Beschränken der Teleportation auf wenige erlaubte Zielpunkte, kann dem Nutzer ein konkreter Pfad durch die virtuelle Welt vorgegeben werden. Der ohnehin vorhandene Vorteil bezüglich der Performanz dieser Technik wird dadurch nochmals verbessert. Trotz der vorgestellten negativen Einflüsse auf die Wegfindung, können Nutzer Aufgaben mit der Technik „Zeigen und Teleportieren“ wesentlich schneller erledigen als mit kontinuierlichen Lokomotionstechniken. Auch die allgemeine Präferenz von Nutzern virtueller Welten unabhängig von bestimmten Parametern spricht für den Einsatz der Technik „Zeigen und Teleportieren“ (vgl. *5.1.1 Steuerungstechniken zur Bewegungskontrolle)*.
Obwohl „Neigen“ ein interessanter Ansatz für die kontinuierliche Fortbewegung in Virtuellen Umgebungen darstellt, fiel die Wahl der Lokomotionstechnik auf das instantane „Zeigen und Teleportieren“. Ohne Einflussnahme auf das Präsenzgefühl der Nutzer, dafür jedoch mit deutlichen Vorteilen bezüglich der Performanz zeichnet sich „Zeigen und Teleportieren“ als die passendste Wahl für die Fortbewegung aus.

8.5.2 Wegfindung

Die Lokomotionstechnik „Zeigen und Teleportieren“ ermöglicht es, die Bewegungsfreiheit des Nutzers gezielt auf bestimmte Teleportationsziele zu beschränken (vgl. *5.1.1 Steuerungstechniken zur Bewegungskontrolle)*. Damit lassen sich die Ziele, also bestimmte Punkte im Raum, auch an bestimmte Spielinhalte oder Aufgaben koppeln. Bowman et al. (2005) nennen drei Aufgabestellungen an Nutzer virtueller Welten, die unmittelbar mit der Wegfindung zusammenhängen: Manövrieren, Suche und Exploration. Das Manövrieren bleibt durch die gewählte Lokomotionstechnik weiterhin benachteiligt, da sich der Nutzer nach jeder Teleportation von neuem orientieren muss. Es lässt sich jedoch geringfügig vereinfachen, indem man dem Nutzer eine Blickrichtung nach erfolgter Teleportation vorgibt. Die Aufgabenstellung „Exploration“ entfällt durch die Kopplung der Teleportationsziele an den Spielverlauf vollständig, Jeder neu angesteuerte Zielpunkt ist an neue Inhalte oder Aufgaben gekoppelt. Es besteht also weder der Bedarf noch die Möglichkeit zur freien Erkundung der Spielumgebung. Durch den Aufbau eines linearen Pfads, der die einzelnen Teleportationsziele miteinander verbindet und gewissermaßen den Spielverlauf repräsentiert, können Suchaufgaben außerdem stark vereinfacht werden (vgl. *5.1 Navigation*).
Als Wegfindungskonzept ergibt sich ein aus einzelnen Teleportationszielen bestehender, linearer Pfad, dessen Bestandteile neben Koordinaten im Raum, auch bestimmte Spielinhalte repräsentieren. Analog zum Spielverlauf folgt der Nutzer dem Pfad von Wegpunkt zu Wegpunkt durch die virtuelle Umgebung.

8.6 Selektions- und Manipulationskonzept

Alle in Kapitel *5.2.1 Selektionstechniken* vorgestellten Selektionstechniken dienen dem Zweck, die Selektion von Objekten über große Distanzen zu ermöglichen. Mit der Präsenz als Leitmotiv spielt häufig auch die Natürlichkeit von Techniken und Lösungen im Vergleich zur Realität eine Rolle. Da aus dem Spielverlauf kein Bedarf für eine Selektion über große Distanzen hervorgeht und die vorgestellten Selektionstechniken allesamt als unnatürlich im Bezug zur Realität zu bewerten sind, wurde in der Konzeption der Anwendung auf derartige Techniken verzichtet (vgl. *5.2.1 Selektionstechniken*).
Als Manipulationstechnik boten sich die virtuellen Hände an, die standardmäßig über das Oculus SDK bereitgestellt werden (vgl. *5.3.1 Manipulationstechniken*). Sie sind zu Handgesten fähig, die über Controllereingaben ausgelöst werden (vgl. *6.2.2 Peripherie*). Manipulationsaufgaben, die aus der Konzeption des Spielverlaufs hervorgehen, sehen vor, dass der Nutzer Objekte greifen, bewegen und wieder ablegen können muss. Er verändert dabei die Objektparameter der Rotation und Position des Objektes im Raum (vgl. *8.3 Spielverlauf)*. Um dies möglichst natürlich umzusetzen, wurde die Greifgeste der von Oculus bereitgestellten virtuellen Hände als Selektionsmechanismus ausgewählt. Sobald der Nutzer die Geste in Reichweite eines Objektes auslöst und hält, ist dieses Objekt selektiert. Bis er die Geste wieder auflöst, wird das Objekt an den virtuellen Händen befestigt, sodass er es frei durch den Raum bewegen kann.

8.7 Virtuelle Menschen

Virtuelle Menschen sind das zentrale Element, um soziale Interaktionen in virtuellen Welten zu ermöglichen und damit gleichzeitig das Präsenzgefühl des Nutzers zu

verbessern. Glaubwürdigkeit ist dabei das entscheidende Stichwort, wenn es um die Bewertung virtueller Menschen geht. Bei der Konzeption virtueller Menschen müssen verschiedene Parameter wie Verhalten, Aussehen oder Bewegungen berücksichtigt werden, um eine glaubwürdige soziale Interaktion zu schaffen (vgl. *7.4.1 Virtuelle Menschen*). Aus dem Konzept zum Spielverlauf gehen Dialoge des Nutzers mit virtuellen Menschen hervor (vgl. *8.3 Spielverlauf*). Von zentraler Bedeutung für die Konzeption der virtuellen Menschen sind daher die Parameter: Aussehen, Animation und Verhalten. Die virtuellen Menschen müssen ein zur Spielumgebung passendes Aussehen vorweisen und sich während eines Dialogs mithilfe von Mimik und Gestik möglichst natürlich verhalten. Vor allem natürliche Mundbewegungen spielen bei der Sprachausgabe eine entscheidende Rolle. Auch an die soziale Interaktion selbst besteht jedoch der Anspruch der Natürlichkeit. Aus diesem Grund sollen Nutzereingaben während eines Dialogs über Mikrofoneingaben erfolgen, die in Text umgewandelt und anschließend maschinell verarbeitet werden können (Spracherkennung).

9 Implementierung

In den folgenden Kapiteln wird die technische Umsetzung der konzeptionell erarbeiteten Inhalte beschrieben:

9.1 Aufbau der Spielumgebung

Unity ermöglicht einen einfachen und intuitiven Aufbau von Spielumgebungen. Von Unity unterstützte 3D-Formate werden dazu als sogenannte „Assets“ in die Entwicklungsumgebung geladen und anschließend via Drag’n’Drop in der Scene View platziert. Dort lassen sich die Objekte mithilfe von Werkzeugen verschieben, rotieren und skalieren (vgl. *6.1.1 Unity*). Eine Spielumgebung lässt sich so relativ einfach und je nach Anzahl der 3D-Objekte auch recht schnell aufbauen. Die Auswahl, Nachbearbeitung und Implementierung der 3D-Daten stellt sich jedoch als wesentlich zeitaufwändiger dar, weshalb diese Aspekte im Folgenden genauer betrachtet werden sollen.

9.1.1 Auswahl von 3D-Daten

Damit die Performanz einer Virtual-Reality-Anwendung nicht beeinträchtigt wird, ist die Auswahl passender 3D-Daten für die Spielumgebung von zentraler Bedeutung. Die Geometrie der 3D-Objekte kann die Leistung von Prozessor (CPU) und Grafikprozessor (GPU) stark beeinflussen. Werden diese durch komplexe Objektgeometrien überlastet, so führt dies zu hohen Latenzen und Einbrüchen in der Framerate, was das Präsenzgefühl des Nutzers in der virtuellen Umgebung wiederum stark beeinträchtigt. Bei der Auswahl passender 3D-Objekte gilt es daher stets auf 3D-Modelle mit einfachen Objektgeometrien, sogenannte LowPoly-Modelle zu bauen (vgl. *7.1 3D-Objekte*).
Als weitere Einschränkung galt es ein Budget von rund 100 Euro einzuhalten. Kostenlose 3D-Modelle haben jedoch nur einen geringen Anteil an allen verfügbaren Modellen. Sie sind darüber hinaus häufig veraltet, weshalb ihre Kompatibilität mit Unity nicht mehr oder nur noch teilweise gegeben ist. Die kostenpflichtigen Alternativen hingegen gehen preislich häufig bis in den dreistelligen Bereich. Die Suche nach passenden Modellen erwies sich daher als äußerst aufwändig.

9.1.2 Nachbearbeitung der 3D-Daten

Abb. 18: *Vergleich zwischen (links) veralteten Rohdaten und (rechts) aufbereiteten Rohdaten. (Quelle: Eigene Darstellung)*

Mit Unity kompatible 3D-Formate importieren neben der Objektgeometrie sofern vorhanden auch Materialien und Texturen in das Projekt. Beim Import veralteter 3D-Modelle gehen die Material- und Texturinformationen jedoch regelmäßig verloren. Die Objekte müssen dann mit Materialien und Texturen nachbearbeitet werden. *Abb. 18* zeigt ein veraltetes 3D-Modell vor und nach der Aufbereitung in Unity.

9.1.3 Terraforming

Um Landschaften wie die in der Spielumgebung „Regenwald“ zu erstellen, stellt Unity eine Terrain-Engine bereit, die das Modellieren von Gelände und Vegetation ermöglicht. Mit der Engine lassen sich Berge, Hügel und Täler realisieren, Böden texturieren sowie Bäume und Pflanzen sehr einfach in großer Zahl platzieren. Das Modellieren der Vegetation über die Terrain-Engine geht dabei mit verschiedenen Vorteilen einher. Offensichtlich ist die Zeitersparnis durch das „Bemalen“ des Geländes mit Bäumen und anderen Pflanzen. Die so durch die Terrain-Engine erzeugte Vegetation ist aber auch für das Rendering durch die Grafik-Engine optimiert. Würden stattdessen einzelne 3D-Modelle für die Modellierung der Vegetation verwendet, so wäre die Grafik-Engine beim Rendering überfordert. Dies ginge mit starken Beeinträchtigungen der Performanz einher.

9.2 Implementierung des Oculus Rift-Systems

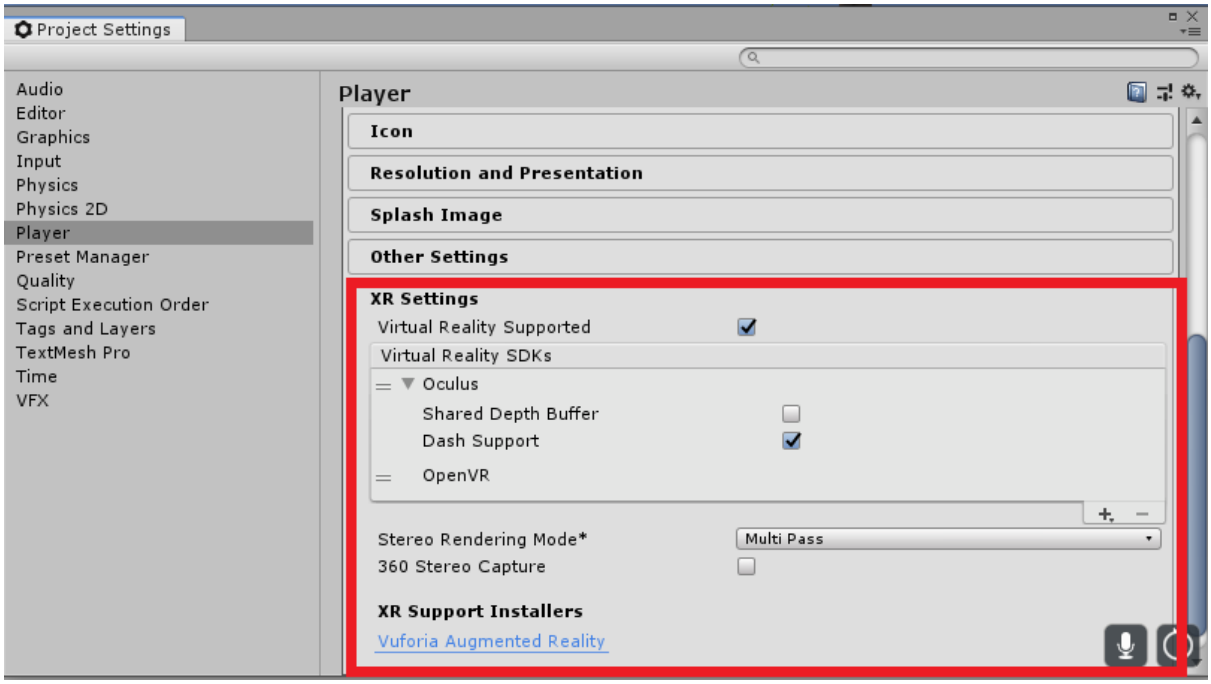

Abb. 19: *Dialogfenster Projekteinstellungen – VR-Schnittstelle aktivieren (Quelle: Eigene Darstellung)*

Unity stellt eine Schnittstelle für Virtual-Reality-Systeme wie Oculus Rift bereit. Nach Aktivierung dieser Schnittstelle steuert das Trackingsystem der jeweiligen Hardware die Position und Rotation des Kameraobjekts in der Szene (vgl. *Abb. 19*). Die Hersteller der Virtual-Reality-Systeme stellen darüber hinaus sogenannte Software-Development-Kits (SDKs) bereit, die die Implementierung des jeweiligen Systems deutlich vereinfachen (vgl. *6.1.1 Unity*). Das Oculus SDK kann kostenlos über das Internet bezogen werden und wird wie jedes andere Paket nach Unity importiert. Dort können schließlich verschiedene Objekte, Skripte und sogenannte Prefabs in der Szene verwendet werden. Ein Prefab ist eine komplexe Hierarchie aus Objekten, die in ihrer Gesamtheit, also im Zusammenspiel aller Objekte miteinander, eine bestimmte Funktion erfüllt. Ein Taschenlampen-Prefab beispielsweise, müsste als Objekte mindestens eine Lichtquelle sowie ein 3D-Modell der Taschenlampe enthalten. Der Komplexität von Prefabs ist jedoch kein Ende gesetzt. Die zentralen Prefabs für die Implementierung des Oculus-Rift-Systems sind das „OVRCameraRig“ und der „LocalAvatar“. Bei der Arbeit mit dem Virtual-Reality-Toolkit spielt darüber hinaus der sogenannte „TrackedAlias“ eine entscheidende Rolle. Dieses Prefab ist eng mit der Implementierung des OVRCameraRigs verbunden und gleichzeitig zentrales Element bei der Implementierung vieler der nachfolgenden Konzepte. Die genannten Prefabs werden im Folgenden vorgestellt.

9.2.1 OVRCameraRig

Das OVRCameraRig ist eine Sammlung von Objekten, die das Headset, die Controller und den Spielbereich in der virtuellen Umgebung repräsentieren. Diese Objekte werden durch die Hardware angesteuert und entsprechend der Trackingdaten hinsichtlich ihrer Rotation und Translation im Raum manipuliert. Ein zentrales Skript („OVRManager“) steuert darüber hinaus die Gesamtfunktionalität des Prefabs. Es kann beispielsweise auf die beim Setup des Oculus-Rift-Systems hinterlegte Größe des Nutzers zugreifen und

damit den Abstand der Kamera zum Boden beim Anwendungsstart entsprechend anpassen. Ohne diese zusätzliche Funktion, müsste die Anwendung auf eine bestimmte Größe des Nutzers ausgelegt werden. Bei Nutzern, deren Körpergröße deutlich von der festgelegten Größe abweicht, könnte dies zu einer Beeinträchtigung des Präsenzgefühls führen.

9.2.2 LocalAvatar

Das LocalAvatar-Prefab ist ein optionaler Bestandteil des OVRCameraRigs. Es stellt die 3D-Modelle der Hände und des Körpers bereit. Bei Einsatz dieses Prefabs werden die Modelle an die Position der jeweiligen Objekte im OVRCameraRig projiziert. Für die virtuellen Hände stehen zwei verschiedene Modelle zur Verfügung: Zum einen 3D-Modelle der Controller und zum anderen 3D-Modelle natürlicher Hände. Die Modelle der Hände werden per Default verwendet, die Controller-Modelle lassen sich über das zum Prefab gehörende OVRAvatar-Skript durch eine Bool'sche Variable aktivieren.

Anwendungsfall: Implementierung beider Handmodelle in einer Szene

In der Tutorial-Szene des Spiels soll der Nutzer zunächst mit den 3D-Modellen der Controller starten. So kann er Informationen, die die Controller betreffen, unmittelbar folgen, ohne das Headset absetzen zu müssen. Um das Präsenzgefühl der Nutzer im weiteren Verlauf der Szene jedoch zu erhöhen, sollen anschließen die natürlichen Handmodelle verwendet werden.

```
34      void OnTriggerEnter(Collider other)
35      {
36          if(other.name == "CollisionFader")
37          {
38              //StartCoroutine(RotateMe2(Vector3.forward * -90, 2));
39              // StartCoroutine(RotateMe(Vector3.forward * 90, 2));
40              if (GameObject.Find("LocalAvatarWithControllers") == true)
41              {
42                  GameObject.Find("LocalAvatarWithControllers").SetActive(false);
43                  LocalAvatarWithHands.SetActive(true);
44              }
45
46
47          }
48
```

Abb. 20: *Funktion zum Umschalten der Handmodelle. (Quelle: Eigene Darstellung)*

Auf den Eingang zum Warenbereich wird hierzu eine simple quaderförmige Kollisionsgeometrie gelegt, die mit dem in *Abb. 20* dargestellten Skript ausgestattet ist. Die Unity-Engine stellt eine Funktion („OnTriggerEnter") bereit, die bestimmte Aktionen auslöst, sobald eine Kollisionsgeometrie von einer anderen durchdrungen wird. Im TrackedAlias ist eine solche Kollisionsgeometrie standardmäßig enthalten. Diese ist zwar Teil eines Prefabs, das eigentlich eine andere Funktion erfüllt, für den vorliegenden Fall wird sie jedoch als eine den Nutzer repräsentierende Kollisionsgeometrie zweckentfremdet.
Sobald der Nutzer den Warenbereich betritt, durchdringen sich die beschriebenen Kollisionsgeometrien. Ursprünglich sollte dann der Wert der entsprechenden Bool'schen Variable geändert werden, um von den Controller-Modellen auf die Handmodelle

umzuschalten. Unglücklicherweise veränderten sich die Modelle bei diesem Setup aber nicht. Es stellte sich heraus, dass die Modelle beim Laden der Szene einmalig initialisiert werden und während der Szene nicht umgeschaltet werden können. Um dieses Problem zu umgehen, wurden dem OVRCameraRig zwei LocalAvatar-Prefabs hinzugefügt. Eines initialisiert die Handmodelle, das andere die Controller-Modelle. Sobald sich die Kollisionsgeometrien durchdringen wird das Prefab mit den Controller-Modellen („LocalAvatarWithControllers“) deaktiviert und das Prefab mit den Handmodellen („LocalAvatarWithHands“) aktiviert (vgl. *Abb. 20*). Mit diesem Workaround sind beide Modelle innerhalb einer Szene nutzbar.

9.2.3 TrackedAlias

Abb. 21: *Vergleich zwischen TrackedAlias und OVRCameraRig. (Quelle: Eigene Darstellung)*

Der TrackedAlias fungiert als Stellvertreter für alle in die Szene implementierten CameraRigs. In seinem Aufbau ähnelt er diesen daher deutlich: Für jedes getrackte Objekt eines CameraRigs besitzt der TrackedAlias einen Stellvertreter (vgl. *Abb. 21*). Beim Start einer Anwendung erkennt er automatisch das durch die Hardware angesprochene CameraRig. Er spiegelt dann die Positions- und Rotationsinformationen der Objekte des CameraRigs auf die jeweiligen Stellvertreterobjekte im TrackedAlias-Prefab. Der TrackedAlias ist also nichts anderes als die gespiegelte Objekthierarchie des aktiven CameraRigs. Für die Anwendungsentwicklung von Bedeutung ist er, weil viele der Konzepte, die im Folgenden implementiert werden, mit dem TrackedAlias zusammenhängen. Sollen die virtuellen Hände beispielsweise um einen Zeigestrahl erweitert werden, so finden diese Erweiterungen immer am TrackedAlias statt. So können verschiedene Virtual-Reality-Systeme über ihre jeweiligen CameraRigs in die Szene implementiert werden, während die Implementierung von nachfolgenden Konzepten auf Basis dieses Stellvertreter-Prefabs vollzogen wird.

9.3 Implementierung der Navigation

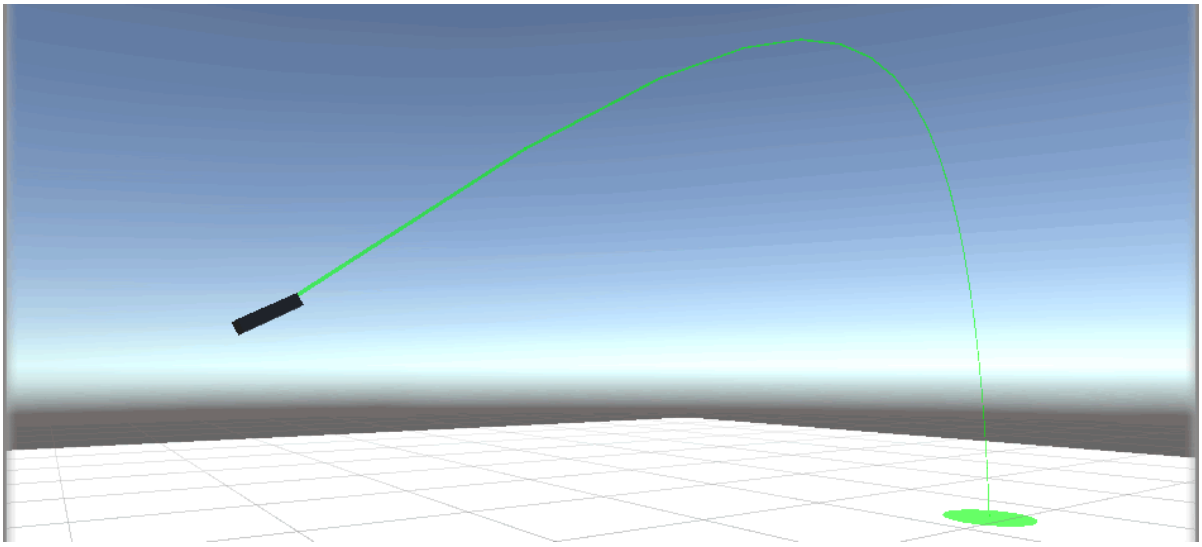

Abb. 22: Zeigewerkzeug mit parabelförmigem Zeiger (Quelle: Eigene Darstellung)

Das Virtual-Reality-Toolkit stellt verschiedene Lokomotionstechniken für Virtual-Reality-Anwendungen bereit, darunter auch die bei der Konzeption der Anwendung ausgewählte Technik „Zeigen und Teleportieren“. Die Technik setzt sich aus zwei Bestandteilen des VRTK zusammen, die es bei der Implementierung zu vereinen gilt. Der Nutzer benötigt zunächst ein sichtbares Zeigewerkzeug, mit dem er den Zielpunkt der Teleportation bestimmen kann. Dieses Zeigewerkzeug ist ein parabelförmiger Zeiger, der an den virtuellen Händen des Nutzers entspringt und dessen Ende auf die Umgebung trifft und damit den Zielpunkt der Teleportation markiert (vgl. *Abb. 22*). Das Zeigewerkzeug muss anschließend mit der Teleportationsmechanik des Virtual-Reality-Toolkits verbunden werden.

9.3.1 Implementierung des Zeigewerkzeugs

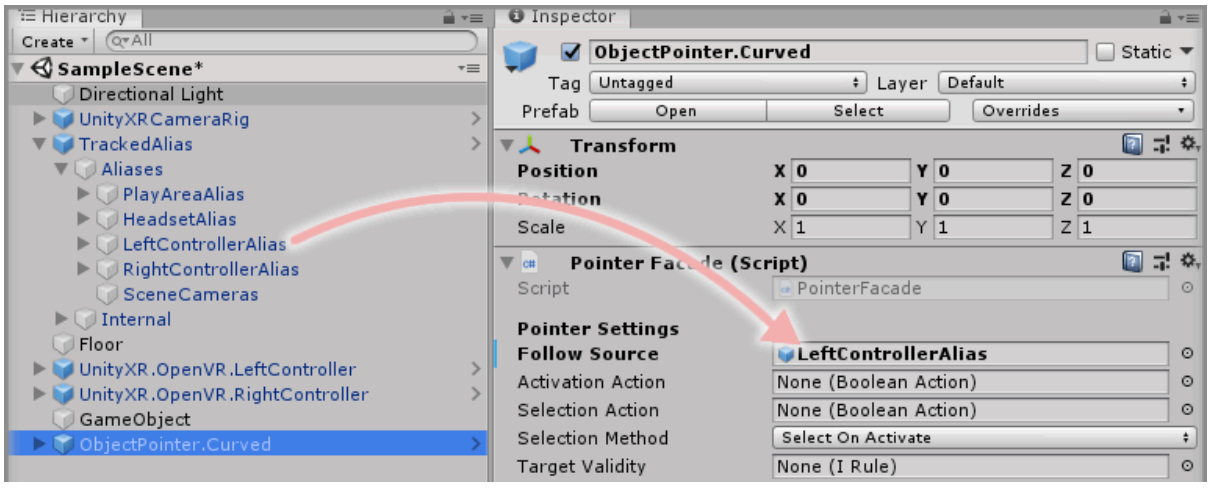

Abb. 23: Zuordnung des Zeigerursprungs. (Quelle: Eigene Darstellung)

Im Virtual-Reality-Toolkit ist ein Prefab für parabelförmige Zeiger („ObjectPointer.Curved“) bereits vorhanden. Dieses Prefab wird per Drag’n’Drop in die Szene implementiert. Das Hauptskript des Zeiger-Prefabs („Pointer Facade“) fordert die Festlegung eines Zeigerursprungs. Im konkreten Anwendungsfall soll der Zeiger seinen

Ursprung an den virtuellen Händen des Nutzers haben. Für jede virtuelle Hand wird deswegen je ein Zeiger-Prefab in die Szene implementiert. Die Controller-Objekte im TrackedAlias werden anschließend als jeweiliger Zeigerursprung definiert. (vgl. *Abb. 23*).

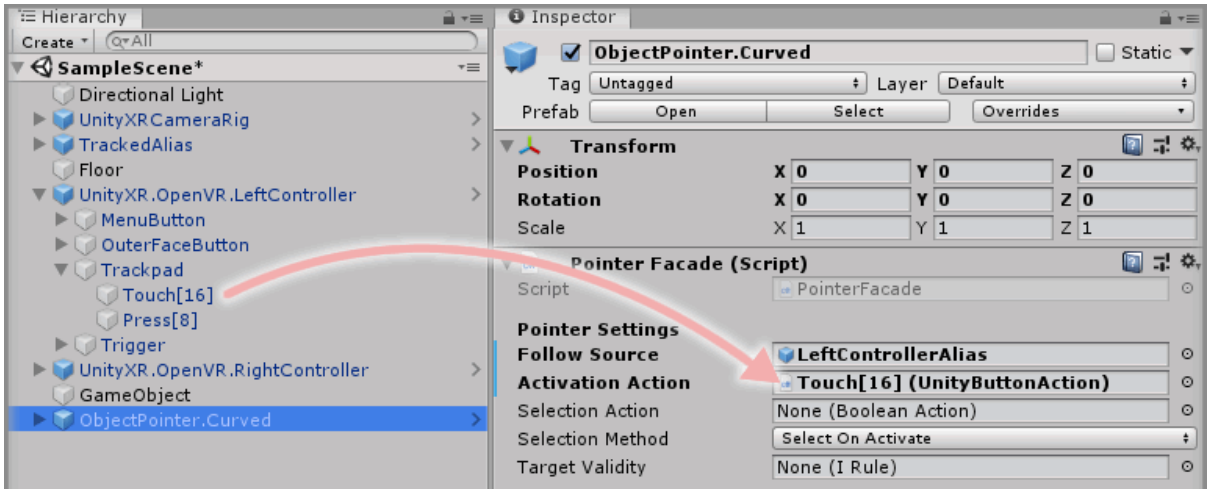

Abb. 24: *Zuordnung des Aktivierungsereignisses des Zeigewerkzeugs. (Quelle: Eigene Darstellung)*

Da der Zeiger nur benötigt wird, wenn sich der Nutzer fortbewegen möchte, ist das Zeiger-Prefab standardmäßig deaktiviert. Über das Skript kann aber eine Aktion („Activation Action“) festgelegt werden, bei deren Eintreten sich der Aktivierungsstatus des Prefabs ändert und eine weitere Aktion („Selection Action“) bei deren Eintreten das Skript die Positionsdaten des ausgewählten Zielpunktes ausgibt. Mithilfe der kapazitiven Sensoren der Touch-Controller lässt sich feststellen, ob ein Finger auf einer bestimmten Taste ruht. Es erweist sich daher als sinnvoll, den Aufruf des Zeigers an die Berührung einer Taste und die Ausgabe der Positionsdaten an den zugehörigen Tastendruck zu koppeln. Das Virtual-Reality-Toolkit stellt hierfür spezielle Skripte („UnityButtonAction“) zur Verfügung, die Tasten überwachen können und deren Status als Bool’sche Variable ausgegeben. Diese Skripte werden mit den oben beschriebenen Aktionen verknüpft. Der Wert der Bool’schen Variable bestimmt fortan über das Auslösen einer dieser Aktionen. (vgl. *Abb. 24*). Für die Anwendung „Climeat Change“ wurde das Berühren des Daumensticks als Auslöser des Zeiger-Prefabs und das Drücken des Daumensticks als Auslöser für die Ausgabe der Positionsdaten definiert.

9.3.2 Implementierung der Teleportationsmechanik

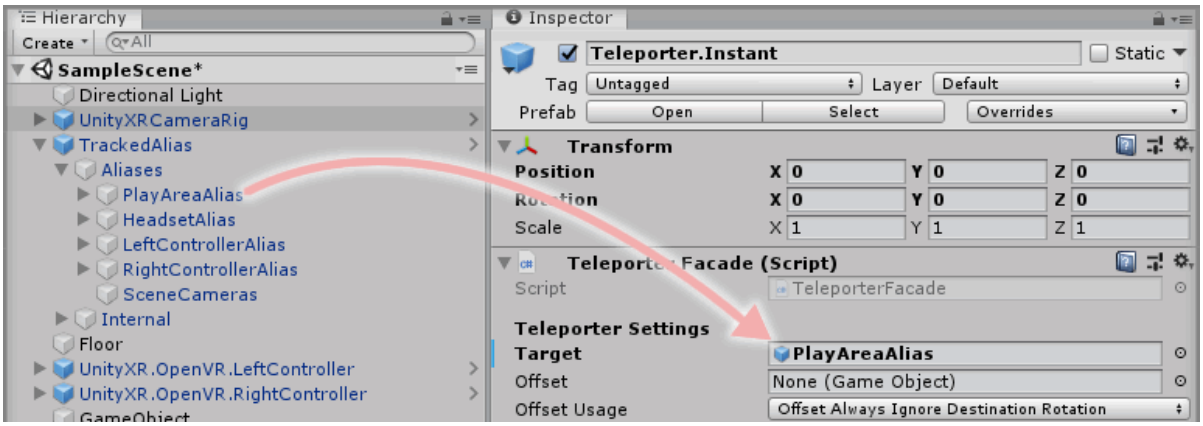

Abb. 25: *Zuordnung des zu teleportierenden Objekts. (Quelle: Eigene Darstellung)*

Auch für die Teleportationsmechanik existiert ein eigenes Prefab („Teleporter.Instant“), das via Drag’n’Drop in die Szene implementiert und anschließend konfiguriert wird. Das Hauptskript des Teleport-Prefabs („Teleporter Facade“) fordert die Zuweisung eines zu teleportierenden Zielobjekts („Target“). Das Headset-Objekt im TrackedAlias, eignet sich nicht für diese Zuweisung. Es aktualisiert kontinuierlich seine Position in Relation zum jeweiligen Spielbereich. Würde man dem Skript dieses Objekt zuweisen, so würde seine Position unmittelbar nach erfolgter Teleportation wieder überschrieben werden. In diesem Fall muss also das Objekt, das den Spielbereich repräsentiert, als Zielobjekt verwendet werden (vgl. *Abb. 25)*.

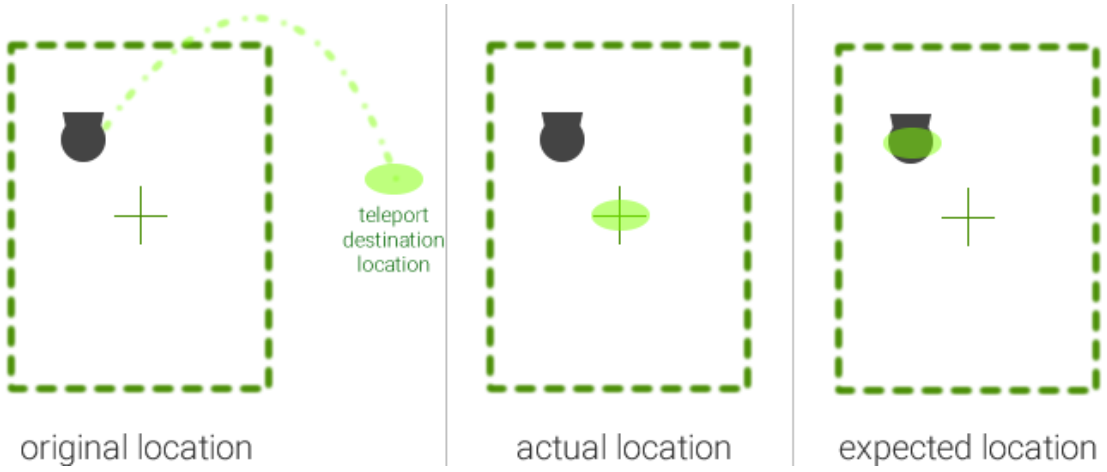

Abb. 26: *Visuelle Darstellung der auftretenden Problematik beim Implementieren der Teleportationsmechanik (Quelle: VRTK Academy o.J.)*

Bei der bisherigen Konfiguration wird das durch den Zeiger bestimmte Teleportationsziel zum Zentrum des Spielbereichs. Befindet sich der Nutzer selbst nicht in diesem Zentrum, so findet er sich nach der Teleportation versetzt zu dem durch ihn gewählten Zielpunkt wieder. Er erwartet jedoch, genau an die Stelle teleportiert zu werden, die er auch mit dem Zeiger ausgewählt hat. Das Ergebnis widerspricht also den Erwartungen des Nutzers, was dessen Präsenzgefühl und Wegfindung möglicherweise beeinträchtigen kann (vgl. *Abb. 26*). Das Skript der Teleportationsmechanik kann dies

jedoch ausgleichen, indem es den Spielbereich nach erfolgter Teleportation so verschiebt, dass der Nutzer sich am erwarteten Punkt wiederfindet. Diesen Versatz errechnet das Skript aus der relativen Position des Headsets zum Zentrum des Spielbereichs. Es gilt also das Headset-Objekt im TrackedAlias mit dem Skript der Teleportationsmechanik zu verknüpfen, damit das Skript besagten Versatz ermitteln kann (vgl. *Abb. 27*).

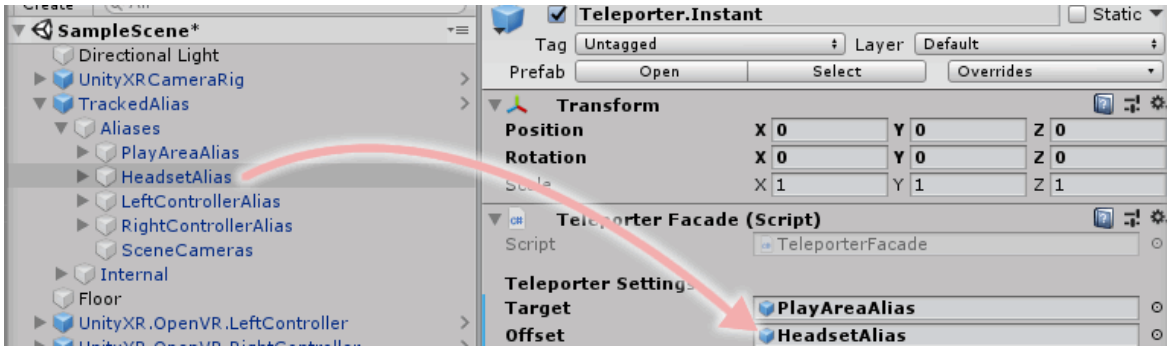

Abb. 27: *Zuordnung des Headset-Objekts zur Ermittlung des Versatzes des Headsets zum Zentrum des Spielbereichs (Quelle: Eigene Darstellung)*

9.3.3 Verknüpfung von Zeiger und Teleportationsmechanik

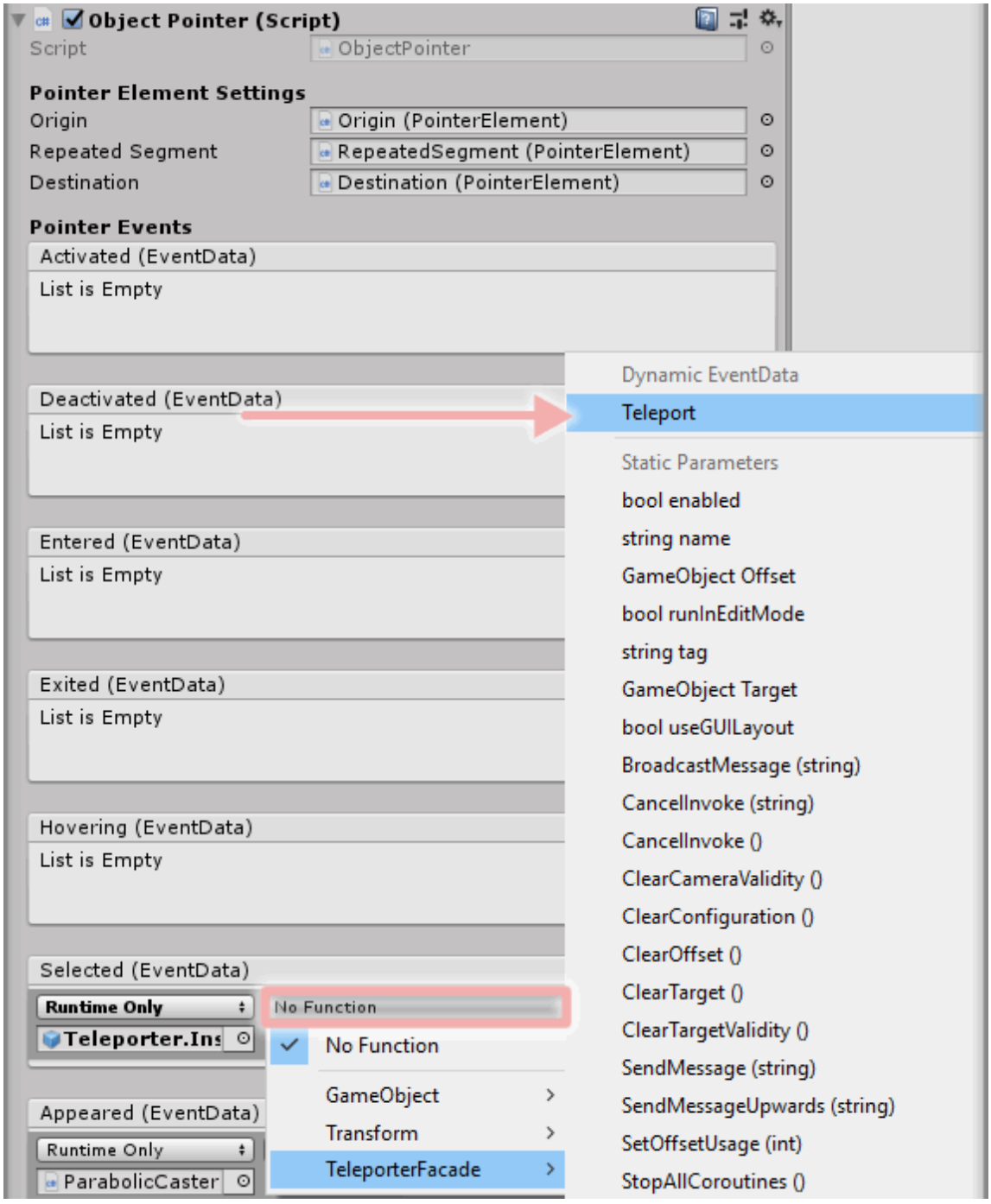

Abb. 28: *Übergabe der Zielkoordinaten an die Teleportationsmechanik (Quelle: Eigene Darstellung)*

Nach der Implementierung des Zeigers und der Implementierung der Teleportationsmechanik gilt es beide Konzepte miteinander zu verbinden. Das Zeiger-Prefab gibt bei Betätigen des Daumensticks die Positionsdaten des Zielpunktes aus. Diese Positionsdaten müssen an die Teleportationsmechanik übermittelt werden. Mit der Ausgabe der Positionsdaten ist ein Ereignis („Selected“) in einem Subskript des Zeiger-Prefabs verknüpft. Für das Eintreten dieses Ereignisses können bestimmte Reaktionen, wie das Auslösen der Teleportation definiert werden. Dazu wird die Teleportationsmechanik via Drag’n’Drop mit diesem Ereignis verknüpft und die Funktion

„Teleport“ im Skript der Teleportationsmechanik immer dann aufgerufen, wenn das Ereignis eintritt (vgl. *Abb. 28*). Die Positionsdaten des Zielpunktes werden schließlich an die Funktion übermittelt und die Teleportation ausgelöst.

9.3.4 Beschränkung der Teleportationsziele

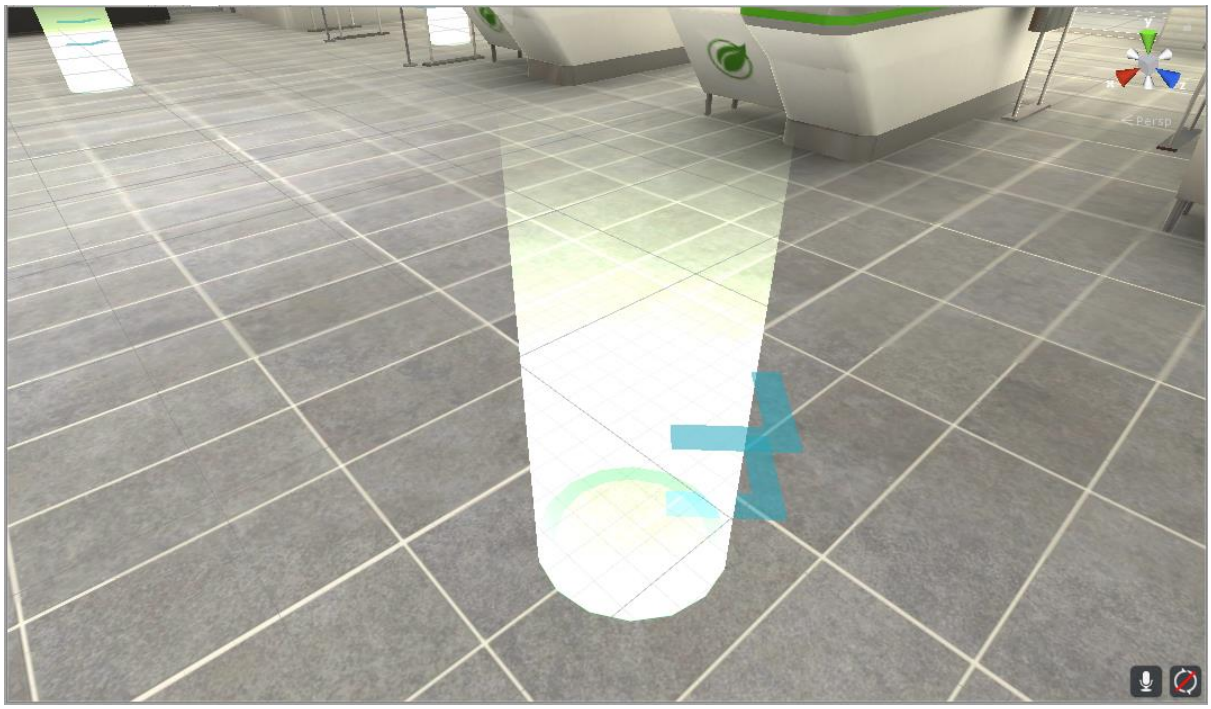

Abb. 29: *Teleportationsziel mit Spezifikation der Blickrichtung*

Mit der zuvor implementierten Lokomotionstechnik ist es dem Nutzer nun möglich, sich an jeden beliebigen Punkt der Spielumgebung zu teleportieren. Das Navigationskonzept fordert jedoch eine Beschränkung der Teleportationsziele (vgl. *8.5.2 Wegfindung*). Das Virtual-Reality-Toolkit stellt dazu ein Skript bereit, das eine solche Beschränkung ermöglicht. Alle Objekte, die mit diesem Skript ausgestattet sind, können fortan nicht mehr als Ziel des Zeigers ausgewählt werden. Der Zeiger färbt sich dann rot, bei Betätigen des Daumensticks findet keine Teleportation statt. Indem alle Objekte mit diesem Skript ausgestattet werden, wird die Lokomotion zunächst vollständig beschränkt. Anschließend werden spezielle Teleportationsziele des Virtual-Reality-Toolkits in die Szene implementiert. Wie im Navigationskonzept gefordert, sind diese zudem dazu in der Lage, die Blickrichtung des Nutzers nach erfolgter Teleportation vorzugeben (vgl. *Abb. 29*). Mit den Zielen lassen sich außerdem die für die Wegfindung gewünschten Pfade realisieren (vgl. *8.5.2 Wegfindung*).

9.4 Implementierung der Selektions- und Manipulationstechniken

Um den durch das Selektions- und Manipulationskonzept geforderten Greifmechanismus zu implementieren, liefert das Virtual-Reality-Toolkit zwei sich ergänzende Prefabs. Zum einen das Interactor-Prefab und zum anderen das Interactable-Prefab. Ein Interactor-Prefab kann mit jedem beliebigen Interactable-Prefab interagieren. Jedes Objekt einer Szene kann zwar zum Interactor werden, in der Regel werden jedoch die Controller-Objekte im TrackedAlias jeweils mit einem Interactor-Prefab ausgestattet. Als Kindelement eines Controller-Objekts hat ein Interactor standardmäßig dessen

Raumkoordinaten. Er bewegt sich also mit den Controller-Objekten durch den virtuellen Raum. Ein Interactor kann mit jedem Interactable-Prefab in einer Szene interagieren. Damit Interactables sichtbar sind müssen sie lediglich mit einem passenden 3D-Modell verknüpft werden. Die Implementation einer Greifmechanik mithilfe von Interactors und Interactables wird im Folgenden erläutert.

9.4.1 Implementierung eines Interactors

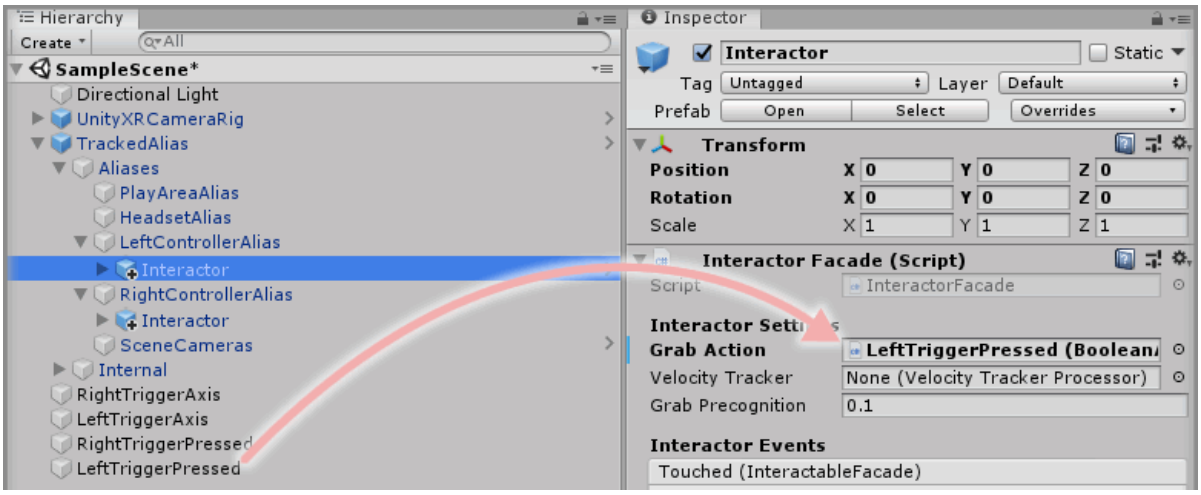

Abb. 30: *Zuordnung des Aktivierungsereignisses der Greifmechanik. (Quelle: Eigene Darstellung)*

Die Interactor-Prefabs werden via Drag'n'Drop als Kindelemente der Controller-Objekte in die Szene eingefügt. Über das Hauptskript des Interactor-Prefabs („Interactor Facade") kann eine Aktion definiert werden, die das Greifen bei entsprechenden Rahmenbedingungen auslöst. Auslöser des Greifens soll im konkreten Anwendungsfall das Betätigen der Greiftaste eines Controllers sein. Dazu kommen erneut Skripte zum Einsatz („BooleanAction"), die den Status einer Taste als Bool'sche Variable ausgeben. Diese Skripte werden jeweils mit dem Hauptskript des zugehörigen Interactors verknüpft (vgl. *Abb. 30*). Der Wert der Bool'schen Variable bestimmt dann darüber, ob die auslösende Aktion erfüllt oder nicht erfüllt ist. Damit der Nutzer gegriffene Objekte eventuell auch werfen kann, muss weiterhin der Geschwindigkeitsvektors des jeweiligen Controllers zum Zeitpunkt des Loslassens an das gegriffene Objekt übergeben werden. Im Skript des Interactors kann dafür der Parameter „Velocity Tracker" belegt werden. Indem man das jeweilige Controller-Objekt mit diesem Parameter verknüpft, kann das Skript die Geschwindigkeitsvektoren der Controller auslesen. Beim Loslassen eines gegriffenen Objekts wird die Geschwindigkeit dann an das Objekt übertragen.

9.4.2 Implementierung von Interactables

Das Virtual-Reality-Toolkit stellt verschiedene Interactable-Prefabs bereit, die sich alle in ihrer Greifmechanik unterscheiden. Es gibt Interactables, die von einer in die andere Hand übergeben werden können, aber auch solche die nur mit beiden Händen gegriffen werden können. Aufgrund der modularen Architektur der Prefabs, können beliebige benutzerdefinierte Varianten für spezifische Anwendungsfälle konfiguriert und implementiert werden. Welches Interactable-Prefab dabei wann zum Einsatz kommt, bestimmt stets der jeweilige Kontext. Soll beispielsweise ein großer Stein bewegt werden, so erweist es sich als sinnvoll, diesen nur mit beiden Händen bewegen zu können.

Auch die Interactables werden per Drag'n'Drop in die Szene implementiert. Sie sind mit einer Rigidbody-Komponente ausgestattet, die sie durch die Physik-Engine beeinflussbar macht. Aus diesem Grund benötigen sowohl die verknüpften 3D-Modelle als auch der Boden der Spielumgebung sogenannte „Collider" (Kollisionsgeometrien). Ohne die Collider fiele das jeweilige Objekt nämlich zunächst durch den Boden hindurch und schließlich endlos weiter. In der Hierarchie eines Interactable-Prefabs findet sich ein funktionsloses Objekt („Meshes"). Ein 3D-Modell, das durch einen Interactor greifbar sein soll, muss als Kindelement dieses Objekts in der Hierarchie platziert werden. Ist das verknüpfte Modell mit einem Collider ausgestattet, so ist das Interactable-Prefab damit schon durch einen Interactor greifbar. Ausgelöst wird das Greifen eines Objekts nämlich durch das Betätigen des im Interactor definierten Auslösemechanismus, während die Collider von Interactor und Interactable sich durchdringen.

9.5 Implementierung virtueller Menschen

Um glaubwürdige und natürliche virtuelle Menschen in eine Virtual-Reality-Anwendung zu implementieren, ist es notwendig, den Anforderungen an virtuelle Menschen im Einzelnen, aber auch in einem sinnhaften Zusammenspiel miteinander gerecht zu werden. Die Implementierung der einzelnen Anforderungen wird im Folgenden vorgestellt. Abschließend wird die Verknüpfung dieser Anforderungen zu einem sinnhaften Ganzen näher erläutert.

9.5.1 Modelle virtueller Menschen

3D-Modelle von virtuellen Menschen, die die konzeptuellen Anforderungen hinsichtlich eines möglichst realistischen Aussehens erfüllen, finden sich überwiegend in einem Preissegment wieder, das den Budgetrahmen dieser Thesis deutlich überschreitet. Animationen darüber hinaus selbst zu erstellen, die auch den Anforderungen an eine möglichst natürliche Gesprächssituation entsprechen, wäre im zeitlichen Rahmen dieser Arbeit kaum möglich gewesen. Glücklicherweise konnten alle Anforderungen aber mithilfe der Plattformen Adobe Fuse und Mixamo zufriedenstellend erfüllt werden. Adobe Fuse, ist eine Software, mit der einfach und schnell benutzerdefinierte 3D-Modelle virtueller Menschen erstellt werden können. Die zugehörige Animationsdatenbank Mixamo, stellt für diese Modelle unzählige kostenlose Animationen bereit. Da Adobe Fuse als Bestandteil der Creative Cloud jedoch einem kostenpflichtigen Abonnement unterliegt, wurde im Rahmen dieser Thesis auf zwei Default-Modelle von Adobe Fuse zurückgegriffen, die kostenlos über Mixamo bereitgestellt werden. Beide Modelle sind einigermaßen passend für ihre jeweilige Spielumgebung.

9.5.2 Animationen

Die Standardmodelle von Adobe Fuse besitzen volle Kompatibilität zu den Animationen der kostenlosen Animationsdatenbank Mixamo. So kann schnell und einfach ein Pool an passenden Animationen für die Gestik zusammengestellt werden, ohne selbst animieren zu müssen. Die Animationen von Mimik und Mundbewegungen bei der Sprachausgabe werden mithilfe des Unity-Plug-Ins „LipSync" umgesetzt. Ein Algorithmus des Plug-Ins kann Laute gesprochener Sprache in Audiodateien erkennen und diesen Lauten die passende Mundstellung zuweisen. Die genaue Implementierung der Animationen für Gestik, Mimik und Sprachausgabe wird im Folgenden näher erläutert.

Gestik

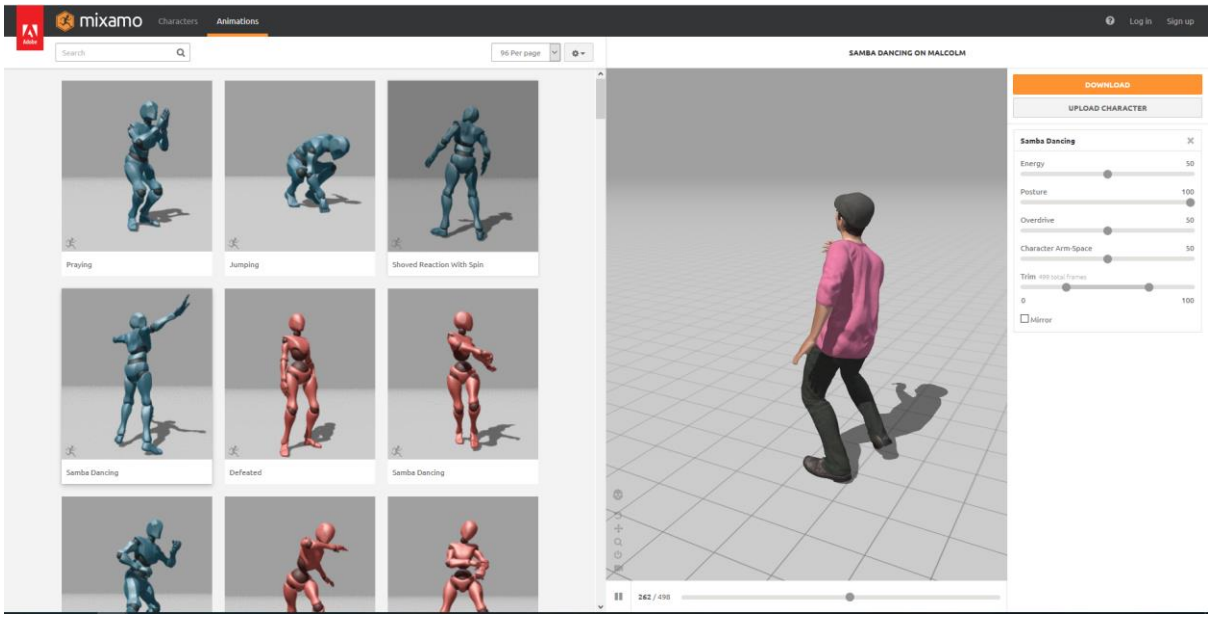

Abb. 31: *Screenshot der Mixamo Webseite (Quelle: Eigene Darstellung)*

Modelle virtueller Menschen, die mit Adobe Fuse erstellt wurden, können über die Animationsdatenbank Mixamo kostenlos mit Animationen ausgestattet und anschließend heruntergeladen werden. (vgl. *Abb. 31*). Die Animationen werden dann nach Unity importiert und dort zu komplexen Animationsabfolgen zusammengefügt. Unity stellt dazu ein Tool bereit, mit dem sich Objekte („Animator Controller“) erstellen lassen, die Vorgaben zum Start, zum Ende und zur Abfolge von Animationen enthalten. Virtuelle Menschen können anschließend mit einer Animator-Komponente ausgestattet werden, die mithilfe eines verknüpften Animator Controllers die Animationen des virtuellen Menschen steuert.

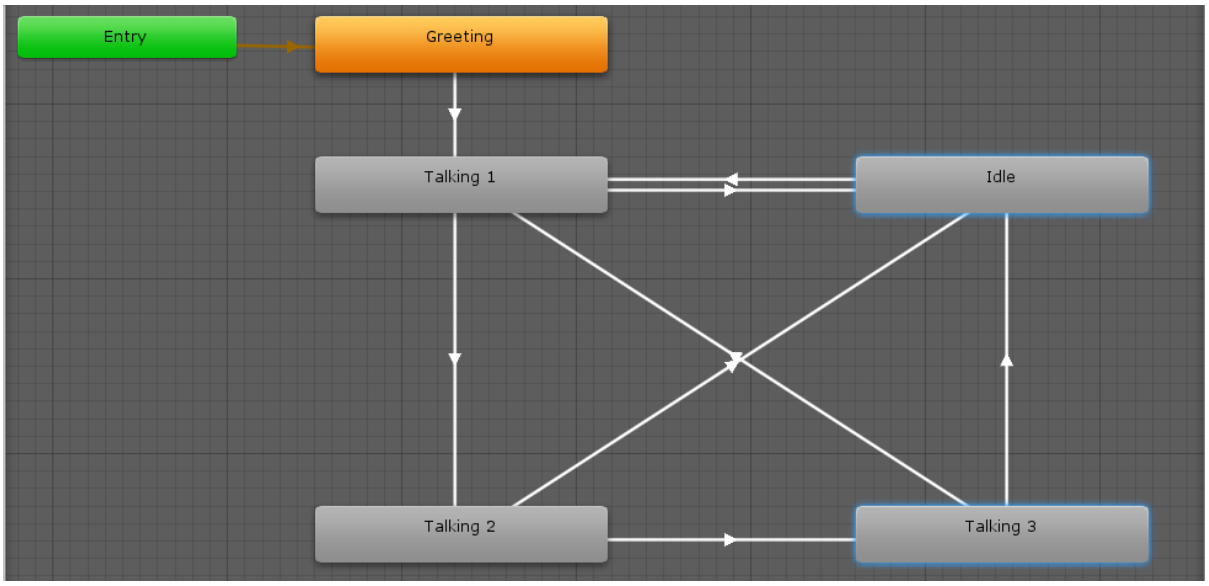

Abb. 32: *Überblick: Animator-Controller. (Quelle: Eigene Darstellung)*

Abb. 32 zeigt den Aufbau eines solchen Animator-Controller-Objekts. Die Rechtecke repräsentieren dabei einzelne Animationen, die Verbindungspfeile deren Abfolge.

Mithilfe von Variablen lassen sich Bedingungen definieren, die bestimmen, wann ein spezifischer Animationspfad durch den Animator gewählt werden soll. Der Pfad von der Animation „Talking 1“ zur Animation „Talking 2“ wird beispielsweise nur beschritten, wenn die im Animator Controller hinterlegte Bool’sche Variable den Wert „true“ annimmt. Ist das Gegenteil der Fall, so beschreitet der Animator den alternativen Pfad zur Animation „Idle“. Die Variablen lassen sich später über Skripte ansteuern und verändern. So können Animationen oder Animationsschleifen durch bestimmte Ereignisse, wie beispielsweise beim Einsetzen der Sprachausgabe des virtuellen Menschen, ausgelöst werden.

Mimik und Sprachausgabe

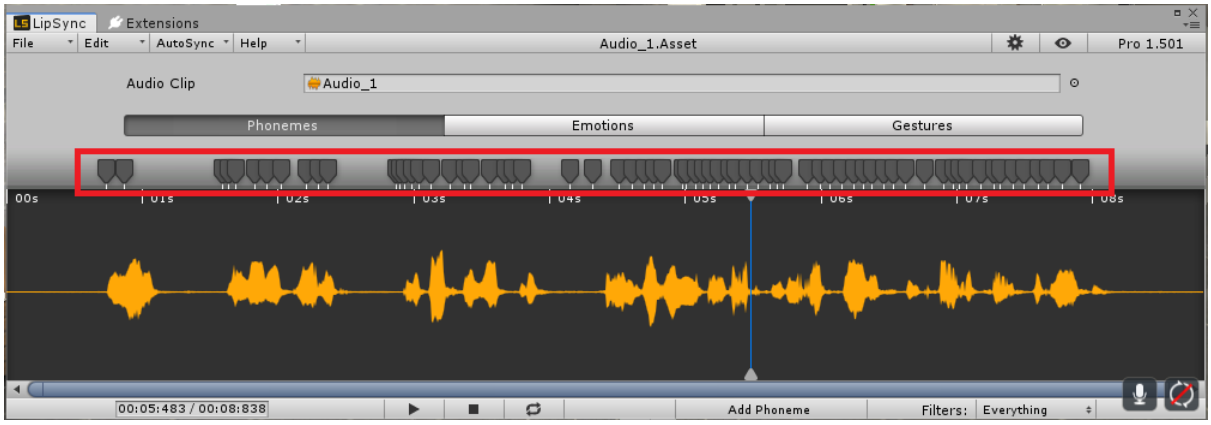

Abb. 33: *LipSync-Oberfläche mit zu einer Audiodatei zugeordneten Lauten. (Quelle: Eigene Darstellung)*

Mit dem Unity-Plug-In „LipSync“ kann die Sprachausgabe und Mimik eines virtuellen Menschen animiert werden. LipSync besitzt eine zu den Modellen von Adobe Fuse kompatible Datenbank mit verschiedenen Stellungen des Mundes. Ein spezieller Algorithmus ist dazu in der Lage, Audiodateien, die gesprochene Sprache enthalten, zu analysieren, die entsprechende Mundstellung zu ermitteln und diese mit dem entsprechenden Zeitpunkt in der Audiodatei zu verknüpfen. Jedem Zeitpunkt in einer kompatiblen Audiodatei ist nach der Verarbeitung durch den Algorithmus also eine passende Mundstellung zugeordnet (vgl. *Abb. 33*). Die Sprachausgabe eines virtuellen Menschen lässt sich durch LipSync außerdem um Mimik ergänzen. Hierzu stellt das Plug-In Animationen in Form von Emotionen bereit, die über einen beliebigen Zeitraum in einer Audiodatei manuell zugeordnet werden können. Alle beschriebenen Informationen werden schließlich in einem eigenen Dateiformat gespeichert. Virtuelle Menschen können dann um eine LipSync-Komponente ergänzt werden, die dieses Dateiformat lesen und verarbeiten kann. Wird eine solche Datei beispielsweise durch ein Skript aufgerufen, dann erzeugt die LipSync-Komponente die jeweilige Audioausgabe inklusive zugeordneter Mundbewegungen und Mimik.

9.5.3 Sprachsteuerung

Um eine Sprachsteuerung zu implementieren ist eine Schnittstelle zu entsprechenden Speech-to-Text-Services notwendig. Solange ein bestimmtes monatliches Volumen an Anfragen nicht überschritten wird, stellt IBM einen solchen Service über die IBM Cloud kostenlos zur Verfügung. Für die Arbeit mit Unity liefert IBM darüber hinaus ein eigenes Software Development Kit aus, in dem ein entsprechendes Skript („Example Streaming“)

für den Zugriff auf den Speech-to-Text-Service bereits vorhanden ist. Das Skript überwacht die Spracheingabe über ein ausgewähltes Mikrofon und erzeugt zur Laufzeit eine Stringvariable des Gesagten. Da der Nutzer wie im Konzept beschrieben nur vorgegebene Sätze nachspricht, muss die erzeugte String-Variable lediglich mit diesen Sätzen abgeglichen werden.

9.5.4 Verknüpfung der Anforderungen

Die einzelnen Anforderungen an virtuelle Menschen müssen sinnhaft aufeinander abgestimmt werden, um eine glaubwürdige soziale Interaktion zu ermöglichen. Auf die Spracheingaben des Nutzers müssen die jeweils korrespondierenden Antworten gegeben werden, die Sprachausgabe des virtuellen Menschen erfordert die passenden Mundbewegungen, und die Gestik des Nutzers muss den natürlichen Bewegungen während eines Gespräches entsprechen. Über ein benutzerdefiniertes Skript werden diese einzelnen Anforderungen aufeinander abgestimmt und zusammengeführt.

Anpassen des Speech-to-Text-Service

```
60  public void Start()
61  {
62          LogSystem.InstallDefaultReactors();
63          Runnable.Run(CreateService());
64  }
```

```
60  IEnumerator Start()
61  {
62      while(true)
63      {
64          LogSystem.InstallDefaultReactors();
65          Runnable.Run(CreateService());
66          yield return new WaitForSeconds(30);
67      }
68
69  }
```

Abb. 34: *links: Funktion zum Aufruf des Speech-to-Text-Services (Original), rechts: überarbeite Funktion zum Aufruf des Speech-to-Text-Services. (Quelle: Eigene Darstellung)*

Der Speech-to-Text-Service ist im Hintergrund dauerhaft aktiv und wandelt die Spracheingaben des Nutzers kontinuierlich in eine String-Variable um. Pausen zwischen zwei Spracheingaben setzen die String-Variable dabei zurück. Wenn das Skript, das mit dem Service kommuniziert für mehr als 60 Sekunden keine Eingabe registriert, dann unterbricht es den Kontakt zum Service. Unglücklicherweise wird dieser Kontakt nur einmalig beim Start der Anwendung hergestellt. Wenn der virtuelle Mensch beispielsweise einen Monolog hält, der länger als 60 Sekunden andauert, dann geht die Verbindung zum Service verloren und der Nutzer kann anschließend nicht mehr mit dem virtuellen Menschen kommunizieren. Im Falle eines Verbindungsabbruchs fehlt also ein Mechanismus, der die Verbindung wiederherstellt. In *Abb. 34* ist links der standardmäßige Aufruf des Services beim Anwendungsstart dargestellt. Dieser Aufruf muss so angepasst werden, dass er nicht einmalig, sondern in regelmäßigen Zeitabständen eine neue Verbindung zum Service herstellt. Der rechte Teil der Abbildung zeigt diese Anpassung. Durch die Umwandlung des Funktionstyps der Funktion „Start“ in eine sogenannte Coroutine, ist es möglich, Zeitschritte in den Code zu implementieren, die die Ausführung der Funktion für einen bestimmten Zeitraum unterbrechen. In der überarbeiteten Funktion folgt auf den Aufruf des Services nun eine Wartezeit von 30 Sekunden („WaitForSeconds(30)“). Die Bedingung der umgebenden while-Schleife ist immer erfüllt, weshalb sich diese Schleife kontinuierlich wiederholt. Mit dieser einfachen Anpassungen der Start-Funktion wird sichergestellt, dass das Skript alle 30 Sekunden eine neue Verbindung zum Speech-to-Text-Service aufbaut.

Abgleich der Spracheingabe

```
168     public void OnTriggerStay(Collider other)
169     {
170         if(StartRoutineRunning == false)
171         {
172             if (other.tag == "Player")
173             {
174                 Debug.Log(other.gameObject.name);
175                 string string1 = VoiceRecord.text.Substring(0, 8).ToLowerInvariant();
176 
177                 foreach (TMP_Text question in questions)
178                 {
179                     string string2 = question.text.Substring(0, 8).ToLowerInvariant();
180 
181 
182 
183                     if (string1 == string2)
184                     {
185                         VoiceRecord.text = "dummyrecording";
186                         if (CroutineRunning == false)
187                         {
188                             StartCoroutine(EricAudioTogglePlay(counter));
189                         }
190                     }
191                     counter++;
192                 }
193                 counter = 0;
194             }
195         }
```

Abb. 35: *Funktion zum Abgleich der Spracheingabe*

Der virtuelle Mensch ist mit einer quaderförmigen Kollisionsgeometrie ausgestattet, die einen bestimmten Bereich um ihn herum aufspannt. Die in *Abb. 35* gezeigte Funktion („OnTriggerStay“) prüft zur Laufzeit, ob sich die Kollisionsgeometrie des Nutzers innerhalb der Kollisionsgeometrie des virtuellen Menschen befindet. Hinter den virtuellen Menschen sind darüber hinaus graphische Nutzeroberflächen platziert, die die möglichen Fragen an den virtuellen Menschen visuell darstellen. Befindet sich der Nutzer innerhalb der besagten Kollisionsgeometrie und spricht dabei eine der dargebotenen Fragen laut aus, so findet ein Abgleich der String-Variable des Speech-to-Text-Services („VoiceRecord“) mit den in einem Array („questions“) hinterlegten Fragen an den virtuellen Menschen statt. Um den Abgleich zu optimieren, werden die Strings, die dabei miteinander verglichen werden, auf die ersten acht Zeichen reduziert („Substring(0, 8)“), während die Groß- und Kleinschreibung unberücksichtigt bleibt („ToLowerInvariant()“). Wenn der Service nun beispielsweise bei der Spracherkennung einer längeren Frage das letzte Wort falsch erkennt, bleibt der Abgleich der ersten acht Zeichen weiterhin positiv. Der Einfluss etwaiger Fehler bei der Spracherkennung wird dadurch reduziert. Der Array, in dem die Fragen hinterlegt sind, wird nach jeder Spracheingabe vollständig durchlaufen. Dabei zählt eine Variable („counter“) mit, die die Position der aktuellen Frage im Array repräsentiert. Im Falle eines positiven Abgleichs, wird eine Funktion aufgerufen, der diese Variable übergeben wird. Die aufgerufene Funktion kann die übergebene Variable interpretieren und spielt die jeweils zur Frage korrespondierende Audiodatei inklusive der Animationen von Mimik, Mundbewegungen und Gestik ab.

Aufruf von Sprachausgabe und Animationen

```
54      IEnumerator JohnAudioTogglePlay(int answer, TMP_Text TextVar)
55      {
56          CroutineRunning = true;
57
58
59          if (ClipConversation[answer] == true)
60          {
61              yield return new WaitForSeconds(2);
62              Animator.SetBool("GetIdle", false);
63              Animator.SetBool("KeepTalking", true);
64
65              if (answer == 3)
66              {
67                  LipSyncSource.Play(ClipConversation[answer+3]);
68                  TextVar.color = new Color32(120, 209, 255, 255);
69                  yield return new WaitForSeconds(aud_source1.clip.length);
70              }
71
72
73              ...
74              ...
75              ...
76              ...
77
78
79              Animator.SetBool("GetIdle", true);
80              Animator.SetBool("KeepTalking", false);
81          }
82          CroutineRunning = false;
83      }
```

Abb. 36: *Funktion zum Aufruf von Sprachausgabe und Animationen. (Quelle: Eigene Darstellung)*

Der Aufruf der in *Abb. 36* gezeigten Funktion ist an einen positiven Abgleich der Spracheingabe mit einer der möglichen Fragen gekoppelt. Wenn diese Funktion aufgerufen wird, hat der Nutzer also eine zulässige Frage gestellt: Er erwartet nun eine Antwort durch den virtuellen Menschen. Die aufgerufene Funktion steuert daher zunächst die Bool'schen Variablen des jeweiligen Animator-Controller-Objekts an und invertiert deren Werte („Animator.SetBool()"). Die Animator-Komponente des virtuellen Menschen geht damit automatisch in die Animationsschleife über, die für die Sprachausgabe vorgesehen ist. Anhand der übergebenen Ganzzahl, ermittelt die Funktion über verschiedene if-Abfragen die zur Frage korrespondierende Antwort und ruft die entsprechende LipSync-Datei („LipSyncSource.Play") auf. Die LipSync-Komponente des virtuellen Menschen verarbeitet diesen Dateiaufruf und gibt die jeweilige Audiodatei inklusive der Animationen für Mimik und Mundbewegungen wieder. Die gestellte Frage des Nutzers wird außerdem auf der Nutzeroberfläche im Hintergrund des virtuellen Menschen farblich gekennzeichnet, damit der Nutzer sieht, auf welche Frage er gerade eine Antwort erhält („TextVar.color ="). Nach Ablauf der Sprachausgabe schaltet die Funktion die Bool'schen Variablen des Animator-Controller-Objekts wieder auf ihren Ausgangswert zurück – Der virtuelle Mensch geht zu seiner ursprünglichen Animation über.

9.6 Komplexe Beispiele

Nachfolgend werden ausgewählte Beispiele vorgestellt, bei denen zusätzliche Anpassungen über die Implementierung der bereits vorgestellten Mechanismen hinaus erforderlich waren.

9.6.1 Beispiel 1: Einkaufskorb

In der ersten Szene des Spiels benötigt der Nutzer einen Einkaufskorb, um die Waren verstauen zu können. Nachdem die Controller mit Interactor-Prefabs ausgestattet und der Einkaufskorb mit einem Interactable-Prefab verknüpft wurde, galt es weitere Probleme zu lösen, um das Objekt wie gewünscht zu implementieren:

Gelenkverbindung zwischen Korb und Griff

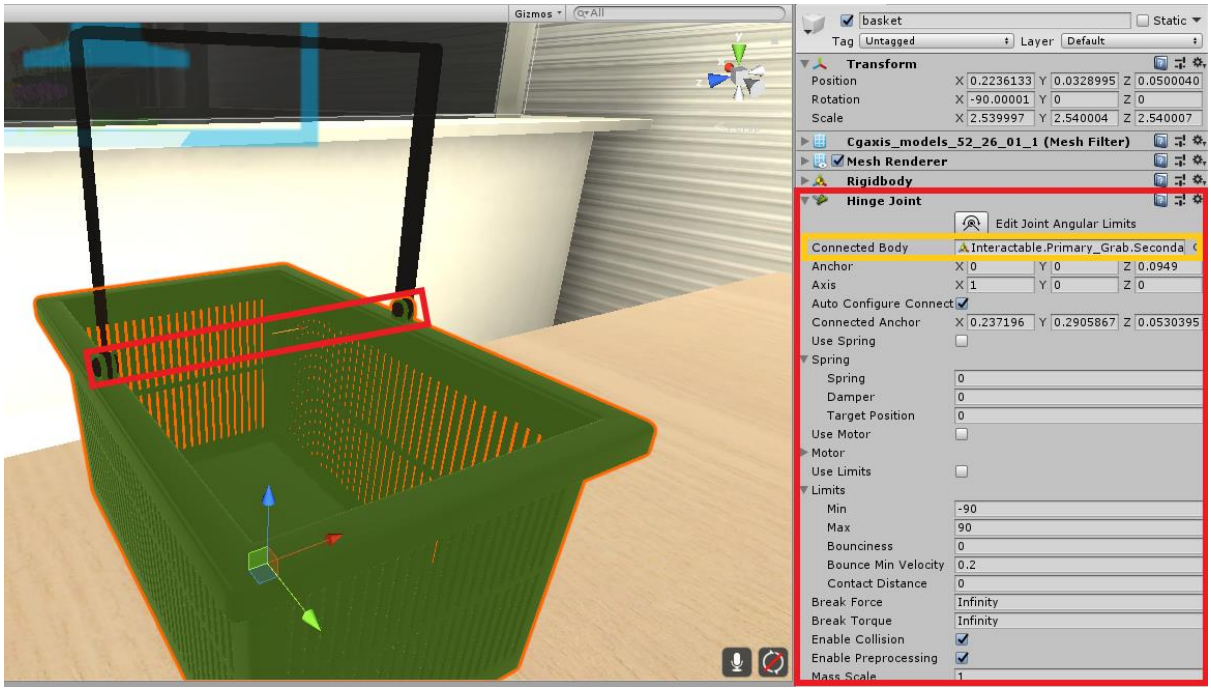

Abb. 37: *links: Darstellung der Position der Gelenkverbindung, rechts: Aufbau der „Hinge Joint"-Komponente. (Quelle: Eigene Darstellung)*

Das Prefab des Einkaufskorbes besteht aus den beiden Objekten „Korb" und „Griff". Die Position beider Objekte ist relativ zum übergeordneten Prefab des Einkaufskorbes. Beide Objekte besitzen also eine feste Raumkoordinate im Bezugssystem des Einkaufskorb-Prefabs. Auf der Ebene der Spielumgebung sind die Bestandteile also statisch miteinander verbunden – Sie werden stets mit dem Prefab verschoben. Unity stellt jedoch Gelenkverbindungen, sogenannte „Joints" bereit, die es ermöglichen, zwei Objekte unter dem Einfluss der Physik-Engine dynamisch miteinander zu verbinden. Für die konkrete Verbindung der beiden Bestandteile „Korb" und „Griff" wird ein Scharniergelenk benötigt, das Unity in Form eines sogenannten „Hinge Joints" als Objektkomponente bereitstellt. Der Korb wird mit dieser Komponente ausgestattet und der Griff mit der Hinge-Joint-Komponente verknüpft. Durch diese Verknüpfung stehen beide Objekte nun über ein Scharniergelenk unter dem Einfluss der Physik-Engine. Die Position der Hinge-Joint-Komponente muss nun nur noch so angepasst werden, dass sie am visuellen Gelenkpunk beider Objekte sitzt (vgl. *Abb. 37*).

Konkave Kollisionsgeometrien für den Einkaufskorb

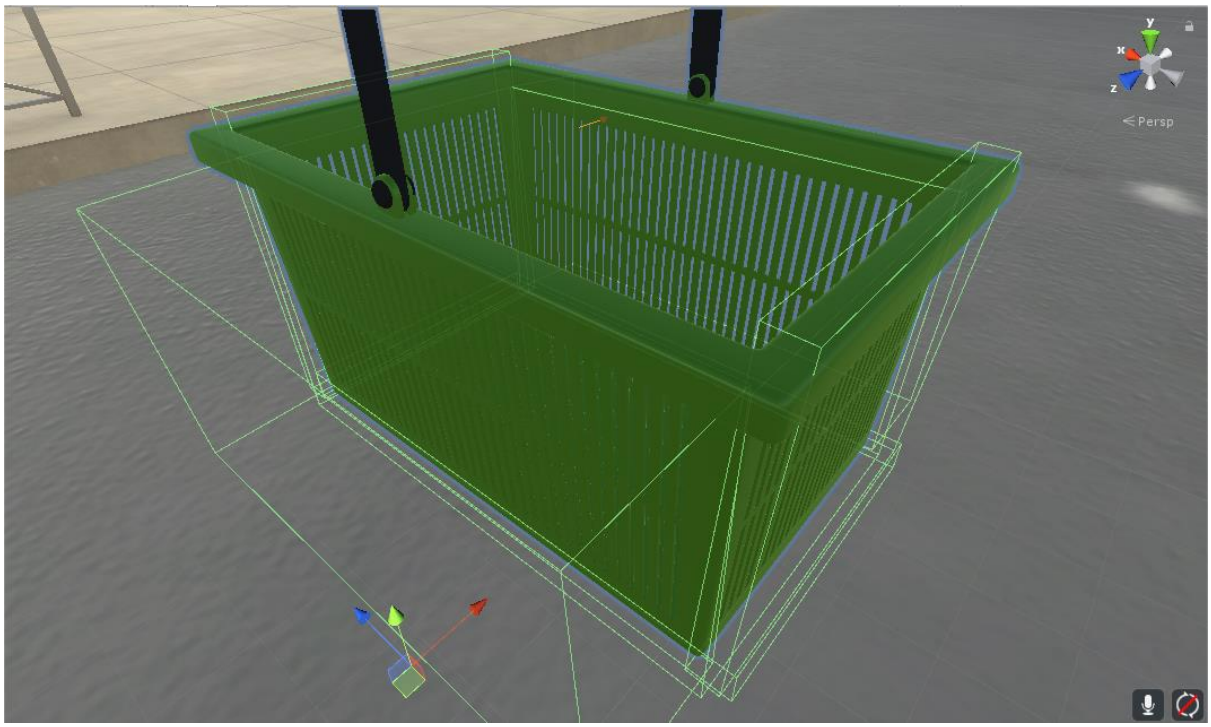

Abb. 38: *Darstellung der aus konvexen Kollisionsgeometrien modellierten konkaven Gesamtkollisionsgeometrie. (Quelle: Eigene Darstellung)*

Objekte können in Unity nur mit konvexen Kollisionsgeometrien ausgestattet werden. Hierfür stellt Unity verschiedene geometrische Figuren („Collider“) wie Kugeln, Quader oder Zylinder bereit. Konkave Kollisionsgeometrien, also solche, die wie ein Korb oder eine Schaufel Tiefe besitzen, müssen mithilfe von mehreren konvexen Collidern modelliert werden. Damit Waren im Einkaufskorb verstaut werden können, benötigt dieser eine solche konkave Kollisionsgeometrie. *Abb. 38* zeigt die aus mehreren konvexen quaderförmigen Collidern („Box Collider“) modellierte konkave Kollisionsgeometrie. Mit diesem Aufbau können nun Objekte, die ihrerseits selbst mit Collidern ausgestattet sind, im Innern des Korbes platziert werden.

Im Korb platzierte Objekte bei der Teleportation berücksichtigen

```
29      void OnTriggerEnter(Collider col)
30      {
31          if (col.GetComponentInParent<Rigidbody>().transform.tag == "Interactable")
32          {
33              col.GetComponentInParent<Rigidbody>().transform.parent = transform.parent;
34          }
35
36      }
```

Abb. 39: *Funktion für den korrekten Teleport der im Korb platzierten Objekte. (Quelle: Eigene Darstellung)*

Die Interactable-Prefabs des Virtual-Reality-Toolkit sind so entwickelt, dass sie kompatibel zur Lokomotionstechnik „Zeigen und Teleportieren“ sind. Teleportiert sich der Nutzer, während er ein Interactable in der Hand hält, so wird dieses Objekt mit dem Nutzer an den jeweiligen Zielpunkt teleportiert. Für die Waren, die sich im Einkaufskorb

befinden und nicht aktiv vom Nutzer festgehalten werden, gilt dies jedoch nicht. Der Einkaufskorb muss also um ein Skript erweitert werden, das es ermöglicht, Waren, die darin enthalten sind, bei der Teleportation zu berücksichtigen (vgl. *Abb. 39*). In Unity ist die Position eines Objektes im Raum abhängig vom jeweiligen Elternelement in der Objekthierarchie. Wird das Elternelement verschoben, so verschieben sich gleichzeitig alle Kindelemente relativ zum Elternelement. Die Objekte, die in den Einkaufskorb gelegt werden, sind standardmäßig Kindelemente zu den jeweiligen Regalen im Supermarkt. Da all diese Objekte auch Interactables sind, sind sie mit einem Tag ausgezeichnet, das die Objekte als Interactables definiert. Über ein Skript lassen sich die Interactables, die im Einkaufskorb landen, anhand ihres Tags eindeutig identifizieren, um sie anschließend als Kindelemente des Einkaufskorbs definieren. Hier kommt erneut die Funktion „OnTriggerEnter“ zum Einsatz. Sie prüft zur Laufzeit in Kombination mit einer If-Abfrage, ob die Kollisionsgeometrie eines Interactables den Einkaufskorb durchdringt. Sobald eine Ware im Korb landet ist dieser Fall erfüllt. Dann steuert das Skript das jeweilige Objekt an und platziert es in der Objekthierarchie als Kindelement zum Einkaufskorb. Von nun an, hat die entsprechende Ware eine relative Position zum Korb. Wenn sich der Nutzer nun mit dem Korb in der Hand teleportiert, bleibt diese relative Verschiebung zum Korb erhalten – die Waren im Korb werden mitteleportiert.

Szenenübergang auslösen

```
29  void OnTriggerEnter(Collider col)
30  {
31
32      if(col.CompareTag("Steak"))
33      {
34          StartCoroutine(FadeOut());
35      }
```

```
22  IEnumerator FadeOut()
23  {
24      yield return new WaitForSeconds(1);
25      overlay.FadeIn();
26      SceneManager.LoadSceneAsync(loadLevel);
27  }
```

Abb. 40: *links: Prüfung auf und Aufruf des Szeneübergangs, rechts: Auslösen des Szenenübergangs (Quelle: Eigene Darstellung)*

Aus dem Konzept geht ein Übergang in die Szene „Regenwald“ hervor, sobald der Nutzer Fleischwaren im Einkaufskorb platziert. Um diesen Übergang auszulösen, muss das zum Korb gehörende Skript zunächst erkennen, dass Fleischwaren im Korb platziert wurden und anschließend eine Funktion aufrufen, die die nächste Szene lädt (vgl. *Abb. 40*). Auch hier prüft eine If-Abfrage, ob der jeweilige Collider, der den Einkaufskorb durchdringt, ein Tag besitzt, das das zugehörige Objekt als Fleischware identifiziert. Ist dies der Fall, wird eine weitere Funktion aufgerufen („FadeOut()“), die die Umgebung des Nutzers fließend ausblendet und dabei die nächste Szene aufruft (vgl. *Abb. 40*).

Offene Probleme – Mögliche Lösungen

Die Physik-Engine von Unity arbeitet mit einer bestimmten Wiederholfrequenz. Sie berechnet die Position und Rotation von Objekten vom einen zum nächsten Zeitschritt und berücksichtigt dabei etwaige Kollisionen. Bewegen sich Objekte jedoch zu schnell wie beispielsweise die Kugel einer Pistole, so ist es möglich, dass die Physik-Engine eine Kollision nicht wahrnimmt und die Objekte durch den entsprechenden Collider tunneln. Dieses Problem tritt auch bei den Waren auf, die sich im Einkaufskorb befinden. Wird der Korb vom Nutzer zu schnell bewegt, tunneln die Objekte durch die Wände des Einkaufskorbes. Da die Objekte gleichzeitig aber Kindelemente des Korbes sind, bleibt ihre Position nach dem Tunneln relativ zum Korb erhalten. Sie befinden sich dann in der

Luft schwebend außerhalb des Korbes und werden weiterhin relativ zum Korb mitbewegt. Weil der Nutzer damit nicht rechnet, beeinträchtigt dieses Phänomen dessen Präsenzgefühl. Um es zu umgehen, kann die Kollisionsphysik für die Collider in Unity angepasst werden. Weiß der Entwickler beispielsweise, dass eine Pistolenkugel auf einen Collider trifft, so kann er die Kollisionserkennung dieses Colliders, sowie die der Kugel entsprechend anpassen. Die Anpassungen lasten die Physik-Engine jedoch wesentlich stärker aus, weshalb derartige Anpassungen für eine große Zahl an Objekten ungeeignet sind. Die Folge wären drastische Auswirkungen auf die Performanz der Anwendung, die sich vor allem durch zeitweise sehr niedrige Framerates äußern würden. Da der Supermarkt aus mehreren hundert Waren besteht, die durch den Nutzer gegriffen und im Einkaufskorb platziert werden können, eignet sich eine Anpassung der Kollisionserkennung zur Lösung des Problems also nicht. Als Alternative bietet es sich an, die Rigidbody-Komponente der Waren auf „Kinematic“ zu schalten. Sobald eine Rigidbody-Komponente mit diesem Parameter ausgezeichnet ist, wird sie nicht mehr durch die Physik-Engine beeinflusst. Wenn Objekte in den Korb fallen gelassen werden, sollte ein kurzer Zeitraum abgewartet werden, bis das Objekt im Korb zum Liegen gekommen ist. Anschließend kann die Rigidbody-Komponente auf „Kinematic“ umgeschaltet werden. Ab diesem Zeitpunkt bleibt das Objekt durch die Physik –Engine unbeeinflusst, es kann also nicht mehr durch die Collider der Korbwände tunneln. Die nun starren Objekte nehmen der Interaktion mit dem Korb nun jedoch geringfügig die Natürlichkeit, da sich die Objekte fortan nichtmehr im Korb bewegen.

9.6.2 Beispiel 2: Kettensäge und Rodung

Das Kettensägen-Prefab wurde über den Asset-Store bezogen und kommt standardmäßig mit einem Skript, das die Animationen und Töne der Kettensäge steuert. Wenn die Kettensäge in einen Collider eintaucht, dann schaltet das Skript automatisch den Ton der Kettensäge auf das typische Betriebsgeräusch um, ansonsten läuft die Kettensäge im Leerlauf. Damit die Kettensäge vom Nutzer gegriffen werden kann, wurde sie zudem mit einem Interactable verknüpft. Um den konzeptuellen Anforderungen gerecht zu werden, muss die Säge aber weiterhin dazu in der Lage sein, den entsprechenden Baum in der Spielumgebung fällen zu können. Über den Default hinaus soll außerdem ein Starter implementiert werden, um der Interaktion mit der Kettensäge mehr Natürlichkeit zu verleihen.

Starter implementieren

```
13      void Start()
14      {
15
16
17          LineRenderer lineRenderer = GetComponent<LineRenderer>();
18
19          lineRenderer.SetPosition(0, Handle.transform.position);
20          lineRenderer.SetPosition(1, Chainsaw.transform.position);
21      }
22
23      void Update()
24      {
25          LineRenderer lineRenderer = GetComponent<LineRenderer>();
26
27          lineRenderer.SetPosition(0, Handle.transform.position);
28          lineRenderer.SetPosition(1, Chainsaw.transform.position);
29      }
```

Abb. 41: *Funktionen zur Erzeugung des Starterkabels (Quelle: Eigene Darstellung)*

Der Starter einer Kettensäge setzt sich aus einem Griff und einem Kabel zusammen, das ins Innere der Kettensäge verläuft. Für den Griff eignet sich eine einfache geometrische Figur wie ein Zylinder. Dieser wird in seiner Größe angepasst, mit einem Interactable-Prefab verknüpft und als Kindelement der Kettensäge an dieser angebracht.
Das Kabel, das den Griff und die Kettensäge visuell miteinander verbindet, wird über eine sogenannte Line-Renderer-Komponente umgesetzt. Diese Komponente erzeugt eine sichtbare Linie zwischen zwei fest definierten Punkten. Mithilfe eines Skripts lassen sich diese Punkte auch dynamisch definieren, indem die Punkte den Koordinaten bestimmter Objekte folgen. Über die Start-Funktion im Skript werden diese Koordinaten bei Anwendungsstart definiert, die Update-Funktion gleicht die Position der Endpunkte des Line-Renderers mit jedem neuen Frame an die Position der verfolgten Objekte (Kettensäge & Griff) an (vgl. *Abb. 41*).

```
21      void Update()
22      {
23          if(OVRInput.GetLocalControllerVelocity(OVRInput.Controller.LTouch).magnitude <= velocity1 && checkgrip == true &&
24              OVRInput.GetLocalControllerVelocity(OVRInput.Controller.LTouch).magnitude > 1)
25          {
26              if(!aud_source.isPlaying)
27              {
28
29                  aud_source.Play();
30              }
31
32          }
33
34          if (OVRInput.GetLocalControllerVelocity(OVRInput.Controller.LTouch).magnitude > velocity1 && checkgrip == true)
35          {
36              if (!aud_source.isPlaying)
37              {
38
39                  aud_source.Play();
40                  ChainSaw_Controller.Enable();
41              }
42
43          }
44      }
```

Abb. 42: *Funktion zum Starten der Kettensäge (Quelle: Eigene Darstellung)*

Der Nutzer kann den Startergriff nun greifen und von der Kettensäge wegziehen. Dabei kommt das Starterkabel zum Vorschein, das denn Griff und die Kettensäge miteinander verbindet. Nun wird ein weiteres Skript implementiert, das über die Update-Funktion die Geschwindigkeit des Controllers, der den Startergriff festhält, überwacht. Sobald der Nutzer den Griff in die Hand nimmt, schaltet eine Bool'sche Variable („checkgrip"), die mit dem Interactable verknüpft ist, auf den Wert „true". Wenn die Geschwindigkeit des Controllers nun einen definierten Wert überschreitet, wird ein kurzes Startgeräusch widergegeben und anschließend das Skript aufgerufen („ChainSaw_Controller"), das die Animationen und Geräusche der Kettensäge steuert. Dieses Skript startet die Kettensäge und versetzt sie in den Leerlauf. Bei positivem Greifstatus aber Unterschreiten der Grenzgeschwindigkeit wird ebenfalls ein kurzes Startgeräusch abgespielt, das entsprechende Skript aber nicht aufgerufen – die Kettensäge startet nicht.
Sobald der Nutzer die Kettensäge gestartet hat, lässt er den Startergriff intuitiv los. Dabei erwartet der Nutzer, dass das Kabel sich im Innern der Maschine wieder aufwickelt und der Griff wieder in seine Ausgangsposition zurückkehrt. Um dies zu ermöglichen muss erneut eine Gelenkverbindung („Joint") implementiert werden. In diesem Fall handelt es sich um einen sogenannten „Spring Joint", der sich wie eine Feder verhält und den Griff unter dem Einfluss der Physik-Engine mit der Kettensäge verbindet. Greift der Nutzer den Griff, so wirkt er eine Kraft auf die Feder aus, sobald er den Griff wieder loslässt wandelt die Feder die gespeicherte Energie wieder in Bewegungsenergie um und kehrt in ihre Ausgangsposition zurück. Mit den beschriebenen Vorkehrungen ist der Starter nun funktionstüchtig.

Baum fällen

Abb. 43: *Aufbau des Baummodells für die Rodung bestehend aus Stumpf und Stamm. (Quelle: Eigene Darstellung)*

Die Kettensäge ist nun voll funktionstüchtig. Sie wird über den Starter gestartet, von da an steuert ein standardmäßig mitgeliefertes Skript deren Töne und Animationen. Für das Fällen des Baumes werden neben der Kettensäge zwei Modelle benötigt: ein Baumstumpf sowie ein gesamter Baum, der über dem Baumstumpf platziert und mit einer Rigidbody-Komponente ausgestattet wird (vgl. *Abb. 43*). Der Baumstumpf und der Baum sind voneinander unabhängige Objekte.

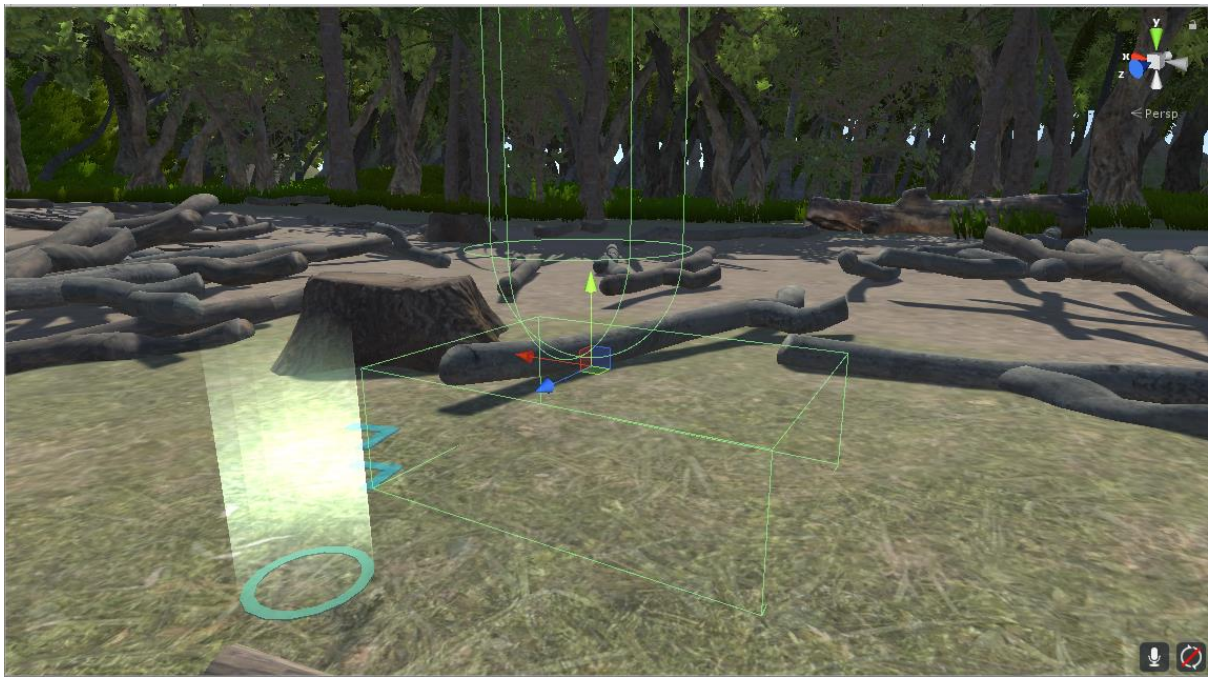

Abb. 44: *Aufbau der Gesamtkollisionsgeometrie des Baummodells für die Rodung. (Quelle: Eigene Darstellung)*

Eine quaderförmige Kollisionsgeometrie („Box Collider“) und eine kapselförmige Kollisionsgeometrie („Capsule Collider“) bilden als Kindelemente des Baums dessen Gesamtkollisionsgeometrie (vgl. *Abb. 44*). Da ein Capsule Collider halbkugelförmige Enden besitzt, wird ein Box Collider als Basis der Gesamtkollisionsgeometrie benötigt. Ohne den Box Collider würde der Baum beim Start der Szene automatisch umkippen. Ein dritter Collider, ebenso Kindelement des Baumes, wird zuletzt für das Skript implementiert, das den Vorgang des Fällens steuern soll (vgl. *Abb. 45*).

```
11      void OnTriggerEnter(Collider col)
12      {
13          if(col.gameObject.tag == "Chainsaw")
14          {
15              StartCoroutine(WaitChop());
16          }
17
18      }
19
20      IEnumerator WaitChop()
21      {
22          yield return new WaitForSeconds(4);
23          Stump_Collider.parent = Abraum;
24
25      }
```

Abb. 45: *Funktion zum Auslösen des Baumfällens. (Quelle: Eigene Darstellung)*

Dieser Collider wird durch die Funktion „OnTriggerEnter“ zur Laufzeit überwacht. Sobald der Collider der Kettensäge diesen Collider durchdringt, wird eine neue Funktion („WaitChop()“) aufgerufen, die den Baum schließlich zum Fallen bringt. Damit der Baum nicht unmittelbar fällt, wenn der Nutzer ihn mit der Kettensäge durchdringt, verzögert die aufgerufene Funktion das Fällen des Baumes zunächst um vier Sekunden. Nach Ablauf der vier Sekunden wird der Box Collider, ursprünglich Kindelement des Baumes, nun als Kindelement des vom Baum unabhängigen Stumpfs deklariert. Fortan gibt es also keine kombinierte Kollisionsgeometrie mehr für den Baum, lediglich der Capsule Collider bleibt als Kindelement des Baumes bestehen. Da der Capsule Collider allein kein stabiles Gleichgewicht besitzt, beginnt der Baum über dem Stumpf nun zu fallen. Es bleiben ein Baumstumpf und ein gefällter Baum zurück.

10 Fazit

Ziel dieser Arbeit war die Konzeption und Umsetzung einer Virtual-Reality-Anwendung als interaktives Aufklärungsspiel zum Thema „Auswirkungen des Fleischkonsums auf den Klimawandel“. Unter dem Leitmotiv „Präsenz“ sollte eine Anwendung entwickelt werden, die den Nutzer sowohl informativ, vor allem aber auch emotional anspricht. Der Nutzer sollte durch das Erleben der Anwendung nachhaltig in seinem Denken und im besten Fall auch in seinem Handeln beeinflusst werden. Blickt man auf das Konzept, so ist ein Narrativ entstanden, das dieser Zielsetzung durchaus gerecht werden kann. Auch die verschiedenen Konzepte zu Lokomotion und Interaktion fügen sich schlüssig in die Handlung der Anwendung. Ohne die zeitliche Begrenzung, wie sie im Rahmen einer Bachelorthesis üblich ist, bestünden aber sicherlich noch Möglichkeiten, konzeptionell tiefer ins Detail zu gehen. Das Verhältnis zwischen konzeptuellen Anforderungen und tatsächlich umgesetzten Inhalten zeigt jedoch, dass der Entwicklungsaufwand einer solchen Anwendung ohne jegliche Vorerfahrung deutlich unterschätzt wurde. Unter dem Strich konnte nur knapp die Hälfte des Gesamtspielverlaufs bei der Entwicklung der Anwendung umgesetzt werden. Für die einzelnen umgesetzten Inhalte ist konzeptionell zwar sichergestellt, dass sie das Präsenzgefühl des Nutzers positiv beeinflussen, die Unvollständigkeit der Handlung fällt als präsenzstörender Faktor jedoch stärker ins Gewicht. Bei einer ganzheitlichen persönlichen Betrachtung ist die Anwendung aufgrund ihrer Unvollständigkeit daher leider nicht dazu in der Lage, eine nachhaltige Nutzererfahrung zu schaffen. Rückblickend hätte dieses Ziel im zeitlich gegebenen Rahmen möglicherweise durch ein kürzeres Handlungskonzept oder aber durch eine stärkere Abgrenzung des Themas erreicht werden können. Ob Logik und Informationsgehalt dann aber darunter gelitten hätten, bleibt zu beantworten. Die Inhalte, die bei der Zielsetzung für den Entwicklungsschluss gefordert wurden, konnten im zeitlich gegebenen Rahmen jedoch allesamt zufriedenstellend umgesetzt werden. Die im Konzept erarbeiteten Fortbewegungs- und Interaktionstechniken wurden implementiert, virtuelle Menschen und soziale Interaktionen sind fortan ein zentraler Bestandteil der Anwendung und die vorgestellten Beispiele zeigen, wie auf Basis der interaktiven und fortbewegungsbezogenen Grundbausteine komplexere anwendungsspezifische Inhalte entstehen konnten.

Am Ende der Bearbeitungsphase dieser Thesis steht eine bezüglich des Spielverlaufs unfertige Anwendung. Eine Anwendung, die aber auch etliche wertvolle Bausteine zur Wiederverwendung besitzt. Diese Bausteine können bei einer möglichen Weiterentwicklung als Grundlage für die Implementierung der noch offenen Inhalte des Konzepts genutzt werden. Um noch unentdeckte Fehler aufzuspüren und die Usability sowie die User Experience zu verbessern, bieten sich zudem verschiedene Tests beziehungsweise auch eine offene Testphase an, wie sie bei neuer Software allgemein üblich ist. Die Vision einer Anwendung zur Aufklärung, die Nutzerfahrungen schafft, die den Nutzer über das Erleben der Anwendung hinaus beeinflussen, bleibt durch die Möglichkeit zur Weiterentwicklung der Anwendung „Climeat Change“ bestehen.

11 Bibliographie

ADAMS, Ernest (2004): „*Postmodernism and the Three Types of Immersion*“. <https://www.gamasutra.com/view/feature/130531/the_designers_notebook_.php> [Stand: Juli 2004. Zugriff: 22.07.2019 21:12 MESZ]

BAILENSON, Jeremy (2018): *„How to create empathy in VR“.* <https://www.wired.co.uk/article/empathy-virtual-reality-jeremy-bailenson-stanford> [Stand: Februar 2018. Zugriff: 17.07.2019 17:53 MESZ]

BAUER, Frank et al. (2009): „*Gamma flicker triggers attentional selection without awareness“.* In: FULTON, Kenneth R. (Hrsg.) (2009): Proceedings of the National Academy of Sciences of the United States of America. Vol. 106 No. 3. Washington : Nation Academy of Sciences, 1666-1671

BOWMAN, Doug A. / HODGES, Larry F. (1999): *An Evaluation of Techniques for Grabbing and Manipulating Remote Objects in Immersive Virtual Environments*. Georgia Institute of Technology; College of Computing; Graphics, Visualization, and Usability Center; Scientific Paper, 1999

-, - et al. (2005): *3D User Interfaces : Theory and Practice.* Boston : Pearson

BOZGEYIKLI, Evren (2016): *Locomotion in Virtual Reality for Room Scale Tracked Areas.* University of South Florida, College of Engineering, Department of Computer Science and Engineering, Dissertation, 2016

BROOKS, Frederick P. (1999): *„What's real about Virtual Reality“.* In: IEEE Computer Society Press (Hrsg.) (1999): IEEE Computer Graphics and Applications. Vol. 19 No. 6. Los Alamitos : Computer Society Press, 16-27

BUTTUSSI, Fabio / CHITTARO, Luca (2019): „*Locomotion in Place in Virtual Reality : A Comparative Evaluation of Joystick, Teleport, and Leaning*“. In: IEEE Computer Society Press (Hrsg.) (2019): IEEE Transactions on Visualization and Computer Graphics. Vol. 25. No. ?. Los Alamitos : Computer Society Press

DAVIS, Simon / NESBITT, Keith / NALIVAIKO, Eugene (2015): *Comparing the onset of cybersickness using the Oculus Rift and two virtual roller coasters*. University of Newcastle, School of Design Communication and IT, Conference Paper, 2015

DÖRNER, Ralf et al. (2014): *Virtual und Augmented Reality (AR/VR) : Grundlagen und Methoden der Virtuellen Realität.* Berlin : Springer

GLATZ, Wolfgang (2016): „*HTC Vive: Virtuelle Welten für große Wohnzimmer*“ <https://www.hardwareschotte.de/magazin/vr-brillen-htc-vive-oder-oculus-rift-kurzzusammenfassung-der-tests-a41925> [Stand: Juli 2016. Zugriff: 21.08.2019, 20:55 MESZ]

GRASNICK, Armin (2016): *3D ohne 3D-Brille : Handbuch der Autostereoskopie.* Berlin : Springer

GROß, Julia (2017): *„Immer in Bewegung“.* <https://www.dasgehirn.info/wahrnehmen/sehen/immer-bewegung> [Stand: Januar 2017. Zugriff: 20.07.2019, 13:50 MESZ]

HANSER, Hartwig (Hrsg.) (2005): *Lexikon der Neurowissenschaft*. Heidelberg : Spektrum

HEANLEY, David (2019): *„VRTK v4 Beta Makes The Best Unity VR Framework Even Better“.* <https://uploadvr.com/vrtk-v4-beta-release/> [Stand: April 2019. Zugriff: 30.07.2019, 16:56 MESZ]

HEINICKE, Andreas M. (2012): *Mensch-Computer-Interaktion : Basiswissen für Entwickler und Gestalter.* Berlin : Springer

HERMAN, Irving P. (2016): *Physics of the Human Body*. Vol. 2. Cham : Springer

HERTEL, Yannic (2017): *„Die Geschichte der virtuellen Realität“.* <https://www.vrnerds.de/die-geschichte-der-virtuellen-realitaet/> [Stand: Februar 2017. Zugriff: 17.07.2019, 20:09 MESZ]

HOFFMANN, David M. et al. (2008): *„Vergence-accommodation conflicts hinder visual performance and cause visual fatigue“.* In: ARVO Journals (Hrsg.) (2008): Journal of Vision. Vol. 8 No. 3. Rockville : ARVO

INTERRANTE, Victoria / ANDERSON, Lee / RIES, Brian (2006): *Distance Perception in Immersive Virtual Environments, Revisited.* University of Minnesota, Department of Computer Science & Department of Architecture, Conference Paper, 2006

IRISVR (o.J.): “*The Importance of Frame Rates*”. <https://help.irisvr.com/hc/en-us/articles/215884547-The-Importance-of-Frame-Rates> [Stand: k.A. Zugriff: 26.07.2019, 15:22 MESZ]

JANNSEN, Jan-Keno (2016): „*Oculus Touch im Test: So echt können sich Hände in VR anfühlen*“. <https://www.heise.de/ct/artikel/Oculus-Touch-im-Test-So-echt-koennen-sich-Haende-in-VR-anfuehlen-3552295.html> [Stand: Dezember 2016. Zugriff: 02.08.2019, 17:04 MESZ]

JOHNSON, Jason (o.J.): *„Atari’s secret VR experiments of the 1980s”* <https://killscreen.com/versions/ataris-secret-vr-experiments-of-the-1980s/> [Stand: k.A. Zugriff: 17.07.2019, 22:02 MESZ]

KORGEL, Daniel (2017): *Virtual Reality-Spiele entwickeln mit Unity® : Grundlagen, Beispielprojekte, Tipps & Tricks.* München : Hanser

KUBOVY, Michael (1986): *The Psychology of Perspective and Renaissance Art.* Cambridge : Cambridge

Kuratorium Gutes Sehen e.V (o.J.): „Das Auge – Aufbau und Funktion“ <https://www.sehen.de/sehen/das-auge/> Stand: k.A. Zugriff: 21.08.2019, 20:11 MESZ]

LAUKKONEN, Jeremy (2019): *„What is Oculus Touch? : Motion controls for Oculus Rift“* <https://www.lifewire.com/oculus-touch-4159174> [Stand: Juni 2019. Zugriff: 31.07.2019, 18:50 MESZ]

MAGNENAT-THALMANN, Nadia / THALMANN, Daniel (2004): *Handbook of Virtual Humans.* Chichester : Wiley

MEEHAN, Michael et al. (2003): *Effect of Latency on Presence in Stressful Virtual Environments.* Stanford University, Conference Paper, 2003

MON-WILLIAMS, Mark / WANN John P. (1998): *„Binocular Virtual Reality Displays: When Problems Do and Don't Occur“.* In: Human Factors (Hrsg.) (1998): Human Factors: The Journal of the Humand Factors and Ergonomics Society. Vol. 40 No. 1. Thousand Oaks : SAGE, 42-49

MORINA, Nexhmedin et al. (2014): *„Sense of presence and anxiety during virtual social interactions between a human and virtual humans“.* In: PeerJ (Hrsg.) (2014): PeerJ. Vol. 2. San Diego : PeerJ

MÜLLER-LINDENLAUF, Maria (2012): *„CO²-Fußabdruck und Umweltbilanz regionaler Lebensmittel“.* <https://www.ifeu.de/landwirtschaft/pdf/IFEU_Umwelt_Regionale_Lebensmittel_2012_final_handout.pdf> [Stand: April 2012. Zugriff: 17.07.2019, 14:31 MESZ]

NEEB, Annette (2017): *„Ein Blick zurück: Die Geschichte der Virtual Reality“.* <https://www.bigfishgames.de/blog/ein-blick-zuruck-die-geschichte-der-virtual-reality/> [Stand: April 2017. Zugriff: 17.07.2019, 20:34 MESZ]

NIELD, David (2016): „*How Oculus Rift works: Everything you need to know about the VR sensation : From your PC to your eyeballs and the bits in between“.* <https://www.wareable.com/vr/how-oculus-rift-works> [Stand: März 2016. Zugriff: 31.07.2019, 16:35 MESZ]

OCULUS (2016): „*Oculus Touch – Hand Presence Technology*“ <https://www.youtube.com/watch?v=pppkQ4jIrMU> [Stand: Oktober 2016. Zugriff: 21.08.2019, 21:07 MESZ]

PEIX, Franziska (2019): „*Oculus Touch: VR-Controller mit gruseligen Botschaften bedruckt*“ <https://www.turn-on.de/play/news/oculus-touch-vr-controller-mit-gruseligen-botschaften-bedruckt-480345> [Stand: April 2019. Zugriff: 21.08.2019, 21:02 MESZ]

REEVES, William T. (1983): *„Particle Systems – A Technique for Modeling a Class of Fuzzy Objects“.* In: ACM (1983): ACM Transactions on Graphics. Vol 17. No. 3. New York : ACM

SLATER, Mel / USOH, Martin / STEED, Anthony (1994): *„Depth of Presence in Virtual Environments“.* In: MIT Press (Hrsg.) (1994): Presence: Teleoperators and Virtual Environments. Vol. 3 No. 2. Cambridge : MIT Press, 130-144

SPEKTRUM HÖREN (o.J): „Gehör und Hörbeeinträchtigungen – Der Aufbau des Ohres“ <https://www.spektrum-hoeren.de/gehoer-und-hoerbeeintraechtigung/9-aufbau-des-ohres/79-der-aufbau-des-ohres> [Stand: k.A. Zugriff: 21.08.2019, 20:17 MESZ]

STEINICKE, Frank et al. (2010): „*Estimation of Detection Thresholds for Redirected Walking Techniques“.* .In: IEEE (2010): IEEE Transactions on Visualization and Computer Graphics. Vol. 16 No. 1. Los Alamitos : Computer Society Press, 17-27

STEINICKE, Frank (2016): *Being Really Virtual : Immersive Natives and the Future of Virtual Reality.* Cham : Springer

STEUER, Jonathan (1992): “*Defining Virtual Reality : Dimensions Determining Telepresence”.* In: Levy, Mark R. (Hrsg.) (1992): Journal of Communication. Vol. 42 No. 4. Stanford : Stanford, 73-93

SORENE, Paul (2014): „*Jaron Lanier’s EyePhone: Head and Glove Virtual Reality in the 1980s*“ <https://flashbak.com/jaron-laniers-eyephone-head-and-glove-virtual-reality-in-the-1980s-26180/> [Stand: November 2014. Zugriff: 21.08.2019, 20:10 MESZ]

ULIN, Don (2010): *„After Trees Are Cut Down“.* <https://indianapublicmedia.org/amomentofscience/rain-forest-soil-poor-trees-cut/> [Stand: November 2010. Zugriff: 17.07.2019, 14:12 MESZ]

UMWELTBUNDESAMT (2019): *„Beitrag der Landwirtschaft zu den Treibhausgasemissionen“.* <https://www.umweltbundesamt.de/daten/land-forstwirtschaft/beitrag-der-landwirtschaft-zu-den-treibhausgas#textpart-1> [Stand: April 2019. Zugriff: 17.07.2019, 14:38 MESZ]

UNITY (2018): „*VR overview*“. <https://docs.unity3d.com/2018.4/Documentation/Manual/VROverview.html> [Stand: 2018. Zugriff: 30.07.2019, 16:27 MESZ]

UNREAL ENGINE (2019a): „*Unreal Engine For AR, VR & MR“.* <https://www.unrealengine.com/en-US/vr> [Stand: 2019. Zugriff: 31.07.2019, 14:52 MESZ]

UNREAL ENGINE (2019b): „*What is Unreal Engine 4“.* < https://www.unrealengine.com/en-US/what-is-unreal-engine-4> [Stand: 2019. Zugriff: 31.07.2019, 14:55 MESZ]

VISCIRCLE (2018): „*Unreal Engine vs. Unity: welche Software sollten Gamedeveloper wählen?“.* <https://viscircle.de/unreal-engine-vs-unity-welche-software-sollten-gamedeveloper-waehlen/> [Stand: Oktober 2018. Zugriff: 31.07.2019, 15:01>

VISHWANATH, Dhanraj / GIRSHICK, Ahna R. / BANKS, Martin S. (2005): *„Why pictures look right when viewed from the wrong place“*. In: nature neuroscience (Hrsg.) (2005): nature neuroscience. Vol. 8 No. 10. London : Nature, 1401-1410

VÖTTER, Raffael (2012): *„Der 24-Fps-Mythos: Warum 24 Frames in Spielen nicht flüssig ist – Bild- und Videobeweise – Update: Slow-Motion-Video“.* <https://www.pcgameshardware.de/Spiele-Thema-239104/Tipps/Wann-laufen-Spiele-fluessig-1034704/> [Stand: November 2012. Zugriff: 26.07.2019, 16:00 MESZ]

VRBRILLEN.NET (o.J): „Oculus Rift Test“ <http://www.vrbrillen.net/oculus-rift/> [Stand: k.A. Zugriff: 21.08.2019, 20:46 MESZ]

VRNERDS (o.J.a): *„VR Brillen Vergleich*“. <https://www.vrnerds.de/vr-brillen-vergleich/> [Stand: k.A. Zugriff: 31.07.2019, 17:09 MESZ]

VRNERDS (o.J.b): *„VR Glossar*“ <https://www.vrnerds.de/vr-glossar/> [Stand: k.A. Zugriff: 21.08.2019, 20:34 MESZ]

VRPLAYGROUND (o.J.): „HTC Vive“ <https://www.vrplayground.de/htc-vive> [Stand: k.A. Zugriff: 21.08.2019, 21:13 MESZ]

VRTK (o.J.): *„Welcome to VRTK“.* <https://vrtoolkit.readme.io/docs/summary> [Stand: k.A. Zugriff: 30.07.2019, 16:53 MESZ]

VRTK ACADEMY (o.J.): *„Adding a Teleporter“* <https://academy.vrtk.io/Documentation/HowToGuides/Locomotion/AddingATeleporter/> [Stand: k.A. Zugriff: 21.08.2019, 21:30 MESZ]

WANGER, Leonard R. / FERWERDA, James A. / GREENBERG, Donald P. (1992): “*Perceiving spatial relationships in computer-generated images”.* In: IEEE Computer Society Press (Hrsg.) (1992): IEEE Computer Graphics and Applications. Vol. 12 No. 3. Los Alamitos : Computer Society Press, 44-58

WEICHERT, Frank et al. (2013): *„Analysis oft he Accuracy and Robustness of the Leap Motion Controller“.* In: MDPI (2013): Sensors Bd. 13 Nr.5. Basel : MDPI, 6380-6393

WIKIPEDIA (2018): *„Lightmap“.* < https://de.wikipedia.org/wiki/Lightmap> [Stand: April 2018 Zugriff: 31.07.2019, 17:37 MESZ]

-, - (2019a): *„HTC Vive“.* < https://de.wikipedia.org/wiki/HTC_Vive> [Stand: Mai 2019 Zugriff: 03.08.2019, 14:19 MESZ]

-, - (2019b): *„Software Development Kit“.* <https://de.wikipedia.org/wiki/Software_Development_Kit> [Stand: Mai 2019 Zugriff: 30.07.2019, 16:31 MESZ]

-, - (2019c): *„Unity (Spiel-Engine)“.* <https://de.wikipedia.org/wiki/Unity_(Spiel-Engine)> [Stand: Juli 2019 Zugriff: 30.07.2019, 15:18 MESZ]

-, - (2019d): „Sechs Freiheitsgrade“. <https://de.wikipedia.org/wiki/Sechs_Freiheitsgrade> [Stand: Mai 2019 Zugriff: 21.08.2019, 20:30 MESZ]

WOHLERT, Claudia (2018): *„Virtual Reality für eine bessere Welt“*. <https://www.saz.com/de/virtual-reality-fuer-eine-bessere-welt> [Stand: k.A. Zugriff: 17.07.2019 16:09 MESZ]

WWF DEUTSCHLAND (2012): *Klimawandel auf dem Teller.* Berlin : Selbstverlag

-, - (2014a): *Schwere Kost für Mutter Erde.* Berlin : Selbstverlag

-, - (2014b): *Fleisch frisst Land.* Berlin : Selbstverlag

WWF INTERNATIONAL (2007): *Der Teufelskreis am Amazonas.* Gland : Selbstverlag

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Alle Ausführungen, die anderen veröffentlichten oder nicht veröffentlichten Schriften wörtlich oder sinngemäß entnommen wurden, habe ich kenntlich gemacht.

Die Arbeit hat in gleicher oder ähnlicher Fassung noch keiner anderen Prüfungsbehörde vorgelegen.

_____ _____

Ort, Datum Unterschrift