Hochschule Kempten
University of Applied Sciences

BACHELORARBEIT IM STUDIENGANG INFORMATIK - GAME ENGINEERING

NON-VERBALE KOMMUNIKATION MIT AUTONOM GESTEUERTEN SPIELFIGUREN IN AUGMENTED REALITY WELTEN

DENNIS VIDAL

Kurzbeschreibung

Das Thema dieser Arbeit ist das Design und die Implementierung einer Augmented-Reality-Anwendung für die Microsoft HoloLens, in welcher ein computergesteuerter virtueller Charakter auf verschiedene Weisen glaubwürdig nonverbal mit dem Spieler kommunizieren kann. Dazu zählt das Design und die Implementation eines Blicksystems, mit dem der Charakter verschiedene Blickverhalten ausführen kann und ein System, mit dem der Charakter versucht eine bestimmte zwischenmenschliche Distanz zum Spieler einzuhalten.

Aufgabensteller/Prüfer:	Prof. Dr. Christoph Bichlmeier
Arbeit vorgelegt am:	02.06.2020
Durchgeführt an der:	Fakultät für Informatik

VORWORT:

Ich möchte mich an dieser Stelle herzlich bei Prof. Dr. Christoph Bichlmeier für sowohl die Betreuung dieser Bachelorarbeit als auch die Beantwortung verschiedener Fragen während der Durchführung dieser Arbeit bedanken.

INHALTSVERZEICHNIS

1 EINLEITUNG

Die Interaktion und Kommunikation zwischen Spielern und computergesteuerten Charakteren (auch non-player characters oder kurz NPCs genannt) sind in den meisten Videospielen wichtige Bestandteile des Spielerlebnisses und werden genutzt, um die Immersion des Spielers zu erhöhen, die Spielwelt lebendiger wirken zu lassen und vor allem um dem Spieler verschiedene Elemente des Spieles und dessen Geschichte zu vermitteln. So werden zum Beispiel die Aufgaben in einem Spiel oft über Dialoge vermittelt, Hostilitäten gegenüber einem Charakter durch einen zornigen Gesichtsausdruck und entsprechende Körperhaltung oder die Verletzung eines Charakters in der Vergangenheit über ein Humpeln während des Gehens.

Die Einflüsse dieser beiden Themen sind dabei nicht nur auf Videospiele begrenzt, sondern spielen auch in anderen Medien, wie etwa in Filmen und Werbungen, eine große Rolle. Unter anderem werden sie in Filmen, sehr ähnlich zu Videospielen, genutzt, um animierte Charaktere lebendiger und natürlicher beziehungsweise „echter“ wirken zu lassen. In Werbungen dagegen finden die beiden Themen eher Anwendung, um ein Produkt in einem guten Licht darzustellen und einen bestimmten Eindruck bezüglich diesem zu erwecken.

Während der Interaktion zwischen einer echten Person und einem virtuellen Charakter kann die Kommunikation von Informationen, allgemein gesehen, auf zwei Arten stattfinden. Die erste und wahrscheinlich offensichtlichere der beiden ist die Übermittlung der Informationen über den verbalen Weg, also mittels den Bedeutungen von Worten. Dagegen stellt die zweite Art die Übermittlung der Informationen über nonverbale Verhalten und Wege, wie etwa der Körpersprache oder dem Aussehen eines Charakters, dar. Der Unterschied in den jeweiligen Effekten, den die beiden Kommunikationsarten sowohl in der Realität als auch in Videospielen und anderen Medien haben können, ist dabei nicht zwingend sofort ersichtlich. Einer dieser unterschiedlichen Effekte wird von Patterson beschrieben: „Although the verbal content of our interactions is obviously important, the nonverbal side usually has a greater impact on how we feel and think about others and, eventually, how we get along with them“ (M. L. Patterson, 2009, S. 131). Dementsprechend kann die nonverbale Kommunikation auch in Videospielen genutzt werden, um NPCs bestimmte Eigenschaften zu verleihen, die vom Spieler aktiv und passiv wahrgenommen werden und somit einerseits den Eindruck, den ein NPC beim Spieler erweckt, zu beeinflussen, andererseits aber auch die wahrgenommene Präsenz eines NPCs zu erhöhen.

Während die verbale Kommunikation in nahezu allen Videospielen, in der einen oder anderen Form, Anwendung findet, variiert die Verwendung der nonverbalen Kommunikation, zwischen NPCs und Spielern, oft stark nach der Art des Spieles und dem Teilgebiet der nonverbalen Kommunikation. Die meisten Videospiele implementieren zwar Teilgebiete, wie Körpersprache und Gesichtsausdrücke, relativ weitgehend (wenn auch oft Animationen aus Kosten- und Zeitgründen wiederverwendet werden), allerdings ist die Implementation anderer Teilgebiete, wie dem Blickkontakt, der Blickverfolgung und dem Einhalten einer zwischenmenschlichen Distanz zwischen NPCs und Spieler, oft nur sehr limitiert vorhanden.

Die NPCs vieler Videospiele haben zwar eine Distanz, die sie versuchen einzunehmen, sollten sie dem Spieler folgen oder sich auf ihn zubewegen, allerdings ignorieren dieselben NPCs in vielen Fällen diese Distanz, sollte der Spieler derjenige sein der sich auf sie zubewegt. Oft führt dies dazu, dass der Spieler mit dem Charakter kollidiert, der Charakter herumgeschoben wird oder im schlechtesten Fall, dass der Charakter den Weg des Spielers blockiert und dadurch das Fortschreiten in einem Level oder einer Quest verhindert. In jedem dieser Fälle leidet die Immersion des Spielers, die Präsenz und Wahrnehmung des Charakters aber auch die Qualität des Videospieles darunter.

Etwas Ähnliches gilt für das Blickverhalten der meisten NPCs. Viele NPCs sehen zwar den Spieler an, sobald dieser mit ihnen interagiert, ignorieren danach aber den Großteil des Blickverhaltens des Spielers völlig. Oft halten NPCs dabei den Blickkontakt mit dem Spieler während der kompletten

Interaktion und reagieren nicht auf dessen eventuelle Blickwechsel. Argumentativ ist das zwar besser, als wenn der Charakter keinen Blickkontakt mit dem Spieler sucht, allerdings kann dies bei längeren Interaktionen auch unnatürlich wirken und einen negativen Einfluss auf das letztendliche Spielerlebnis des Spielers haben.

Sowohl die grundsätzlichen Arten und Weisen mit denen NPCs mit einem Spieler interagieren können als auch die Qualität dieser Interaktionen variiert je nach der Darstellungsform der Anwendung und der verwendeten Hardware. Die Darstellungsformen von Videospielen können dabei grob in drei Kategorien aufgeteilt werden:

- „2D“ über einen herkömmlichen Bildschirm
- Virtual Reality (VR)
- Augmented Reality (AR)

Die meisten traditionellen Spiele verwenden das zweidimensionale Bild eines herkömmlichen Bildschirmes, um das Geschehen des Spieles darzustellen. Die Eingaben des Spielers erfolgen hierbei gewöhnlicherweise über eine Tastatur und Maus beziehungsweise über einen Controller. Bei dieser Art von Videospielen kann ein Charakter zwar lebendig wirken, allerdings ist dessen Präsenz limitiert auf Grund der Tatsache, dass die Spielwelt, in der sich der Charakter befindet und mit dem Spieler interagieren kann, rein virtuell ist und der Spieler sich dadurch noch sozusagen getrennt von dieser Welt und dem Charakter befindet.

Bei VR-Spielen hingegen, trägt der Spieler das Ausgabegerät, meistens in der Form eines Head-Mounted Displays (kurz HMD) auf dem Kopf und verwendet einen Controller, oft je Hand, um Eingaben zu tätigen. Das Bild selbst wird dabei mittels zweier Bilder, die von leicht unterschiedlichen Positionen gerendert werden, stereoskopisch ausgegeben. Die meisten VR-Spiele erlauben es den Spielern zudem Bewegungen im Spiel durchzuführen, indem sie sich in der realen Welt bewegen. Die mögliche Präsenz der Charaktere in einem solchen VR-Spiel ist im Vergleich zu herkömmlichen Videospielen höher, da der Charakter und die Welt des Spieles zwar immer noch rein virtuell sind, der Spieler allerdings nun diese virtuelle Welt stereoskopisch wahrnimmt und mit dieser viel direkter interagieren kann.

Die argumentativ höchste Präsenz kann ein virtueller Charakter momentan wahrscheinlich in AR-Anwendungen erreichen. Hauptsächlich da der Spieler in vielen Fällen, wie bei VR-Spielen, ein HMD trägt, allerdings mit dem Unterschied, dass keine komplett virtuelle Welt dargestellt wird, sondern die virtuellen Objekte und Charaktere überlagernd mit der realen Umgebung gerendert werden. Somit kann sich der Spieler ebenfalls in der realen Umgebung bewegen, aber die Interaktion mit den Charakteren erfolgt nun nicht mehr in einer virtuellen, sondern einer realen Umgebung. Die Implementierung und Nutzung von Teilgebieten der nonverbalen Kommunikation kann dementsprechend auch einen erhöhten Einfluss auf die wahrgenommene Präsenz eines Charakters haben.

1.1 ZIELSETZUNG DIESER ARBEIT

Das Thema dieser Arbeit ist das Erstellen einer Augmented-Reality-Anwendung für die HoloLens Mixed-Reality-Brille[1] von Microsoft[2], in welcher die Interaktion und Kommunikation zwischen einem computergesteuerten virtuellen Charakter und dem Spieler glaubwürdig über verschiedene Mittel der nonverbalen Kommunikation stattfinden kann. Das grundlegende Ziel ist es dabei die nonverbale Kommunikation über die drei folgenden Wege zu ermöglichen:

[1] Microsoft HoloLens - https://docs.microsoft.com/de-de/hololens/
[2] Microsoft - https://www.microsoft.com/

- Der Charakter stellt den Blickkontakt mit dem Spieler her, sollte dieser den Charakter ansehen.
- Der Charakter sieht den fokussierten Punkt des Spielers an, sollte dieser einen Punkt im Raum oder ein Objekt für eine kurze Zeit ansehen.
- Der Charakter versucht eine gewisse zwischenmenschliche Distanz zum Spieler einzuhalten, falls dieser zu nahe an den Charakter kommt oder sich zu weit von diesem entfernt.

Dazu werden zwei Systeme designt und implementiert. Zum einen ein Blicksystem, dass es dem Charakter ermöglicht unterschiedliche Blickverhalten ausführen und zum anderen ein System, mittels welchem der Charakter versucht eine bestimmte zwischenmenschliche Distanz zum Spieler einzuhalten.

ABBILDUNG 1: EINE SPIELSITUATION DER ANWENDUNG AUS DER SICHT DES SPIELERS.

1.2 THEMENVERWANDTE ARBEITEN

Die nonverbale Kommunikation trägt nicht nur zum Erlebnis digitaler Unterhaltungsmedien bei, sondern kann auch in ernsteren Zusammenhängen, wie etwa Serious Games, genutzt werden, um bessere Effekte zu erzielen. Auch AR-Technologien können in diesem Kontext genutzt werden. So deutet zum Beispiel die Arbeit von Liu et al. darauf hin, dass AR einen positiven Effekt auf die nonverbale Kommunikation in der Therapie von Autismus-Spektrum-Störungen hat (R. Liu et al., 2017, S. 5) aber auch eventuell im allgemeineren therapeutischen Kontext genutzt werden kann. Vor allem in Kombination mit verschiedenen AR-Technologien kann, bei einer glaubwürdigen Implementation nonverbaler Verhalten eines virtuellen Charakters, davon ausgegangen werden, dass ein verbesserter Effekt in verschiedenen Gebieten erzielt werden kann. Die Ergebnisse der Arbeit von Bailenson et al. weisen darauf hin, dass die Verkörperung eines virtuellen Charakters in Kombination mit glaubwürdigen Verhalten, sowohl die Präsenz dieses Charakters als auch dessen Einfluss auf den Nutzer erhöht. (J. Bailenson et al., 2018, S. 9). Die Darstellung menschlicher virtueller Charaktere in Kombination mit AR ist jedoch bisher weniger verbreitet. (A. Hartholt et al., 2019, S. 1).

Eine themenverwandte Arbeit, in der ebenfalls ein virtueller computergesteuerter Charakter implementiert ist und sich sowohl in einer Mixed-Reality-Umgebung bewegen als auch mit dem Spieler interagieren kann, wird in dem Paper „Welbo: An Embodied Conversational Agent Living in Mixed Reality Space“ von Anabuki et al. beschrieben. Anabuki et al. (2000) implementieren in ihrer Arbeit einen menschenähnlichen Charakter namens „Welbo“ mit einem roboterhaften Aussehen. (S. 1). Die Mixed-Reality-Umgebung in ihrer Anwendung besteht dabei aus einem halb ausgestatteten Raum, in welchem virtuelle Objekte platziert werden. (S. 1). Welbo kann dabei sowohl verbal als auch nonverbal mit dem Spieler kommunizieren. Er kann, unter anderem, eine Reihe an vordefinierten Sätzen sprechen, um Anweisungen zu geben oder seine Gefühle zu vermitteln, auf Anweisungen des Spielers reagieren, um virtuelle Möbel zu bewegen, Gesten während einer Konversation ausführen oder seine Position ändern während er auf Anweisungen wartet und nimmt allgemein auch die Bewegungen und Aktionen des Spielers wahr (S. 2). In ihren Experimenten kommen Anabuki et al., zu den Schlüssen, dass Personen es bevorzugen, wenn sich zum einen Welbos kompletter Körper innerhalb ihres Sichtfeldes befindet und zum anderen, wenn Welbo eine bestimmte Distanz zu ihnen einhält. (S. 2).

Eine weitere Arbeit, in der der Zusammenhang zwischen Blickkontakt und eingehaltener Distanz zwischen einer Person und einem virtuellen Charakter untersucht wird, wird in dem Paper „Equilibrium Theory Revisited: Mutual Gaze and Personal Space in Virtual Environments“ von Bailenson et al. beschrieben. Bailenson et al. (2001) führen in ihrer Arbeit ein Experiment durch, in dem die Probanden eine Art VR-Headset tragen und sich in einem virtuellen Raum mit einem virtuellen Charakter befinden. Die Probanden bekommen dabei die Aufgabe sich bestimmte Eigenschaften des Charakters und Aufschriften auf der Vorder- und Rückseite dessen Shirts zu merken. Die Probanden wissen dabei nicht, dass Bailenson et al. tatsächlich ihre Position und Orientierung aufzeichnen (S. 585). Der Charakter selbst kann verschiedene Level an Blickverhalten ausführen (S. 586), allerdings interagiert jeder Proband nur auf einem dieser Level mit dem Charakter (S. 587). Das Ziel ihres Experimentes ist es die Equilibrium Theory von Argyle und Dean zu testen (S. 583). Die Resultate ihres Experimentes indizieren, dass die Probanden durch den virtuellen Charakter beeinflusst werden und dessen persönlichen Raum mehr beachten als den eines Zylinders in der Kontrollgruppe, aber auch, dass es geschlechtsbedingte Unterschiede bezüglich der Rolle des Blickverhaltens des Charakters gibt (S. 595).

1.3 AUFBAU DIESER ARBEIT

In den nächsten Kapiteln dieser Arbeit, wird zunächst auf verschiedene Grundlagen der nonverbalen Kommunikation und derer Teilgebiete, die grundlegende Ziele dieser Arbeit darstellen, eingegangen. Im Anschluss werden einige Dinge zu den im Kontext der in dieser Arbeit implementierten Anwendung verwendeten Technologien beschrieben. In Kapitel 4 werden verschiedene Anforderung an das Blicksystem und das System zum Einhalten der zwischenmenschlichen Distanz definiert und sowohl das grundlegende Design der beiden Systeme beschrieben als auch wie diese beiden Systeme die jeweiligen Anforderungen erfüllen. In Kapitel 5 wird die letztendliche Implementation der Anwendung und der dafür nötigen Komponenten erläutert. Am Ende werden die Ergebnisse dieser Arbeit diskutiert, mögliche zukünftige Arbeiten beschrieben und eine detailliertere Zusammenfassung dieser Arbeit gegeben.

2 GRUNDLEGENDES ZUR NONVERBALEN KOMMUNIKATION

Die wahrgenommene Präsenz eines virtuellen Charakters kann über unterschiedliche Wege erhöht werden. Zu den umfangreichsten und wahrscheinlich effektivsten Wegen dies zu erreichen zählt sowohl die Interaktion als auch die damit nahezu immer auf eine oder andere Weise in Verbindung stehende Kommunikation mit dem Nutzer und der Umgebung.

Wie bereits erwähnt, kann die Kommunikation in sowohl Videospielen als auch in der Realität verbal und nonverbal stattfinden. Die nonverbale Kommunikation stellt dabei einen wichtigen Bestandteil der alltäglichen Kommunikation zwischen zwei oder mehreren Individuen dar. Neben der Übermittlung von relativ simplen Informationen, wie dem Zeigen in eine Richtung, um die Aufmerksamkeit des Gegenübers zu einem bestimmten Punkt zu lenken, können während der Kommunikation ebenfalls komplexere Informationen, wie der momentane emotionale Zustand einer Person, in vielen Fällen viel leichter nonverbal übermittelt werden, als zu versuchen diese in Worte zu fassen. (J. H. Ellgring, 1986, S. 8).

Kommunikationswissenschaftler sind sich jedoch nicht komplett einig, wie die beste Definition der nonverbalen Kommunikation lautet. (M.L. Patterson, 2009, S. 132). So versucht zum Beispiel die Definition von Patterson ein möglichst breites Spektrum an Phänomenen abzudecken: „[…] the sending and receiving of information and influence through one's immediate environment, appearance cues, and behavior" (M.L. Patterson, 2009, S. 132). Im Vergleich dazu basieren Guye-Vuillème et al. ihre Definition der nonverbalen Kommunikation auf einer Ausschließung: „[…] one defines nonverbal communication as the whole set of means by which human beings communicate except for the human linguistic system and its derivatives […]" (A. Guye-Vuillème et al., 1999, S. 2). Im Gegensatz zur verbalen Kommunikation befasst sich die nonverbale Kommunikation im Allgemeinen also mit nahezu allen Wegen, über die ein Informationsaustausch zwischen zwei oder mehreren Individuen stattfinden kann, ohne diese Informationen selbst mittels den Bedeutungen von Wörtern zu übermitteln.

Im Gegensatz zur verbalen Kommunikation ist der Informationsaustausch durch zumindest einige Aspekte der nonverbalen Kommunikation nahezu unabdinglich. Ein gutes Beispiel dafür erwähnt Gifford. Er erklärt, dass selbst bei einer Unterhaltung zwischen zwei komplett bewegungslosen Personen, die beide keinerlei ausdrucksvolle Bewegungen ausführen, trotzdem ein nonverbaler Austausch von Informationen über ihre Kleidung, Körperhaltung und Distanz zueinander stattfinden würde und selbst wenn sich beide Personen in unterschiedlichen Räumen aufhalten und nur über ein Telefon kommunizieren würden, würde ihre Sprechweise trotzdem noch nonverbal Informationen vermitteln. (R. Gifford, 2011, S. 2). Auch Giri (V. N. Giri, 2009, S. 690) kommt zu einer ähnlichen Folgerung:

> Nonverbal communication is both powerful and indispensable in communication. Our verbal communication would be ineffective if our nonverbal messages did not accompany them. No matter where we look, nonverbal communication is at the heart of every message conveyed or received whether in face-to-face encounters or over the telephone.

Die Vielzahl an Elementen, die in der nonverbalen Kommunikation enthalten sind und deren Ausmaße sind allerdings durch die meisten Definitionen nicht zwingend direkt ersichtlich. Einige gute Beispiele für die Ausmaße liefert Ellgring (1986) in seiner Arbeit „Nonverbale Kommunikation". Er erwähnt dabei zunächst die offensichtlicheren Elemente wie Mimik, Blickverhalten, Gestik, Körperhaltung und Stimme (S. 20), allerdings auch weniger offensichtliche Elemente wie interpersonale Distanz, räumliches Verhalten (S. 20), Gang, Geruch, Körperwärme und Tastempfindung aber auch die Kleidung, Frisur und Gestaltung der persönlichen Umgebung (S. 21). Neben den Ausmaßen zeigen diese Beispiele des Weiteren vor allem zwei Dinge. Zum einen, dass es für eine Person tatsächlich unmöglich ist die nonverbale Kommunikation im Verlauf eines normalen Tages zu vermeiden und zum anderen, dass das Gebiet der nonverbalen Kommunikation, auf Grund dieser Vielzahl an Wegen

nonverbal zu kommunizieren, auch eine Vielzahl an wissenschaftlichen Gebieten umfassen muss. Zu den weitbekanntesten und am meisten verwendeten Gebieten und Mitteln der nonverbalen Kommunikation zählen dabei laut Giri (2009) sowohl Proxemics, Haptics, Oculesics, Chronemics (S. 692) als auch Kinesics (S. 693) und Paralanguage (S. 694) aber auch die Umgebung und das Aussehen einer Person, da sie einen Einfluss auf das Verhalten haben können (S. 693) und somit ebenfalls Teil der nonverbalen Kommunikation sind.

Der Begriff Proxemics ist von Edward T. Hall geprägt und beschreibt laut ihm „the interrelated observations and theories of man's use of space as a specialized elaboration of culture“ (E. T. Hall, 1966, S. 1). Das wissenschaftliche Gebiet der Proxemics befasst sich dabei mit der Relation von menschlichen Körpern im Raum und beschreibt, unter anderem, die Positionierung von Personen zueinander aber auch die Variation in diesen Distanzen auf Grund von Faktoren wie Alter oder Geschlecht. (J. Tanenbaum et al., 2014, S. 18). Eines der grundlegenden Themen dieser Arbeit, die zwischenmenschliche Distanz, fällt ebenfalls in dieses wissenschaftliche Gebiet.

Im Gegensatz dazu beschäftigt sich das Gebiet der Haptics mit dem Berührverhalten in der nonverbalen Kommunikation. (V. N. Giri, 2009, S. 692). Berührungen selbst sind dabei für die soziale Entwicklung eines Menschen notwendig (A. Hans & E. Hans, 2015, S. 48) und sowohl die Art und Weise als auch die Bedeutung einer Berührung sind abhängig vom allgemeinen Kontext, in dem die Interaktion stattfindet und von der Beziehung, die die beteiligten Personen zueinander haben. (V. N. Giri, 2009, A. Hans & E. Hans, 2015). So ist zum Beispiel eine Umarmung im engen Freundeskreis oder der Familie kein Problem, in einem geschäftlichen Meeting allerdings eher fehl am Platz.

Das Gebiet der Oculesics beschäftigt sich mit der Rolle der Augen. (V. N. Giri, 2009, S. 692). Zu diesem Gebiet zählt ein weiteres grundlegendes Thema dieser Arbeit, der Blick beziehungsweise das Blickverhalten einer Person (V. N. Giri, 2009, A. Hans & E. Hans, 2015), aber auch Themen wie die Lidschlagrate oder die Erweiterung der Pupillen. (V. N. Giri, 2009, S. 692).

Die Nutzung der Zeit in der nonverbalen Kommunikation wird in dem Gebiet der Chronemics untersucht. (V. N. Giri, 2009, S. 692). Was unter Nutzung der Zeit verstanden werden kann, beschreibt Giri anhand einiger Beispiele wie folgt: „Our notions of time, how we use it, the timing of events, our emotional responses to time, and even the length of our pauses contribute to the communicative effect of time” (V. N. Giri, 2009, S. 692).

Die Kinesics beschäftigen sich mit der Körpersprache einer Person. (J. Tanenbaum et al., 2014, S. 19). Dazu zählen also sowohl die Haltung und Gesten einer Person (J. Tanenbaum et al., 2014, S. 19) als auch deren körperliche Bewegungen (A. Hans & E. Hans, 2015, S. 47).

Als letztes befasst sich das Gebiet der Paralanguage (auch Vocalics genannt) mit nonverbalen Hinweisen in der Stimme beziehungsweise der Art und Weise wie eine Person etwas sagt. (V. N. Giri, 2009, S. 694). Die nonverbalen Wege der Kommunikation in diesem Gebiet gehen logischerweise meistens Hand in Hand mit der verbalen Kommunikation. Zur Paralanguage zählen dabei Dinge, wie die Tonlage und die Geschwindigkeit, mit der eine Person spricht (V. N. Giri, 2009, S. 694), aber auch Füllwörter, da sie keine spezielle verbale Bedeutung haben, die versucht wird zu übermitteln (Anonymer Autor, 2012, S. 210) [3].

Unabhängig vom wissenschaftlichen Gebiet findet sowohl der Großteil des Sendens als auch des Empfangens von Informationen über die meisten Mittel der nonverbalen Kommunikation unbewusst und automatisch statt. (M. L. Patterson, 2009, S. 133). Ob die unbewusst und bewusst gesendeten Informationen überhaupt auf die richtige Art und Weise vom Empfänger aufgefasst werden oder dieser sie mit einer komplett anderen Bedeutung interpretiert, ist dabei nicht garantiert. So

[3] Der Autor diese Werkes möchte anonym bleiben. Weitere Informationen dazu sind über die Links im Literaturverzeichnis verfügbar.

beschreibt Ellgring (J. H. Ellgring, 1986, S. 14) kurz das auftretende Problem, aber auch die sich dadurch bietende Möglichkeit, wie folgt:

> Die häufig vorhandene Mehrdeutigkeit nonverbaler Signale macht es schwierig, sie valide zu interpretieren. Für die Kommunikation bietet diese Mehrdeutigkeit allerdings gleichzeitig den interessanten Aspekt der "Bedeutungsmöglichkeit": Wir bieten mit dem Verhalten eine Interpretationsmöglichkeit an, ohne daß wir darauf, wie in der Sprache, festgelegt werden könnten. Wir können unser Mißfallen durch ein kurzes mimisches Verhalten kundtun, ohne dem durch Worte Eindeutigkeit zu verleihen. Der Adressat kann diese Bedeutung annehmen, zu einer anderen Interpretation gelangen oder sie übersehen, und dies viel leichter, als er eine sprachliche Äußerung überhören könnte.

Zusätzlich zu dieser Mehrdeutigkeit ist die letztendliche Interpretation der gesendeten Informationen von verschiedenen Faktoren abhängig. Zum einen spielt der Kontext, in dem die Kommunikation stattfindet, eine Rolle (M. L. Patterson, 2009, S. 133), aber auch individuelle Faktoren wie etwa das Geschlecht und die Personalität der beteiligten Personen, die Beziehung zwischen diesen Personen und ihre jeweiligen Kulturen spielen eine Rolle (R. Gifford, 2011, M. L. Patterson, 2009).

Wie am Anfang dieser Arbeit erwähnt, spielt die nonverbale Kommunikation nicht nur in der Realität eine große Rolle. In jeder Implementation eines virtuellen Charakters, in sowohl Videospielen als auch anderen Medien, werden zumindest einige Mittel der Gebiete der nonverbalen Kommunikation eingesetzt, um dem Charakter, bewusst oder unbewusst, verschiedene Eigenschaften zu verleihen. Schon während des Design- und Modellierungsprozesses wird das Aussehen der meisten Charaktere so gewählt, dass deren Persönlichkeit, soziale Position oder allgemeiner deren spätere Rolle in der Geschichte zwar nicht immer direkt erkennbar, allerdings grob klassifizierbar sind. Sehr häufig finden auch die Gebiete der Kinesics und der Paralanguage Anwendung im Erstellungsprozess eines Charakters. So werden die Animationen eines Charakters meistens so designt, dass sie in der späteren Anwendung sowohl die Persönlichkeit als auch Teile der Geschichte des Charakters widerspiegeln und auch der Synchronsprecher eines Charakters wird zumeist so gewählt, dass deren Sprechweise und Stimme mit der Persönlichkeit des Charakters übereinstimmen.

Die nonverbale Kommunikation dient allerdings nicht nur als passives Mittel, um den Charakter lebendiger wirken zu lassen. Einige Videospiele, wie zum Beispiel das Spiel L.A. Noire von Rockstar Games[4], nutzen Aspekte der nonverbalen Kommunikation sogar als Hauptspielelemente. In dieser Art von Spielen werden vor allem das bewusste und unbewusste Senden und die Mehrdeutigkeit der nonverbalen Signale genutzt, um den Spieler vor Entscheidungen zu stellen, die durch die individuelle Interpretation dieser Signale beeinflusst werden.

Da es in dieser Arbeit nicht möglich ist alle Mittel der nonverbalen Kommunikation im Detail zu behandeln, beschreiben die folgenden Unterkapitel hauptsächlich grundlegende Dinge zu den drei Mitteln die zur Erfüllung der Ziele dieser Arbeit nötig sind und in Kombination mit virtuellen Charakteren in Videospielen häufig nur eingeschränkt Anwendung finden: Blickkontakt und Blickverfolgung beziehungsweise allgemeiner Blickverhalten und zwischenmenschliche Distanz.

2.1 BLICKVERHALTEN

Das Herstellen des Blickkontaktes, das Verfolgen des Blickes einer anderen Person aber auch andere Blickverhalten sind wichtige Bestandteile nahezu jeder Interaktion zwischen Personen. (F. Broz et al., 2012, S. 1). Unter Blickkontakt kann allgemein das gegenseitige Ansehen der Augen zwischen zwei Individuen verstanden werden (J. Bailenson et al., 2001, S. 584), obwohl dabei, wie Ellgring (J. H.

[4] Offizielle Website von L.A. Noire - https://www.rockstargames.com/lanoire/

Ellgring, 1986, S. 27) erklärt, nicht zwingend direkt die Augen des Gegenübers selbst angesehen werden müssen:

> Verschiedene experimentelle Untersuchungen zeigen hierzu, daß es nicht möglich ist, einen Blick-Kontakt als Blick von Auge zu Auge von Blicken auf andere Punkte im Gesicht zu unterscheiden. [….] Es ist zwar wahrscheinlich, daß die Augenregion angeblickt wird, es kann aber auch die Nasenwurzel oder, wie von Schwerhörigen berichtet, die Mundregion sein.

Der Blickkontakt erfüllt während einer Interaktion eine Reihe an Funktionen. Eine der wichtigsten ist dabei das Erfassen der Reaktion der gegenüberstehenden Person (M. Argyle & J. Dean, 1965, S. 289). Der Blickkontakt kann allerdings auch Aufschluss über die Absichten und Emotionen der gegenüberstehenden Person geben. So zeigt ein Experiment von Baron-Cohen et al., dass allein die Augen genauso viele Informationen über den mentalen Zustand preisgeben wie das komplette Gesicht und erheblich mehr als der Mund. (S. Baron-Cohen et al., 1997, S. 328). Die Augen werden dementsprechend auch nicht umsonst als „Spiegel der Seele“ bezeichnet. (J. H. Ellgring, 1986, S. 27). Menschen brauchen diese Informationen der Augen, um bei einer Interaktion angemessen funktionieren zu können. (F. Broz et al., 2012, S. 1).

Durch das Sehen in die Richtung eines bestimmten Events oder dem Ansehen eines Objekts zeigt eine Person Interesse an diesem. Verfolgt eine andere Person den Blick einer Person richtet sie ihre Aufmerksamkeit auf das gleiche Event oder Objekt. Dieses Verhalten wird oft auch mit dem Begriff Joint Attention bezeichnet. (K. Marquardt et al., 2017, S. 1). Dies ist vor allem in möglichen Gefahrensituationen nützlich. Etwa im Falle, dass eine Person ein sich schnell näherndes Fahrzeug wahrnimmt, eine zweite Person allerdings nicht, wird die erste höchstwahrscheinlich die zweite zunächst mittels des Greifens oder Vorhaltens eines Armes zum Anhalten bewegen und daraufhin in die Richtung des Fahrzeuges blicken. Die zweite Person wird nach dem Anhalten zuerst die erste Person ansehen, dann deren Blickrichtung wahrnehmen und verfolgen und schlussendlich auch das Fahrzeug sehen, ohne dass die erste den Grund für das Anhalten erklären muss.

Beide Blickverhalten werden also genutzt, um, wie bei allen Mitteln der nonverbalen Kommunikation, Informationen zu vermitteln und zu sammeln oder wie es Broz et al. (F. Broz et al., 2012, S. 1) beschreiben:

> Gazing and the ability to follow the eye gaze of others enables us to communicate non-verbally and improves our capacity to live in large social groups. It serves as a basic form of information transmission between individuals which understand each other as intentional agents.

Blickverhalten allgemein sind laut Ellgring allerdings nicht unabdinglich. (J. H. Ellgring, 1986, S. 31). Als Begründung dafür benutzt er die Interaktion mit einer blinden Person, bei der natürlich Blickverhalten, wenn überhaupt, nur eingeschränkt oder unbeabsichtigt ausgeführt werden. Blickverhalten während einer Interaktion haben dabei für ihn sowohl eine vermittelnde als auch eine regulative Funktion. (J. H. Ellgring, 1986, S. 31).

Die Implementation von Blickverhalten in Bezug auf virtuelle Charaktere ist jedoch mit einer Reihe von Problemen behaftet. Eines der größten Probleme, das bei der Nutzung von Blickverhalten im Zusammenhang mit menschenähnlichen Charakteren auftreten kann, wird von Broz et al. (F. Broz et al., 2012, S. 1) beschrieben:

> Having the capability of producing readable gaze behavior may lead humans to expect these agents to exhibit natural and/or meaningful gaze, and the quality of interaction may be reduced if these expectations are not met.

Dementsprechend kann auch in Anwendungen, die zwar verschiedene Blickverhalten implementieren, diese aber nicht natürlich wirkend einsetzen oder wechseln, die Qualität der kompletten Anwendungen darunter leiden oder zumindest der Charakter als weniger „echt“

wahrgenommen werden. Sollte ein NPC zum Beispiel nicht auf ein Event in der Spielwelt, wie einer weit entfernten Explosion oder ein aus einem Busch springendes Monster, reagieren, indem er seine Aufmerksamkeit durch einen Blick in die Richtung des Events dorthin richtet, wird sich dies definitiv auf den Spieler auswirken. Die Immersion des Spielers wird sich dementsprechend verringern und er wird den Charakter als weniger präsent wahrnehmen.

Viele Blickverhalten sind dabei ebenfalls abhängig von anderen Ereignissen in der Umgebung einer Person. Um die Blickverhalten eines NPCs natürlich wirkend wechseln zu können, sollten also ebenfalls sowohl die Blickverhalten des Individuums mit dem der NPC interagiert (F. Broz et al., 2012, S. 6) als auch verschiedene Ereignisse in der Umgebung des NPCs miteinbezogen werden. So ist etwa die Wahrscheinlichkeit in die Augen einer anderen Person zu sehen höher, wenn diese einen selbst bereits ansieht, als wenn sie in eine komplett andere Richtung sieht.

2.2 ZWISCHENMENSCHLICHE DISTANZ

Genauso wie verschiedene Blickverhalten, stellt das Einhalten einer bestimmten Distanz zu anderen Personen ebenfalls ein wichtiges Mittel zur Kommunikation über nonverbale Wege dar. Die zwischenmenschliche Distanz (auch interpersonale Distanz genannt) kann Aufschluss über selten verbal angesprochene Aspekte der Beziehung zwischen den beteiligten Personen, wie etwa Intimität, Zuneigung, Status und Macht, geben. (J. H. Ellgring, 1986, S. 34). Die tatsächliche Distanz, die dabei gegenüber anderen Menschen eingenommen wird, variiert, wie viele andere Mittel der nonverbalen Kommunikation auch, auf Grund von Faktoren wie dem kulturellen Einfluss oder der Intimität der beteiligten Personen und findet zum größten Teil unbewusst statt. (J. Tanenbaum et al., 2014, S. 19). Die zwischenmenschliche Distanz wird von Gifford sogar als eines der besten Beispiele für den Einfluss der Kultur auf nonverbale Aktionen genannt. (R. Gifford, 2011, S. 20).

Eine der ersten Definitionen und Aufteilungen der Distanzen, die in unterschiedlichen sozialen Situationen eingehalten werden, sind die Distanzzonen die Edward Hall (1966) in seinem Buch „The Hidden Dimension“ definiert und vorstellt. Er teilt dabei die Distanzen die Menschen zueinander einnehmen in vier allgemeine Zonen ein (S. 114):

- Intimate Distance
- Personal Distance
- Social Distance
- Public Distance

Er definiert für jede dieser Zonen zusätzlich eine „Close-Phase“ und eine „Far-Phase“ (S. 114), die jeweils die untere und obere Grenze der jeweiligen Zone darstellen, aber gleichzeitig auch jede Zone nochmals in zwei kleinere Intervalle unterteilen. Er erwähnt allerdings auch, dass diese Zonen kein allgemeines Verhalten darstellen und nur für die Personen in seinen Experimenten gelten (S. 116). Dennoch können diese vier Distanzzonen und ihre Grenzen als erste Approximation verwendet werden.

Intimate Distance

Die Zone der Intimate Distance stellt die Zone dar, bei der Menschen die kleinste Distanz zueinander haben. Die Close-Phase dieser Zone wird von Hall wie folgt beschrieben: “This is the distance of love-making and wrestling, comforting and protecting. Physical contact or the high possibility of physical involvement is uppermost in the awareness of both persons.” (E. T. Hall, 1966, S. 117). Die Distanz dieser Phase liegt bei bis zu 6 Zoll (etwa 15 cm). Die Far-Phase dieser Zone befindet sich nach Hall in einer Distanz von 6 bis 18 Zoll (etwa 15 - 46 cm), wobei sich die Gliedmaßen der gegenüberstehenden Person mit den Händen leicht berühren lassen. (E. T. Hall, 1966, S. 117).

Personal Distance

Die Zone der Personal Distance kann man sich nach Hall (1966) wie eine schützende Blase zwischen sich selbst und anderen vorstellen (S. 119). Während man in der Close-Phase, mit einer Distanz von 1,5 bis 2,5 Fuß (etwa 46 - 76 cm), die andere Person halten kann (S. 119), ist das Berühren in der Far-Phase, mit einer Entfernung von 2,5 bis 4 Fuß (etwa 76 cm - 1,2 m), möglich, wenn beide Personen ihre Arme ausstrecken (S. 120).

Social Distance

Hall (1966) bezeichnet die Social Distance als die Distanzzone, in der keine Berührung stattfindet und niemand erwartet eine andere Person zu berühren (S. 121). Hierbei finden sowohl in der Close-Phase (4 - 7 Fuß, etwa 1,2 - 2,1 m) (S. 121) als auch in der Far-Phase (7 - 12 Fuß, etwa 2,1 - 3,7 m) (S. 122) unpersönliche Angelegenheiten statt. Obwohl diese laut Hall in der Far-Phase formeller wirken (S. 122). Auch das Halten des Blickkontaktes ist laut ihm in dieser Distanzzone wichtiger als bei näheren Distanzen (S. 122).

Public Distance

Nach Hall (1966) liegt die Public Distance weit außerhalb der Distanz in der Personen direkt miteinander involviert sind (S. 123). Wobei er die Close-Phase (12 - 25 Fuß, etwa 3,7 - 7,6 m) als Abstand bezeichnet in der, falls nötig, rechtzeitig Ausweich- und Verteidigungsaktionen ausgeführt werden können (S. 123). Dagegen ist die Far-Phase (ab 25 Fuß, etwa 7,6 m) die Distanz, die oft von Person des öffentlichen Lebens eingenommen wird (S. 124).

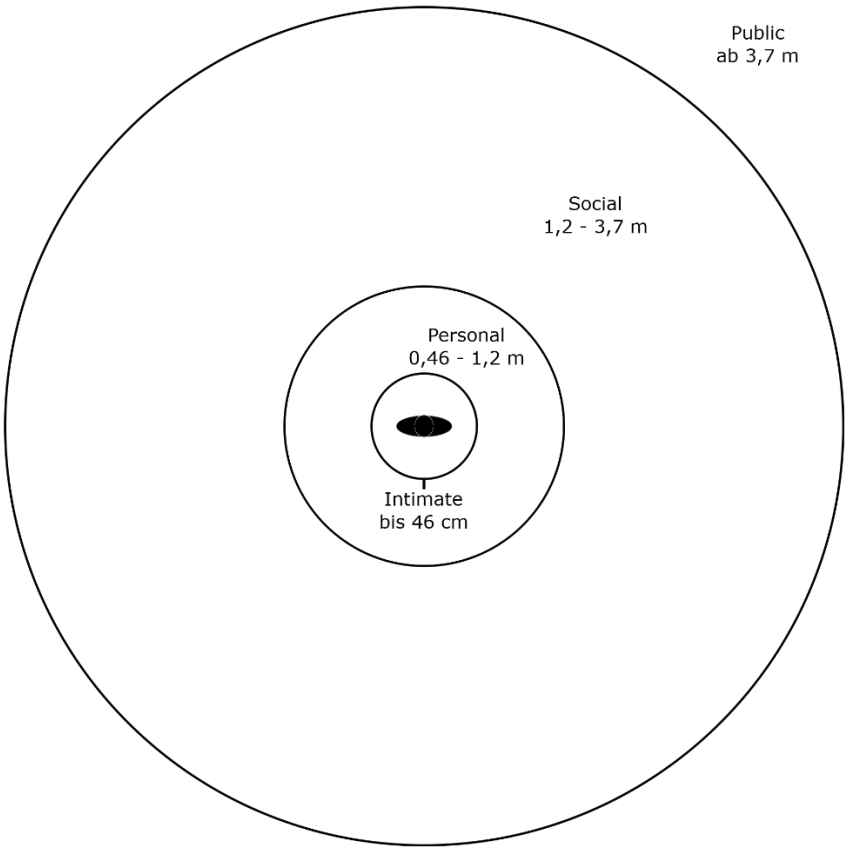

ABBILDUNG 2: VISUALISIERUNG DER DISTANZZONEN NACH HALL.

Jede Implementation eines virtuellen Charakters erzeugt bereits grundlegende Distanzverhalten. (J. Tanenbaum et al., 2014, S. 19). Dennoch kann ein virtueller Charakter vom Einhalten einer gewissen zwischenmenschlichen Distanz profitieren. Neben den offensichtlichen Vorteilen auf die allgemeine Präsenz des Charakters würde die Implementation einer solchen Distanz vor allem für Videospiele verschiedene Vorteile bieten. Zum Beispiel können die in der Einleitung erwähnten auftretenden Probleme bei der Interaktion zwischen Spielern und Charakteren in den meisten Fällen gelöst werden, indem der Charakter versucht eine bestimmte Distanz einzuhalten.

2.3 EQUILIBRIUM THEORY

Die Menge an Blickkontakt zwischen zwei Personen und deren eingenommene Distanzen stehen in einer engen Beziehung zueinander (J. H. Ellgring, 1986, S. 36) und sind zumindest teilweise voneinander abhängig. So zeigt sich in beiden dieser Mittel der nonverbalen Kommunikation sowohl die Nähe als auch die Vertrautheit der beteiligten Personen. (J. Bailenson et al., 2001, S. 583). Wie es Tanenbaum et al. beschreiben bringt dabei die Equilibrium Theory die beiden Aspekte in Verbindung miteinander und wird dabei zuerst in einer Arbeit von Argyle und Dean im Jahr 1965 vorgeschlagen und erforscht. (J. Tanenbaum et al., 2014, S. 19).

Argyle und Dean (1965) führen in ihrer Arbeit mehrere Experimente durch, um zum einen den Effekt von Blickkontakt auf den Ausgleich der Distanz (S. 294) und zum anderen den Effekt von Distanz auf Blickkontakt (S. 296) zu messen. Die Ergebnisse ihrer Experimente fassen Argyle und Dean (M. Argyle & J. Dean, 1965, S. 304) am Ende ihrer Arbeit wie folgt zusammen:

> It is postulated that if this equilibrium is disturbed along one of its constituent dimensions, e.g., by increasing physical proximity, there will be compensatory changes along the other dimensions. It has already been shown that greater intimacy of topic leads to less eye-contact. We have now shown that reducing eye-contact makes greater proximity possible, and that greater proximity reduces eye-contact.

Nach Tanenbaum et al. schlägt die Equilibrium Theory dabei im Wesentlichen vor, dass, zusätzlich zum Zeigen der Intimität mittels Blickkontakt und eingenommener Distanz, Personen versuchen ein individuelles Gleichgewicht aus Distanz und Blickkontakt zu erreichen. (J. Tanenbaum et al., 2014, S. 19).

Fehlt die Regulierung des Blickkontaktes auf Grund der Distanz bei einem virtuellen Charakter, beeinflusst dies höchstwahrscheinlich unbewusst die wahrgenommene Präsenz dieses Charakters auf eine negative Weise. Folglich kann die Implementation der Equilibrium Theory eventuell den gegengesetzten Effekt erzielen, also positive Auswirkungen auf die Präsenz eines virtuellen Charakters haben.

3 VERWENDETE TECHNOLOGIEN

3.1 AUGMENTED REALITY

Die Darstellung der Abläufe in der Szene der erstellten Anwendung findet mittels Augmented Reality statt. Im Vergleich zu VR, bei der eine virtuelle Welt und deren Objekte stereoskopisch gerendert und wahrgenommen werden, kann unter AR, allgemein gesehen, die computergestützte Ergänzung und Erweiterung der Wahrnehmung verstanden werden. (R. T. Azuma, 1997, S. 2). Praktisch gesehen, können dabei virtuelle Objekte, mittels verschiedener AR-Technologien, stereoskopisch in der realen Umgebung gerendert werden. In vielen AR-Spielen und Anwendungen ist diese Erweiterung allerdings limitiert auf das Überlagern der realen Umgebung mit virtuellen Objekten. Die tiefgreifendere

Integration in die reale Umgebung, wie etwa das Verdecken von virtuellen mit realen Objekten, stellt dabei eigentlich einen Teil des Gebietes der Mixed Reality (MR) dar. Obwohl die beiden Begriffe Augmented Reality und Mixed Reality also theoretisch leicht unterschiedliche Gebiete beschreiben und AR normalerweise einen Teil der MR darstellt, werden die beiden Begriffe in der Praxis oft synonym miteinander verwendet, daher werden sie in dieser Arbeit ebenfalls synonym verwendet.

Im Vergleich zur Darstellung über einen normalen Monitor kann die Verwendung von AR eine Reihe an Vorteilen für die nonverbalen Verhalten während einer Interaktion mit sich bringen. Einen der Hauptvorteile erwähnen Hartholt et al. (A. Hartholt et al., 2019, S. 2):

> In AR, the user experiences the presence of characters in an immersed environment rather than through a 2D monitor, which means that eye contact between the character and the user can no longer be faked. When using a monitor, the character can be set up to look at the virtual camera in the game engine, making it appear as if they always look at anyone in the room. In AR […], the user basically is the camera, moving around rather than being fixed.

Für den Spieler in einer AR-Anwendung wirkt es in anderen Worten also immer so, als ob ihn der Charakter direkt ansehen würde, sobald dieser Charakter die Kamera ansieht. Zu den weiteren Vorteilen zählen vor allem auch die verbesserte Wahrnehmung der Größenverhältnisse von Gegenständen und Charakteren, auf Grund des stereoskopischen Renderns und die mögliche Interaktion zwischen virtuellen Charakteren und der realen eventuell vertrauten Umgebung. Vor allem letzteres kann natürlicherweise einen großen Einfluss auf die wahrgenommene Präsenz eines Charakters haben, sollte dieser Charakter glaubhaft mit der realen Umgebung interagieren können. Viele dieser möglichen Vorteile können aber auch negative Auswirkungen haben, falls die implementierten Verhalten nicht glaubwürdig wirken. Sollte der Charakter zum Beispiel versuchen die Distanz zum Spieler einzuhalten und währenddessen eine Wand des Raumes ignorieren und sich durch diese hindurchbewegen wird sich dies definitiv negativ auf die wahrgenommene Präsenz des Charakters auswirken. Inwiefern die meisten durch AR ermöglichten Verhalten überhaupt implementiert werden können ist dabei ebenfalls stark abhängig von der verwendeten Hard- und Software.

3.2 MICROSOFT HOLOLENS

Die grafische Ausgabe der Szene erfolgt über die erste Generation der HoloLens. Das Nutzen der HoloLens erlaubt es dem Benutzer sich nahezu uneingeschränkt in der Umgebung zu bewegen, da jegliche benötigte Hardware zum Ausführen einer AR-Anwendung in die Brille selbst integriert ist und daher während der Verwendung keinerlei physische Verbindungen zu anderen Geräten, wie etwa Kabel zur Stromversorgung oder der Bild- und Tonübertragung, nötig sind. Dies ist besonders wichtig, um sicherzustellen, dass das Erlebnis des Spielers nicht auf Grund von zusätzlichen Einschränkungen bezüglich der Bewegungsfreiheit beeinflusst wird.

Die Interaktion zwischen der virtuellen und der physischen Welt wird von der HoloLens auf verschiedene Weisen ermöglicht. Eine der wahrscheinlich am meisten verwendeten Funktionen der HoloLens, um dies zu erreichen, ist das Ausführen von Eingaben in der Form von Gesten. Die HoloLens erlaubt es dem Nutzer dabei die sogenannten „Air Tap“ und „Bloom“ Gesten mit einer Handbewegung auszuführen, um Eingaben zu tätigen. Diese können in Anwendungen, ähnlich zu Eingaben über eine Maus oder Tastatur, genutzt werden, um verschiedene Funktionen auszuführen und somit den Spieler mehr oder weniger direkt mit den virtuellen Objekten interagieren zu lassen.

Eine weitere Funktion der HoloLens ist das Nutzen der realen Umgebung innerhalb einer Anwendung. Die meisten Anwendungen ermöglichen dadurch die Interaktion von virtuellen Charakteren und Objekten mit Gegenständen in der realen Umgebung. Dazu scannt die HoloLens die Umgebung nach und nach und wandelt die dadurch erhaltenen Informationen in eine virtuelle Repräsentation der

realen Umgebung um. Dieses 3D-Objekt kann danach in einer Anwendung wie ein herkömmliches Mesh genutzt werden, um Objekte damit kollidieren oder auf andere Arten interagieren zu lassen. Das generierte Mesh repräsentiert die reale Umgebung allerdings nicht perfekt, es verbessert sich jedoch, bis zu einem gewissen Grad, schrittweise mit der Zeit, die gescannt wird.

Eine ebenfalls von der HoloLens unterstützte Funktion ist das Verdecken von virtuellen Objekten der Szene mit realen Objekten der Umgebung. Die HoloLens ermöglicht dies, indem virtuelle Objekte über ein transparentes Display angezeigt werden. Die Transparenz des Displays ist abhängig von der Helligkeit des anzuzeigenden Pixels, wobei die Transparenz abnimmt umso heller der Pixel an dieser Stelle ist. Das Licht der Szene wird also in anderen Worten auf das Licht der echten Umgebung addiert. Schwarze Pixel sind dementsprechend komplett transparent. In Kombination mit dem generierten Mesh der Umgebung oder anderen virtuellen Objekten, können damit bestimmte Objekte teilweise oder komplett verdeckt werden, um den Eindruck zu verleihen, dass sich das Objekt hinter zum Beispiel einer Wand oder unter einem Tisch befindet.

3.3 UNITY GAME ENGINE UND HOLOTOOLKIT

Die Engine die für die Implementation der Anwendung verwendet wird, ist die Unity Game Engine[5]. Unity unterstützt, unter anderem, die Programmiersprache C# und verwendet Komponenten, um den verschiedenen Objekten der Szene Funktionalitäten und andere Eigenschaften zuzuweisen. Erstellte Klassen können von einer bereitgestellten Klasse abgeleitet werden, um als Komponenten an ein Objekt gehangen werden zu können. Da Komponenten dementsprechend lediglich aus Klassen bestehen werden in dieser Arbeit die beiden Begriffe Klasse und Komponente ebenfalls synonym miteinander verwendet.

Durch das Nutzen von Unity bietet sich außerdem die Möglichkeit das HoloToolkit[6] zu nutzen. Das HoloToolkit ist eine ältere Version des von Microsoft getriebenen Mixed Reality Toolkits[7]. Es fügt zahlreiche Funktionen und Komponenten in Unity hinzu, welche das Arbeiten mit der HoloLens, in vielen Fällen, stark erleichtern. Die mitgelieferten Assets enthalten, unter anderem, die Funktionen die Kamera und die Szene selbst für die Nutzung mit der HoloLens zu konfigurieren, Eingaben durch Gesten zu erhalten, aber auch verschiedene Komponenten, die sich um das Verarbeiten der von der HoloLens gelieferten Umgebungsinformationen kümmern.

Die Spatial-Mapping-Komponenten kümmern sich dabei hauptsächlich um das Umwandeln der von der HoloLens gescannten Umgebung in mehrere kleine Objekte in der Szene der Anwendung und deren Verwaltung. Diese einzelnen Objekte stellen dabei Teile des kompletten Meshes der Umgebung dar und werden ebenfalls jeweils mit Collidern versehen, damit Objekte mit diesen kollidieren und interagieren können.

Mittels der Spatial-Understanding-Komponenten kann das Mesh der Umgebung weiterverarbeitet werden. So können zum Beispiel Löcher im Mesh, die in nicht gescannten Bereichen auftreten, gefüllt oder die Art einer Fläche in diesem Mesh bestimmt werden, aber auch die Abfrage einer geeigneten Position, um ein Objekt zu platzieren, kann über diese Komponenten getätigt werden.

Über die Spatial-Processing-Komponenten kann das Mesh ebenfalls weiterverarbeitet werden. Im Vergleich zu der Weiterverarbeitung durch die Spatial-Understanding-Komponenten, kann das Mesh durch diese Komponenten auf einige größere Planes reduziert werden. Zum einen werden dadurch Flächen in der Umgebung, wie Wände oder Böden, welche nicht zwingend viele Polygone benötigen,

[5] Unity - https://unity.com/
[6] HoloToolkit 2017.4.3.0 - Refresh - https://github.com/microsoft/MixedRealityToolkit-Unity/releases/tag/2017.4.3.0-Refresh
[7] Mixed Reality Toolkit - https://github.com/microsoft/MixedRealityToolkit-Unity

durch eine einzige große Plane dargestellt und zum anderen werden diese Flächen dadurch geglättet, um eventuelle Unebenheiten im gescannten Mesh zu beheben.

3.4 NAVIGATION MESH

Die Bewegung des Charakters, in der implementierten Anwendung, erfolgt mit der Hilfe eines Navigation Meshes, kurz auch Navmesh genannt. Ein Navmesh besteht dabei aus einem Polygonmesh und stellt die Fläche dar, in der sich ein NPC bewegen kann. Das Finden eines Pfades auf diesem Mesh kann mittels Pfadfindungsalgorithmen, wie etwa A*, erfolgen.

Unity stellt allgemein Funktionen bereit, um ein Navmesh für verschiedene Szenen und Arten von NPCs zu erstellen. Einem NPC kann dabei die NavMeshAgent-Komponente angehangen werden, um diesem die Fähigkeit zu geben sich auf dem Navmesh zurechtzufinden und einen Pfad zu einer Zielposition zu berechnen.

Durch die Komponente NavMeshObstacle können sowohl statische als auch dynamische Objekte als Hindernisse auf dem Navmesh gekennzeichnet werden. Je nach den Einstellungen dieser Komponente und der NavMeshAgent-Komponente eines Charakters kann dieser mehr oder weniger gut dem Hindernis ausweichen und um dieses herum manövrieren.

Nachdem das Mesh der Umgebung beim Start der Anwendung noch nicht existiert, kann das Navmesh, auf dem sich der Charakter bewegen kann, nicht direkt am Anfang erstellt werden, sondern es muss darauf gewartet werden, dass das Scannen der Umgebung abgeschlossen ist. Das Erstellen des Navmeshes zur Laufzeit ist mit den von Unity gelieferten Funktionen jedoch nicht vorgesehen. Um dies dennoch zu ermöglichen werden zusätzlich die von Unity Technologies bereitgestellten NavMeshComponents[8] genutzt, welche nicht in der Standardinstallation von Unity enthalten sind. Diese enthalten, unter anderem, die Komponente NavMeshSurface mit der das Generieren eines Navmeshes zur Laufzeit möglich ist.

4 ANFORDERUNGEN UND DESIGN

Sowohl die Implementation des Blicksystems als auch das System der zwischenmenschlichen Distanz müssen in dieser Arbeit eine Reihe an Anforderungen erfüllen, um brauchbar zu sein, die gewünschten grundlegenden Ziele zu erfüllen und vor allem, natürlich zu wirken. Die Definition der dafür nötigen Anforderungen, die die beiden Systeme mehr oder weniger unabhängig von der Implementation erfüllen müssen, die Beschreibung des grundlegenden Designs der beiden Systeme und wie das jeweilige Design die Anforderungen erfüllt, erfolgt in den folgenden Unterkapiteln.

4.1 ANFORDERUNGEN AN DAS BLICKSYSTEM

Zu den Anforderungen, die das Blicksystem erfüllen muss, zählt zunächst einmal, dass der Charakter die beiden grundlegenden Blickverhalten, den Spieler anzusehen und den Blick des Spielers zu verfolgen, ausführen kann. Da es im echten Leben allerdings nicht garantiert ist, dass die gegenüberstehende Person den Blickkontakt erwidert oder einen Blick in Richtung des fokussierten Punktes wirft, sollte zu den beiden Blickverhalten nur mit einer bestimmten Wahrscheinlichkeit gewechselt werden. Daraus folgend sind die ersten beiden Anforderungen wie folgt definiert:

[8] Unity Technologies GitHub - NavMeshComponents - https://github.com/Unity-Technologies/NavMeshComponents

1. Der Charakter sieht den Spieler mit einer bestimmten Wahrscheinlichkeit an, sobald dieser ihn ansieht.

2. Der Charakter verfolgt den Blick des Spielers mit einer bestimmten Wahrscheinlichkeit, sobald dieser einen Punkt fokussiert.

Da es nicht natürlich wirken würde, wenn der Charakter ununterbrochen zwischen dem Ansehen des Spielers und dem Ansehen dessen fokussierten Punktes wechseln würde und dies oft auch nicht möglich ist, da der Spieler nicht zwingend einen Punkt fokussiert, muss mindestens ein weiteres Blickverhalten implementiert werden, in welches der Charakter wechseln kann, sobald der Wechsel in kein anderes Blickverhalten möglich ist. Dieses Blickverhalten dient damit sozusagen als Standardblickverhalten:

3. Es gibt ein Standardblickverhalten, in welches der Charakter wechseln kann, sobald der Wechsel zu keinem anderen Blickverhalten möglich ist.

Unabhängig davon welches Blickverhalten eine Person momentan ausführt, wird zu jedem Zeitpunkt entweder ein anderes Lebewesen, ein Objekt oder zumindest ein bestimmter Punkt im Raum angesehen:

4. Jedes Blickverhalten hat ein Objekt oder eine Zielposition, die angesehen wird.

Damit eine Person überhaupt beurteilen kann, ob eine andere Person ein bestimmtes Objekt oder einen bestimmten Punkt des Raumes fokussiert, muss diese Person normalerweise zunächst einmal in die Richtung der Augen der anderen Person sehen, um die Blickrichtung dieser Person erkennen zu können (K. Marquardt et al., 2017, S. 2):

5. Der Wechsel zum Ansehen des vom Spieler fokussierten Punktes ist nur dann möglich, wenn der Charakter zunächst den Spieler ansieht.

Die Zeit, die eine Person einen bestimmten Punkt oder ein bestimmtes Objekt am Stück ansieht, ist immer begrenzt. Im Falle des Ansehens einer anderen Person ist ein zu langer Blickkontakt im besten Fall unangenehm und sollte daher vermieden werden:

6. Die Zeit, die ein Charakter am Stück in einem bestimmten Blickverhalten verbringen kann, ist zeitlich begrenzt.

Sollte das Blickverhalten gewechselt werden, wird normalerweise erst einmal eine kurze Zeit im neuen Blickverhalten verbracht. Die Wahrscheinlichkeit des Wechsels des Blickverhaltens ist also zunächst recht klein und wächst mit der in diesem Blickverhalten verbrachten Zeit:

7. Die Wahrscheinlichkeit des Charakters, von einem Blickverhalten in ein anderes zu wechseln, wächst mit der in dem momentanen Blickverhalten verbrachten Zeit.

Die achte Anforderung an die Implementation des Blicksystems entsteht durch die Equilibrium Theory. Durch diese sollte die Wechselwahrscheinlichkeit zum Blickverhalten, in dem der Spieler angesehen wird, davon abhängen, wie nahe sich der Spieler am Charakter befindet, wobei eine nähere Distanz im Allgemeinen mit einer Abnahme der Wahrscheinlichkeit verbunden ist und eine höhere Distanz mit einer Erhöhung dieser:

8. Die Wahrscheinlichkeit, dass der Charakter den Spieler ansieht, ist abhängig von der Distanz, die der Spieler zum Charakter hat.

Beim Blickwechsel zu einem beliebigen Raumpunkt werden, vereinfacht gesehen, zuerst die Augen rotiert bis sie eine bestimmte Rotation erreicht haben. Darauf folgt mit einem ähnlichen Ablauf der Rest des Körpers, also der Kopf, der Hals, die obere und dann die untere Hälfte des Oberkörpers, bis hin zur Hüfte. Sobald die Hüfte erreicht ist, rotiert sich eine Person, falls nötig, auch mit dem

kompletten Rest des Körpers mit. Zudem können sich die hierfür verwendeten Körperteile nur um einen bestimmten Winkel drehen, bevor sie ein Limit erreichen und haben meisten auch eine Limitierung der Geschwindigkeit, mit der sie sich drehen können. Hieraus schließen sich die neunte, zehnte und elfte Anforderung:

9. Die Körperteile, die der Charakter benutzt, um in die Richtung des Zieles zu sehen, haben jeweils maximale Winkel, die sie sich von ihrer jeweiligen initialen Ausrichtung, in eine bestimmte Richtung rotieren können.

10. Die jeweiligen Körperteile des Charakters rotieren nur in die Richtung des Zieles, sobald alle vorherigen Körperteile mindesten einen ihrer maximalen Winkel erreicht haben oder die daraus folgende Rotation näher an der initialen Ausrichtung des Körperteiles liegt.

11. Die Körperteile des Charakters haben eine maximale Geschwindigkeit, mit der sie sich drehen können.

Die Wahrnehmung des Blickes einer anderen Person, ist in der Regel nur möglich, falls sich die Augen dieser Person innerhalb des eigenen Sichtfeldes befinden und ist umso wahrscheinlicher desto mehr sich diese Augen im Fokus des eigenen Blickes befinden:

12. Die Wahrscheinlichkeit den Blick des Spielers wahrzunehmen ist umso größer, desto kleiner der Winkel zwischen der Blickrichtung des Charakters und der Richtung zum Spieler ist.

Es können bestimmt noch eine Vielzahl an weiteren Anforderungen formell definiert werden, allerdings wird sich in dieser Arbeit auf die hier definierten zwölf Anforderungen beschränkt, da sie grundlegende Anforderungen darstellen die, unabhängig von der letztendlichen Implementation, erfüllt werden sollten und eine vollständige und detailliertere Liste den Rahmen dieser Arbeit sprengen würde. Dazu kommt, dass argumentiert werden kann, welche Anforderungen und Verhalten tatsächlich einen Teil des Blickes beziehungsweise des Blicksystems darstellen sollten und der letztendliche Umfang dieser auch von der jeweiligen Implementation abhängig ist. So ist in dieser Arbeit zum Beispiel der Lidschlag des Charakters ebenfalls implementiert, wird aber nicht als direkter Teil des Blicksystems gesehen.

4.2 GRUNDLEGENDES DESIGN DES BLICKSYSTEMS

Um die ersten vier Anforderungen an das Blicksystem zu erfüllen, zugleich das Implementieren der verschiedenen Blickverhalten zu erleichtern und zu vermeiden das derselbe Code öfters geschrieben werden muss, ist zuerst einmal eine Komponente nötig, die die Blickverhalten im Allgemeinen darstellt. Diese Basisklasse der Blickverhalten stellt dabei sechs Grundfunktionen mit den folgenden Aufgaben bereit:

1. Das Liefern der Wahrscheinlichkeit von diesem Blickverhalten zu einem anderen zu wechseln.
2. Das Liefern der Wahrscheinlichkeit in dieses Blickverhalten zu wechseln.
3. Die Rückgabe eines Boolean, ob aus diesem Blickverhalten gewechselt werden sollte.
4. Die Aktualisierung des Blickzieles.
5. Die Rückgabe des Blickzieles.
6. Das allgemeine Aktualisieren des Blickverhaltens, inklusive dem Rufen der vierten Funktion, Prüfen der ersten Funktion und eventuelle Setzen des Boolean den die dritte Funktion zurückgibt.

Die einzelnen Komponenten, die die drei grundlegenden Blickverhalten darstellen, werden von dieser Basisklasse abgeleitet und überschreiben einige der erwähnten Grundfunktionen mit ihren eigenen Definitionen, um dadurch die für sie relevanten Werte zurückzuliefern.

Die fünfte Anforderung könnte erfüllt werden, indem nur das Standardblickverhalten eine Limitation der Blickverhalten, in die von diesem gewechselt werden kann, bekommt. Allerdings könnten deswegen, bei der Implementation weiterer Blickverhalten, Probleme entstehen und es ist daher einfacher und allgemein von mehr Nutzen, diese Limitation in allen Blickverhalten zu ermöglichen. Hierzu wird eine Liste von Blickverhalten zu der Basisklasse hinzugefügt. Die Blickverhalten dieser Liste stellen dabei alle möglichen Blickverhalten dar, in die von dem momentanen gewechselt werden kann. Die von der Basisklasse erbenden Klassen fügen dementsprechend individuell die von ihnen aus möglichen nächsten Blickverhalten in ihre Liste ein.

Um die Zeit, die maximal in einem Blickverhalten verbracht werden kann, zu limitieren, enthält die Basisklasse eine Variable, die diese maximale Zeit darstellt. Wird diese Zeit überschritten kann der Wechsel zu einem anderen Blickverhalten zwangsweise erfolgen, um die sechste Anforderung zu erfüllen.

Anforderung sieben wird erfüllt, indem in der Funktion zur Berechnung der Wechselwahrscheinlichkeit in eines der nächsten Blickverhalten, die berechnete Wahrscheinlichkeit mit einem weiteren Wert multipliziert wird. Dieser weitere Wert repräsentiert die Wahrscheinlichkeit auf Grund der in diesem Verhalten verbrachten Zeit zu wechseln und ermittelt sich aus der Teilung der momentan verbrachten Zeit durch die maximale Zeit und anschließenden Limitierung auf einen Wert zwischen 0,0 und 1,0.

Damit die achte Anforderung erfüllt und damit die Equilibrium Theory eingebracht wird, wird in der Funktion des Blickverhaltens, bei dem der Spieler angesehen wird, die die Wechselwahrscheinlichkeit zu diesem liefert, die berechnete Wahrscheinlichkeit ebenfalls mit einem Wert multipliziert, welcher die Wahrscheinlichkeit den Spieler auf Grund der Entfernung anzusehen darstellt und von der Komponente der zwischenmenschlichen Distanz geliefert wird.

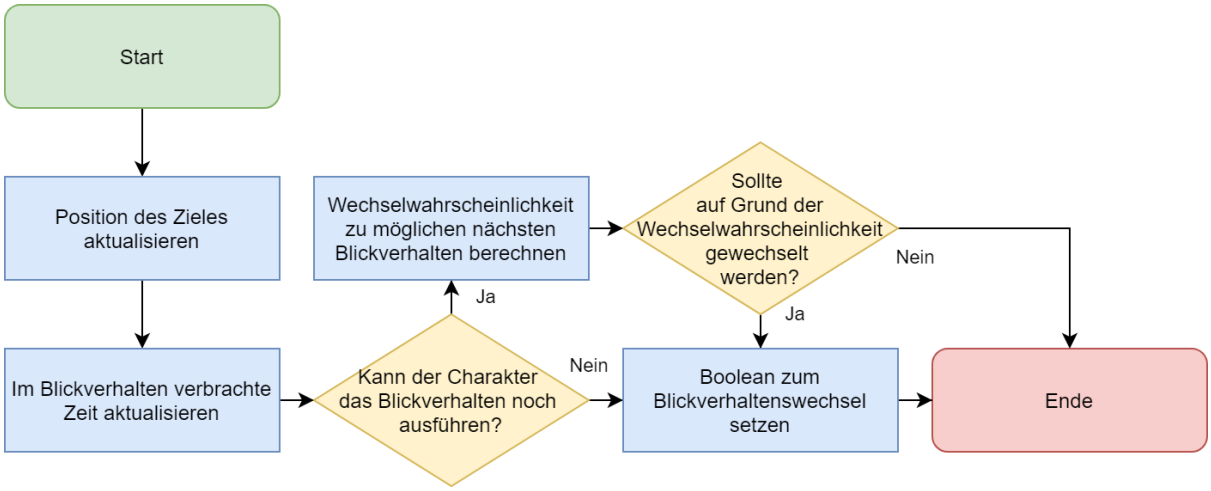

ABBILDUNG 3: EIN FLUSSDIAGRAMM DAS DEN ABLAUF BEI DER AKTUALISIERUNG EINES BLICKVERHALTENS DARSTELLT. DIE ENTSCHEIDUNG, OB DER CHARAKTER DAS BLICKVERHALTEN NOCH AUSFÜHREN KANN, IST, UNTER ANDEREM, ABHÄNGIG DAVON, OB DIE MAXIMALE ZEIT ÜBERSCHRITTEN IST.

Das Ziel des Charakterblickes kann zwar momentan theoretisch abgefragt werden, allerdings können sich die Körperteile, die der Charakter benutzt, um in eine bestimmte Richtung zu sehen, noch nicht zu diesem Ziel rotieren. Um dies zu ermöglichen ist eine weitere Komponentenart nötig. Die Körperteile selbst werden dabei durch die in der Animation häufig genutzten Knochen des Charakterskeletts dargestellt. Eine eigene Komponente je Knochen verwaltet die momentane Rotation und die Berechnung der benötigten Rotation zum Ziel des Blickes. Die einzelnen Arten der Knochen werden, wie bei den Blickverhalten, von einer Basisklasse abgeleitet und überschreiben

ebenfalls die einzelnen Funktionen der Basisklasse, falls dies nötig ist. Damit sich die Knochen möglichst leicht zur Zielposition rotieren lassen, stellt die Basisklasse eine Funktion bereit, welcher nur die Zielposition und ein Boolean, der angibt, ob die maximalen Winkel aller vorherigen Knochen erreicht sind, übergeben werden muss.

Um das Erfüllen der neunten und zehnten Anforderung zu gewährleisten, werden sowohl maximale Winkel, um die sich die jeweiligen Knochen rotieren dürfen, mittels Variablen definiert als auch Funktionen bereitgestellt, um zu überprüfen, ob diese maximalen Winkel erreicht sind.

Die elfte Anforderung wird erfüllt, indem sich die tatsächliche Zielrichtung des Knochens, aus der Interpolation der momentanen Ausrichtung des Knochens und der gewünschten Zielrichtung, in Abhängigkeit einer maximalen Drehgeschwindigkeit, berechnet.

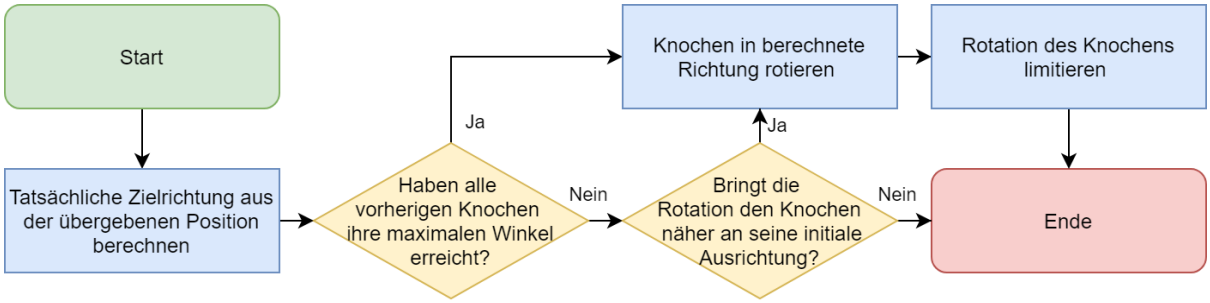

ABBILDUNG 4: EIN FLUSSDIAGRAMM DAS DEN GRUNDLEGENDEN ABLAUF BEI DER ROTATION DER MEISTEN KNOCHEN DARSTELLT.

Die letzte nötige Komponentenart für das Blicksystem stellt die Blicke des Charakters und des Spielers dar. Die Komponenten der beiden werden dabei von derselben Basisklasse abgeleitet. Während sich die Komponente des Spielers hauptsächlich darum kümmert einen fokussierten Punkt zu berechnen und zu verwalten, ist die Komponente des Charakters dafür zuständig, die Funktion zum Aktualisieren des momentanen Blickverhaltens zu rufen, gegebenenfalls das Blickverhalten zu wechseln, das Ziel des momentanen Blickverhaltens an die einzelnen Knochenkomponenten mitzuteilen und eine Funktion bereitzustellen, um die Wahrscheinlichkeit, dass der Charakter den Blick des Spielers sieht, zu ermitteln und somit das Erfüllen der zwölften Anforderung zu ermöglichen.

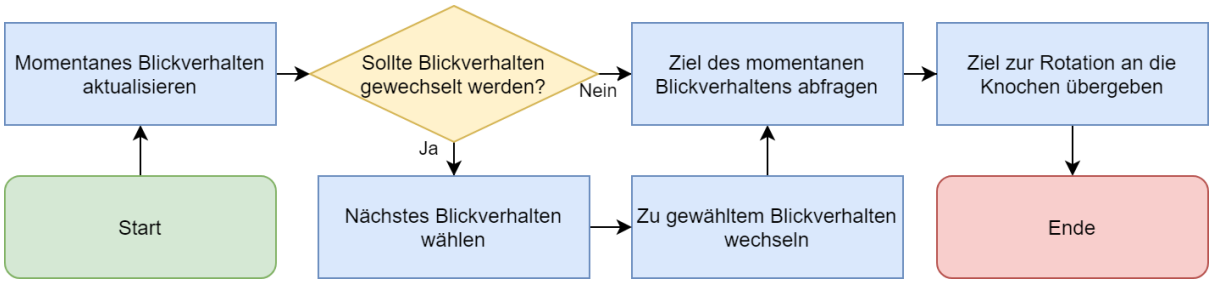

ABBILDUNG 5: EIN FLUSSDIAGRAMM DAS DEN ABLAUF EINES AKTUALISIERUNGSSCHRITTES DER BLICK-KOMPONENTE DES CHARAKTERS ZUR AKTUALISIERUNG DES MOMENTANEN BLICKVERHALTENS UND ROTATION DER KNOCHEN DARSTELLT.

4.3 ANFORDERUNGEN AN DIE ZWISCHENMENSCHLICHE DISTANZ

Für das System der zwischenmenschlichen Distanz ist es ebenfalls nötig einige Anforderungen zu definieren. In jedem Fall muss es zunächst einmal möglich sein, dass der Charakter versucht eine bestimmte Distanz zum Spieler einzuhalten, falls dieser zu nahe an ihn kommt oder sich zu weit von

ihm entfernt. Das Einnehmen dieser Distanz findet allerdings im echten Leben normalerweise nur statt, falls die betreffende Person auch wahrgenommen wird. Dazu kommt, dass die Näherung an eine andere Person, um diese Distanz einzunehmen, ebenfalls nur stattfindet, wenn mit dieser Person interagiert wird. Daraus bilden sich die folgenden beiden Anforderungen an die Komponente der zwischenmenschlichen Distanz:

1. Der Charakter versucht eine bestimmte Distanz zum Spieler einzunehmen, sollte dieser zu nahe an ihn kommen und wahrgenommen werden.
2. Der Charakter versucht eine bestimmte Distanz zum Spieler einzunehmen, sollte sich dieser bei einer Interaktion zu weit entfernt von ihm befinden.

Die tatsächliche Distanz, die dabei eingenommen wird, variiert, wie in Kapitel 2.2 erwähnt, auf Grund von verschiedenen Faktoren, wie der Persönlichkeit oder der Beziehung zu der beteiligten Person. Dazu kommt ebenfalls, dass ein Mensch nicht jedes Mal die exakt selbe Distanz zu einem anderen Menschen einnehmen kann. Aus diesen beiden Gründen kann die Distanz auch nicht mit einem einzigen festen Wert dargestellt werden:

3. Die Distanz, die der Charakter gegenüber dem Spieler einnimmt, ist variabel.

Existiert keine Position, an die sich der Charakter bewegen kann, um die Distanz einzuhalten, sollte und kann die Bewegung nicht gestartet werden. Im Falle, dass sich der Charakter bereits auf dem Weg zu einer solchen Position befindet, diese Position aber aus verschiedenen Gründen nicht mehr gültig ist, zum Beispiel weil sich der Spieler bewegt und daher die Zielposition nicht mehr in der richtigen Distanz liegt, sollte der Charakter entweder versuchen eine neue Position zu finden oder, falls dies nicht möglich ist, die Bewegung stoppen, um zu verhindern, dass er sich an eine nun ungültige Position bewegt:

4. Die Bewegung des Charakters, um die Distanz einzuhalten, startet nur wenn eine Position mit einer gültigen Distanz existiert.
5. Wird die Zielposition während der Bewegung ungültig, wird versucht eine neue Position zu finden oder die Bewegung des Charakters gestoppt, falls dies nicht möglich ist.

Kommt eine Person, nahe genug an eine andere Person, um diese direkt zu berühren, wird diese im Normalfall auf Grund der Berührung wahrgenommen. Die zweite Person entfernt sich daraufhin höchstwahrscheinlich von der ersten, um wieder etwas Distanz herzustellen:

6. Kommt der Spieler nahe genug an den Charakter, um diesen theoretisch direkt zu berühren, so nimmt der Charakter den Spieler wahr und versucht sich ebenfalls von diesem zu entfernen.

Im Vergleich zum Blicksystem werden für das System der zwischenmenschlichen Distanz nur ein halbes Dutzend an grundlegenden Anforderungen definiert. Dies liegt einerseits daran, dass sich dieses System mit einem etwas spezifischeren Thema als das Blicksystem beschäftigt und andererseits, dass die tatsächlichen Anforderungen ebenfalls sehr stark von den Faktoren die in der letztendlichen Implementation verwendet werden, um die eingehaltene Distanz zu beeinflussen, abhängig sind.

4.4 GRUNDLEGENDES DESIGN DER ZWISCHENMENSCHLICHEN DISTANZ

Das grundlegende Design des Systems der zwischenmenschlichen Distanz ist recht simpel gehalten. Es besteht dabei aus einer einzigen Komponente, die sich um das Prüfen der Distanz zum Spieler und der Berechnung der Zielposition der Bewegung des Charakters kümmert.

Um die ersten beiden Anforderungen zu erfüllen, prüft diese Komponente, solange sich der Charakter noch nicht auf dem Weg in die momentane Distanzzone befindet, ob die Distanz des Charakters

gegenüber dem Spieler, eine minimale beziehungsweise maximale Distanz überschreitet, der Charakter den Spieler überhaupt wahrnimmt und im Falle der Überschreitung der maximalen Distanz ebenfalls, ob der Charakter momentan mit dem Spieler interagiert. Je nachdem welcher dieser Fälle eintritt, wird dem Charakter mitgeteilt, dass er eine bestimmte Distanz einnehmen soll.

Die dritte Anforderung ist grundlegend erfüllt, indem die Distanzen, die der Charakter einnehmen kann, als Intervall mit einer unteren und oberen Grenze dargestellt werden und die letztendliche Distanz, die der Charakter einnimmt, ein zufälliger Wert aus diesem Intervall ist. Um allerdings die Distanzen auch in Abhängigkeit der Beziehung, die der Charakter zum Spieler haben soll, zu wählen, enthält die Komponente der zwischenmenschlichen Distanz ebenfalls eine Liste mit Komponenten die Distanzzonen darstellen. Diese Zonen haben jeweils ein eigenes Intervall für ihre Distanzbereiche.

Die vierte Anforderung ist leicht erfüllt, indem die Bewegung nur gestartet wird, wenn eine gültige Position in der momentanen Distanzzone des Charakters existiert. Gleichermaßen wird für die fünfte Anforderung geprüft, ob die Zielposition bei der Bewegung in die Distanzzone noch gültig ist. Ist dies nicht der Fall, wird versucht eine neue Position zu finden. Kann keine gültige Position gefunden werden, wird die Bewegung des Charakters gestoppt.

Zur Erfüllung der letzten Anforderung enthält die Komponente der zwischenmenschlichen Distanz eine Variable, die die horizontale Distanz darstellt, ab der der Spieler den Charakter berührt. Sollte sich der Spieler innerhalb dieser Distanz bewegen, wird dem Charakter ebenfalls mitgeteilt, dass er sich in die momentane Distanzzone bewegen sollte.

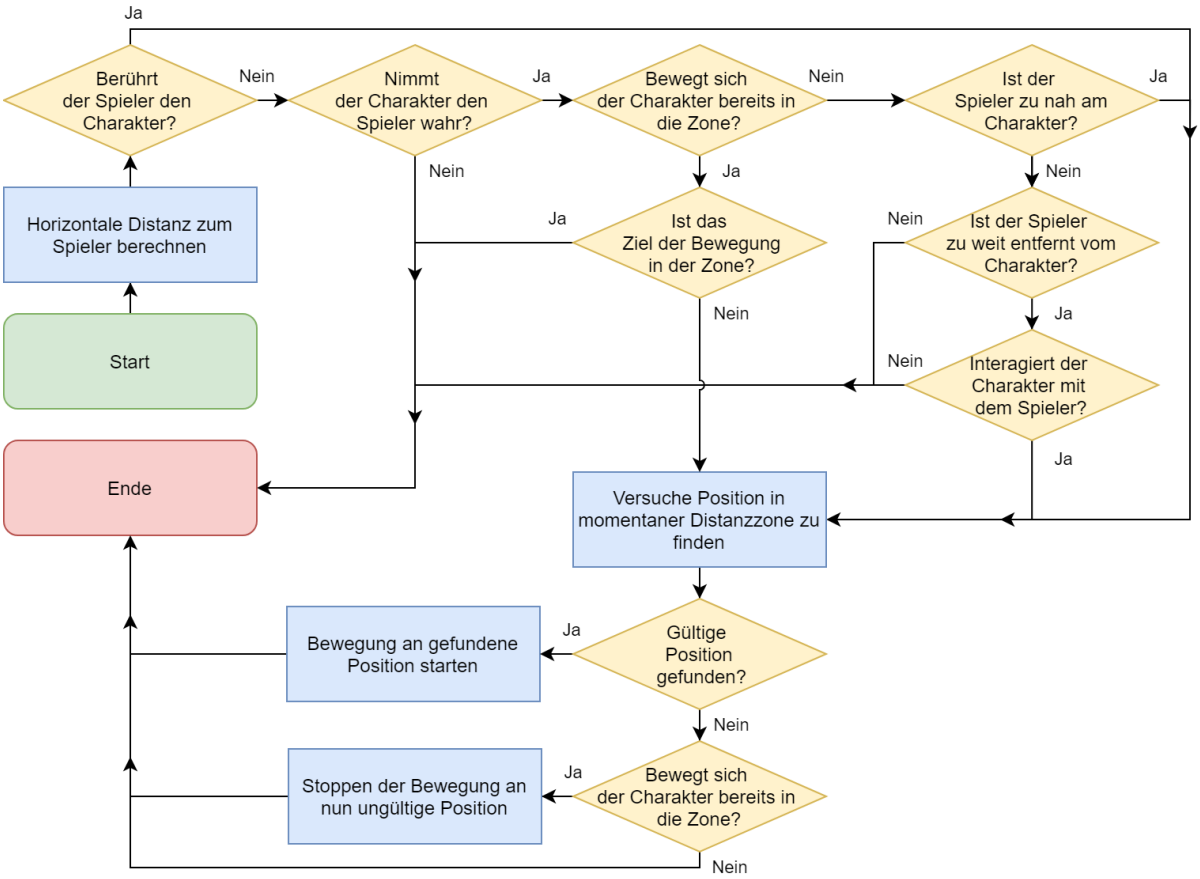

ABBILDUNG 6: EIN FLUSSDIAGRAMM DAS DEN ABLAUF EINES AKTUALISIERUNGSSCHRITTES DER KOMPONENTE ZUM EINHALTEN DER ZWISCHENMENSCHLICHEN DISTANZ DARSTELLT.

5 IMPLEMENTIERUNG DER ANWENDUNG

Nachdem nun die grundlegenden Anforderungen an das Blicksystem und das System der zwischenmenschlichen Distanz bekannt sind und geklärt ist wie das jeweilige grundlegende Design der beiden Systeme diese Anforderungen erfüllt, kann nun auf die letztendliche Implementation der Anwendung, in der die beiden Systeme genutzt werden, eingegangen werden.

5.1 CHARAKTERDESIGN

Die erstellte Anwendung benötigt natürlicherweise einen Charakter, der das Blicksystem und die Komponente zur Einhaltung der zwischenmenschlichen Distanz nutzen kann. Der komplette Prozess der Charaktererstellung (Modellierung, Texturierung, und Rigging) erfolgt in der kostenlosen 3D-Grafiksoftware Blender[9].

Das Design des Charakters ist relativ schlicht gehalten. Vom Aussehen her stellt er einen Jungen dar. Die Proportionen seiner einzelnen Körperteile zueinander orientieren sich zwar an der Realität, sind allerdings nicht hundertprozentig gleich zu denen eines echten Menschen. So sind vor allem die Augen um einiges größer als sie es in der Realität wären. Der Hauptgrund dafür ist, dass, wie bereits in Kapitel 1.2 erwähnt, Anabuki et al., in ihren Experimenten finden, dass Personen es bevorzugen, wenn sich der in ihrer Arbeit verwendete Charakter komplett in ihrem Sichtfeld befindet. (M. Anabuki et al., 2000, S. 2). Es wird davon ausgegangen, dass sich dies zu einem gewissen Grad auch auf den Charakter dieser Arbeit übertragen lässt. Nachdem allerdings das Sichtfeld des Displays der HoloLens, im Vergleich zu anderen MR- und AR-Brillen, relativ eingeschränkt ist, muss der Charakter entweder kleiner skaliert werden oder sich weiter entfernt vom Spieler aufhalten, damit möglichst viel von ihm auf einmal sichtbar ist. Beide Lösungswege sind im Kontext dieser Anwendung jedoch mit Problemen behaftet. Der Charakter selbst kann nicht zu klein skaliert werden, da der Spieler ansonsten durchgehend in einem eventuell unangenehmen Winkel nach unten sehen muss. Der Charakter kann sich aber auch nicht dauerhaft zu weit entfernt vom Spieler aufhalten, da er ansonsten die zwischenmenschliche Distanz nicht glaubhaft einhalten kann. Als Lösung ist ein Kompromiss zwischen beiden gewählt. Die Distanz, die der Charakter versucht einzuhalten, ist also leicht erhöht und er selbst etwas kleiner skaliert. Damit es für den Spieler allerdings trotzdem leicht ist zu sehen, wohin der Charakter momentan sieht, sind die Augen unproportional groß.

Die Texturierung ist ebenfalls recht simpel gehalten. So sind die verschiedenen Kleidungsstücke einfarbig und nur das Gesicht enthält etwas mehr Details. Auf Grund der Art und Weise wie die HoloLens die Szene über das Display ausgibt und dem Fehlen von Informationen bezüglich der echten Lichtquellen, kann es, je nach der Orientierung des Charakters, dazu kommen, dass verschiedene Teile des Gesichtes, wie der Nase und der Mund, nur schwer erkennbar sind. Unter diesem Problem leidet während der Entwicklung die erste Iteration des Charakters. Neben einem noch größeren Kopf und Augen hat dieser keinerlei Betonungen der Nase und des Mundes. Auf der HoloLens sind diese beiden Teile des Gesichtes in der ersten Version daher nahezu nicht sichtbar, was den Charakter eher unheimlich und merkwürdig aussehen lässt. Diese Probleme sind dementsprechend in der zweiten Iteration des Charakters adressiert. Diese neue Charakterversion verwendet eine Ambient-Occlusion-Map in Kombination mit einer leichten Betonung auf der Textur des Charakters, um diese Stellen auf der HoloLens leichter sichtbar zu machen.

Das für den Charakter verwendete Skelett orientiert sich ebenfalls an dem eines echten Menschen. Der Hüftknochen bildet dabei den sogenannten Rootknochen, von dem alle anderen Knochen ausgehen. Um die freie Rotation der Augen zu ermöglichen ist, je Auge, ebenfalls ein Knochen vorhanden, mit denen diese, wie die anderen Körperteile des Charakters, rotiert werden können.

[9] Blender - https://www.blender.org/

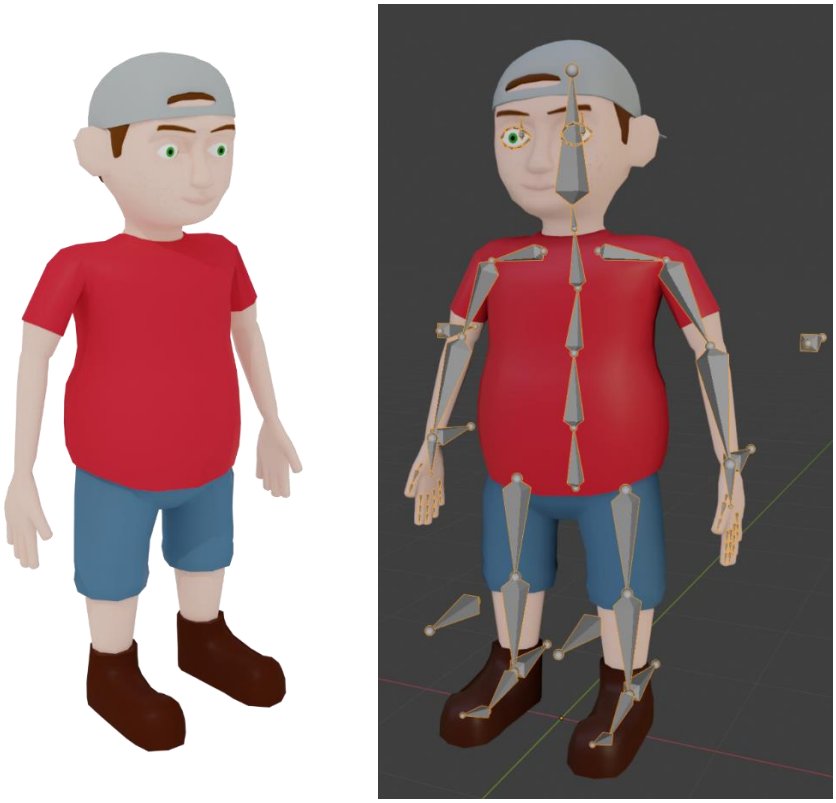

ABBILDUNG 7 UND 8: LINKS IST DAS MODELL DES CHARAKTERS INKLUSIVE TEXTURIERUNG UND RECHTS DER SKELETTAUFBAU DES CHARAKTERS ZU SEHEN.

5.2 ANIMATIONEN UND BEWEGUNG DES CHARAKTERS

Die Bewegung des Charakters erfolgt zum größten Teil durch Animationen. Die Erstellung dieser Animationen findet ebenfalls in Blender statt. Damit es allerdings nicht nötig ist Animationen für alle Knochen, die der Charakter benutzt, um in die Richtung des momentanen Blickzieles zu sehen, zu erstellen, findet die Ausrichtung der meisten dieser Knochen nicht über Animationen statt, sondern über Berechnungen in den verschiedenen Knochen-Komponenten.

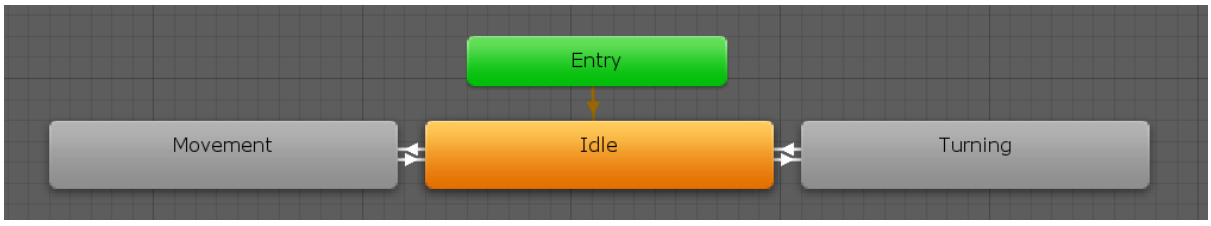

ABBILDUNG 9: DER ANIMATION-CONTROLLER DES CHARAKTERS. DER WECHSEL ZWISCHEN DEN ANIMATIONSZUSTÄNDEN ERFOLGT JEWEILS DURCH EINEN BOOLEAN.

Eine Reihe an Animationen die trotzdem für die unterschiedlichen Blickverhalten des Charakters benötigt werden, sind verschiedene Drehanimationen, damit sich der komplette Charakter drehen kann, sobald alle Knochen des Charakterskeletts, die sich in die Richtung des Blickzieles rotieren, die jeweiligen maximalen Rotationswinkel erreichen. Hierfür werden drei verschiedene Animationen verwendet. Die erste dieser Animationen ist eine Animation zum Rotieren des Charakters um 90° nach links. Die zweite Animation ist die erste Animation, allerdings gespiegelt, um die Drehung um 90° nach rechts auszuführen. In der letzten Drehanimation steht der Charakter lediglich still und führt somit eine Drehung um 0° aus. Um es zu ermöglichen, dass das Rotieren nicht nur um -90°, 0° und 90°

möglich ist, wird zwischen diesen drei Animationen interpoliert. Der Interpolationswert befindet sich dabei in einem Intervall zwischen -1,0 und 1,0, wobei -1,0 die Rotation um 90° nach links, 0,0 die Rotation um 0° und 1,0 die Rotation um 90° nach rechts darstellt.

Neben den Animationen für die Rotation auf der Stelle, muss sich der Charakter auch bewegen können, um nach der Berechnung einer Zielposition und des dorthin führenden Pfades auch die zwischenmenschliche Distanz einhalten zu können. Für die Bewegung werden hierbei fünf verschiedene Animationen verwendet. Diese Animationen bestehen sowohl aus Geh-Animationen für das Vorwärts- und Rückwärtsgehen und zwei Animationen, um seitlich nach links und rechts gehen zu können als auch einer fünften Animation bei der der Charakter, wie bei den Drehanimationen, stillsteht. Zwischen diesen fünf Animationen wird ebenfalls interpoliert. Zur Interpolation werden zwei Interpolationswerte für die seitliche Bewegung und Vorwärts-/Rückwärtsbewegung an die Animator-Komponente des Charakters übergeben.

Um die Position und Rotation des Charakters mit den Werten der Animationen zu synchronisieren, wird das von Unity bereitgestellte Root-Motion-System genutzt. Dadurch werden die Position und Rotation des Root-/Hüftknochens genutzt, um die Position und Rotation des Charakters zu bestimmen. Falls sich der Rootknochen in einer Animation bewegt oder rotiert, wird dies auch auf den Charakter angewendet.

Weder die Komponenten des Blicksystems noch die der zwischenmenschlichen Distanz kümmern sich direkt um das Abspielen der Animationen und somit die Bewegung des Charakters, stattdessen übergeben sie den Winkel, um den sich der Charakter rotieren sollte, beziehungsweise die Position, an die er sich bewegen sollte, an dessen Movement-Komponente.

In ersterem Fall berechnet diese Komponente den Interpolationswert für die Drehanimation, teilt diesen Wert dem Animator des Charakters mit und wechselt in den entsprechenden Animationszustand. Die Berechnung des Interpolationswertes selbst erfolgt lediglich durch die Teilung des übergebenen Winkels durch 90 (der maximale Winkel, um den sich auf einmal in der Animation rotiert werden kann). Der daraus entstehende Wert beschreibt dementsprechend wie viel von der Drehanimation nach links oder rechts verwendet werden sollte. Am Ende der Animation wird ein Event genutzt, um wieder aus diesem Animationszustand heraus zu wechseln. Die Rotation des Charakters auf der Stelle kann in jedem Frame erfolgen, solange er sich nicht bereits dreht oder bewegt. Sollte das Ausführen einer Drehanimation nicht ausreichen, kann dementsprechend die Rotation erneut stattfinden, um den Charakter um den restlichen Winkel zu rotieren.

Die Bewegung des Charakters an eine bestimmte Stelle auf dem Navmesh ist, im Vergleich zur Drehung, etwas aufwendiger und es ist daher notwendig einige Prüfungen vorzunehmen, um die Gültigkeit der Bewegung an die übergebene Position sicherzustellen. So ist die gewünschte Zielposition nur gültig, wenn ein vollständiger Pfad dorthin existiert. Existiert kein Pfad oder ein Pfad, der nur teilweise zur Zielposition führt, wird das Ziel der Bewegung auf die momentane Position des Charakters gesetzt, um die eventuelle Bewegung entlang eines zuvor gesetzten und nun ungültigen Pfades zu stoppen.

Bevor die Bewegung zu der gesetzten Zielposition und der dazugehörende Wechsel in den entsprechenden Animationszustand stattfinden kann, wird zusätzlich sichergestellt, dass der Charakter nicht bereits das Ziel der Bewegung erreicht hat. Das Ziel gilt dabei als erreicht, sobald momentan kein Pfad berechnet wird, die übrige Distanz entlang des Pfades kleiner als die eingestellte Stopdistanz ist und entweder kein Pfad gesetzt ist oder sich der Charakter laut seiner NavMeshAgent-Komponente nicht mehr bewegt.

Ist sowohl die übergebene Position als auch der dorthin führende Pfad gültig erfolgt die Berechnung der Interpolationswerte für die Bewegungsanimation in jedem Frame. Standardmäßig würde sich die NavMeshAgent-Komponente des Charakters ebenfalls um das Berechnen und Setzen dessen

Bewegung und Rotation entlang des Pfades kümmern. Diese beiden Funktionen werden daher deaktiviert. Da jedoch die Steuerungsrichtung dieser Komponente genutzt wird, um die Interpolationswerte zu berechnen, muss vor deren Berechnung die simulierte Position der NavMeshAgent-Komponente auf die Position des Charakters gesetzt werden. Die Richtung, in die sich der Charakter als nächstes bewegt, ist die normalisierte und in dessen lokales Koordinatensystem umgerechnete Steuerungsrichtung der NavMeshAgent-Komponente multipliziert mit dessen maximaler Bewegungsgeschwindigkeit. Die letztendlichen Interpolationswerte sind die X- und Z-Komponente dieses berechneten Richtungsvektors.

Das Stoppen der Bewegung und der entsprechende Wechsel des Animationszustandes erfolgt, indem, während sich der Charakter bewegt, in jedem Frame geprüft wird, ob der Charakter das Ziel erreicht hat. Entsprechend wird die Bewegung des Charakters zu diesem Ziel beendet.

Des Weiteren werden zwei Animationen genutzt, die zwar nicht zwingend nötig sind, allerdings viel zu der Präsenz des Charakters beitragen können. Zum einen ist dies eine Animation für den Lidschlag des Charakters, um starren zu vermeiden und zum anderen eine Idle-Animation, damit der Charakter, falls er gerade keine der anderen Animationen abspielt, nicht absolut stillsteht. Da es weder sehr natürlich wirken würde, wenn der Charakter immer nach der exakt selben Zeit blinzeln würde noch, wenn er jedes Mal die exakt selbe Idle-Animation ausführen würde, wird bei beiden Animationen etwas Varianz eingebracht. Bei der Lidschlag-Animation wird dazu am Ende jeder Wiederholung eine Zeit aus einem bestimmten Intervall gewählt, um die das erneute Abspielen der Animation verzögert wird. Das Gleiche hätte jedoch nicht den gewünschten Effekt bei der Idle-Animation, daher zeigt sich die Varianz dieser in der Form der Interpolation mit einer weiteren Stillsteh-Animation. Am Ende jedes Animationsdurchgangs wird dementsprechend ein neuer Interpolationswert für den nächsten Durchgang gewählt.

5.3 AUFBAU UND ABLAUF DER SZENE

Die Szene der Anwendung enthält neben den Objekten, die vom HoloToolkit bereitgestellt werden und das Nutzen der unterschiedlichen Funktionen der HoloLens ermöglichen, zusätzlich ein Game-Manager-Objekt, welches verschiedene Komponenten enthält, die sich um allgemeine Funktionen und den generellen Ablauf der Anwendung kümmern.

Das Vorbereiten der virtuellen Umgebung für das Platzieren des Charakters und Interagieren mit diesem erfolgt über mehrere Schritte. Als Erstes muss ein gewisser Teil der Umgebung gescannt werden, damit der Charakter genug Platz hat, um sich relativ frei bewegen zu können. Dazu wird nach dem Start der Anwendung direkt angefangen die Umgebung nach und nach zu scannen. Damit der Spieler erkennt, dass ein gewisser Teil der Umgebung bereits gescannt ist, erhält er visuelles Feedback über ein Wireframe-Material, dass sozusagen über die gescannten Gebiete gelegt wird. Um zu entscheiden, ob genug gescannt ist, wird die horizontale Fläche der gescannten Umgebung genutzt. Dies ist weitestgehend ausreichend, da sich der Charakter nur auf dem Boden bewegen kann und dementsprechend das Scannen von vertikalen Flächen nicht zwingend notwendig ist.

Sobald die minimale Fläche erreicht ist, kann der Scanprozess mittels eines Air-Taps abgeschlossen werden, woraufhin noch vorhandene Löcher in der gescannten Umgebung gefüllt werden. Auf diesem fertiggestellten Mesh der Umgebung wird daraufhin das Navmesh für die Bewegung des Charakters erstellt.

Sind all diese Schritte erledigt, kann der Spieler den Charakter über einen weiteren Air-Tap an der momentanen Position des Cursors platzieren. Nachdem der Charakter jedoch nur auf dem Boden platziert werden sollte, wird zusätzlich die Art der Fläche über die Funktionen des HoloToolkits geprüft. Zusammen mit dem Platzieren des Charakters wird ebenfalls das Wireframe-Material des

Umgebungsmeshes komplett schwarz eingefärbt, um stattdessen die reale Umgebung durchscheinen zu lassen aber dennoch virtuelle Objekte zu verdecken.

Damit der Spieler jederzeit weiß, welche Aufgabe er als Nächstes hat, erhält er bis nach der Platzierung des Charakters Anweisungen über einen im Raum schwebenden und der Blickrichtung der Kamera folgenden Text.

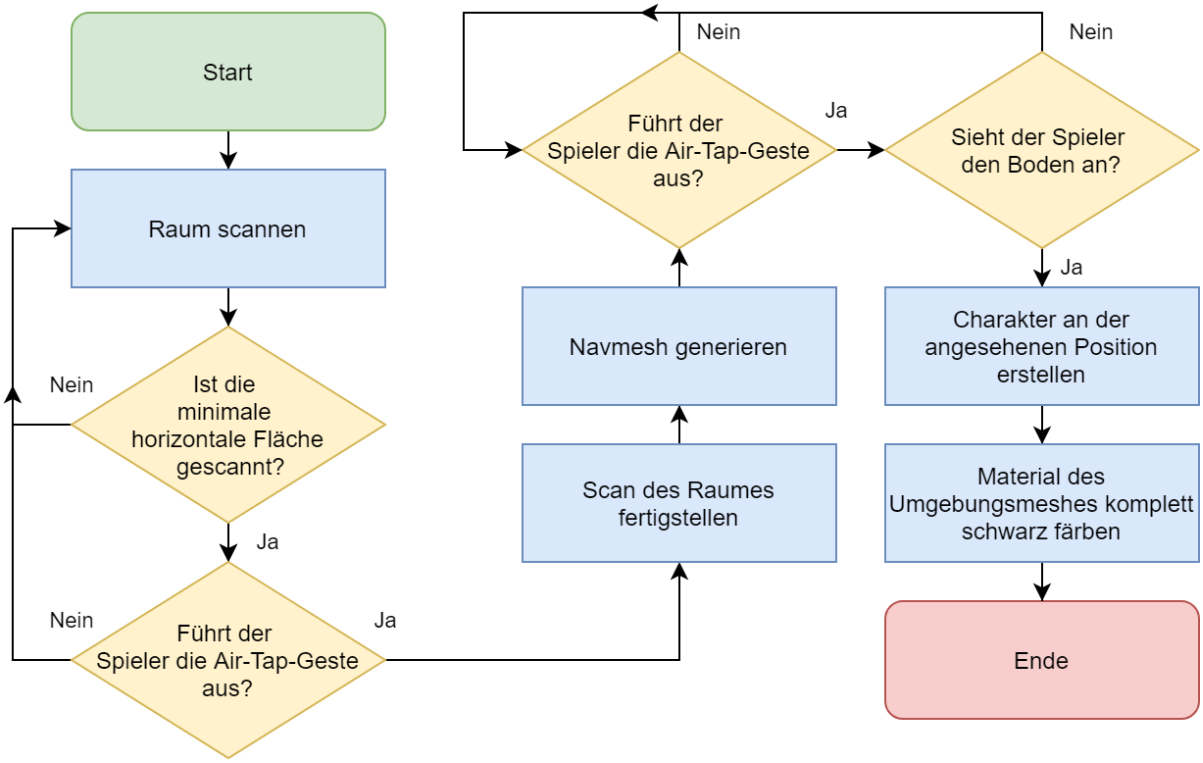

ABBILDUNG 10: EIN FLUSSDIAGRAMM DAS DEN GRUNDLEGENDEN ABLAUF DER ANWENDUNG DARSTELLT BIS DER CHARAKTER PLATZIERT IST.

Nachdem der Charakter platziert ist, beginnt direkt die Interaktion mit diesem. Neben der Änderung der Distanz zum Charakter, um ihn zur Änderung seiner eigenen Distanz zu bewegen und dem Blicken an unterschiedliche Stellen, um die Blickrichtung des Charakters zu ändern, ist die nonverbale Interaktion mit diesem ebenfalls über einige andere Wege möglich. Einer dieser Wege ist das Zeigen in eine Richtung, um die Aufmerksamkeit und den Blick des Charakters an einen bestimmten Punkt zu lenken. Dies geschieht in der Form des Sehens in eine Richtung und darauffolgend dem Ausführen und Halten der Air-Tap-Geste für eine kurze Zeit. Wird stattdessen der Charakter angesehen und die Air-Tap-Geste ausgeführt, wechselt der Charakter zum Ansehen des Spielers. Sieht der Spieler den Charakter nicht an während er die Air-Tap-Geste ausführt, „wirft“ er kleinere Objekte in den Raum auf die der Charakter reagieren kann, sobald sie mit einer Wand, dem Boden oder einem anderen Objekt kollidieren.

Damit der Charakter an eine neue Position bewegt werden kann und sichergestellt ist, dass die Anwendung nicht neugestartet werden muss, falls der Charakter aus verschiedenen Gründen in einer Ecke feststeckt oder sich allgemein nicht bewegen kann, ist es möglich dessen Position über das längere Halten der Air-Tap-Geste auf die momentan angesehene Position auf dem Boden zu setzen. Letzteres sollte im Normalfall nicht passieren, es kann aber je nach der Wahl der Umgebung und einer ungünstigen Generierung des Navmeshes durchaus auftreten.

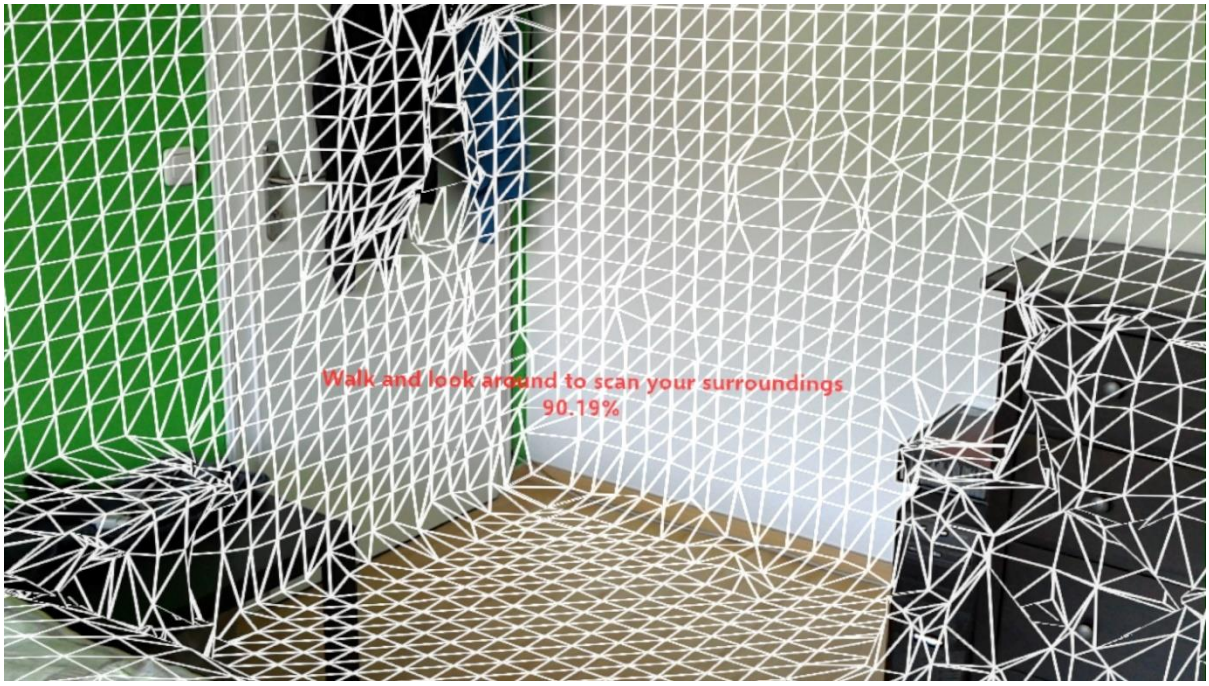

ABBILDUNG 11: DAS SCHRITTWEISE SCANNEN DER UMGEBUNG AUS DER SICHT DES SPIELERS.

5.4 BLICKVERHALTEN

Die verschiedenen Blickverhalten des Charakters werden in der Implementierung durch die Klassenart GazeBehaviour repräsentiert. Die als Basisklasse für alle anderen Blickverhalten dienende gleichnamige Klasse GazeBehaviour stellt dabei sowohl alle im grundlegenden Design definierten als auch weitere in der Implementation benötigten Variablen und Funktionen bereit, die von allen anderen Blickverhalten genutzt werden.

Die Limitation der möglichen Blickverhalten, in die von einem Blickverhalten gewechselt werden kann, erfolgt mit einer Liste bestehend aus Referenzen auf Blickverhalten, die ebenfalls am Charakter hängen. Die Verwendung einer Liste mit Referenzen anstatt anderer Wege, wie etwa einer List die lediglich die Typen der möglichen Blickverhalten enthält, bringt mehrere Vorteile mit sich. Der größte Vorteil ist, dass die Wechselwahrscheinlichkeit zu einer bestimmten Instanz eines Blickverhaltens direkt abgefragt werden kann, ohne dass die entsprechende Instanz großartig gesucht werden muss. Andere theoretische Vorteile inkludieren sowohl den möglichen einfachen Wechsel zu unterschiedlichen Instanzen desselben Blickverhaltens als auch den erneuten Wechsel zum selben Blickverhalten.

Damit die maximale Zeit, die der Charakter in einem Blickverhalten verbringen kann, nicht jedes Mal auf die gleiche Länge gesetzt wird, enthält die GazeBehaviour-Klasse nicht nur eine, sondern zwei Variablen, welche die maximale Zeit in der Form einer unteren und oberen Grenze eines Intervalls definieren. Sollte zu einem Blickverhalten gewechselt werden, wird dessen maximale Zeit auf einen zufälligen Wert aus diesem Intervall gesetzt.

Im momentanen Zustand könnte der Charakter das Blickverhalten wechseln und bei der nächsten Aktualisierung direkt wieder in das gleiche vorherige Verhalten zurückwechseln, was bei den meisten Blickverhalten eher unnatürlich wirken würde. Um dies zu unterbinden, enthält die Basisklasse zwei Variablen, welche eine Abklingzeit (im Folgenden „Cooldown“ genannt) definieren. Ähnlich der maximalen Zeit, die der Charakter in einem Blickverhalten verbringen kann, definiert dieser Cooldown die Zeit, die der Charakter nach einem Blickverhaltenswechsel nicht in dieses Verhalten zurückwechseln kann.

Die erste der in Kapitel 4.2 definierten Grundfunktionen, zum Liefern der Wechselwahrscheinlichkeit zu einem der nächsten Blickverhalten, wird durch die Funktion GetSwitchFromProbability dargestellt und die zweite, zum Liefern der Wechselwahrscheinlichkeit zu diesem Blickverhalten, durch die Funktion GetSwitchToProbability. Erstere berechnet die Wahrscheinlichkeit zum Wechseln, auf Grund der Wahrscheinlichkeiten zu den nächsten Blickverhalten zu wechseln und der Wahrscheinlichkeit auf Grund der momentanen Blickzeit zu wechseln, wie folgt:

$$P(Blickverhalten\ wechseln) = P(Blickzeit) \times \frac{1}{n}\sum_{i=1}^{n} P(mögliches\ Blickverhalten_i)$$

Wobei n die Anzahl der möglichen nächsten Blickverhalten darstellt, die Wahrscheinlichkeiten $P(mögliches\ Blickverhalten_i)$ von deren GetSwitchToProbability-Funktionen geliefert werden und sich die Wahrscheinlichkeit $P(Blickzeit)$ wie folgt zusammensetzt:

$$P(Blickzeit) = \left(\frac{momentane\ Blickzeit}{maximale\ Blickzeit}\right)^{p}$$

Der Parameter p ist hier eine Variable, die je Blickverhalten angegeben werden kann, um den Einfluss, den die Blickzeit auf die Wechselwahrscheinlichkeit hat, zu verändern. Damit die Wahrscheinlichkeit auf einen Wert zwischen 0,0 und 1,0 beschränkt bleibt und nicht potenziell exponentiell wächst, sobald die momentane größer als die maximale Blickzeit ist, wird das aus der Teilung der beiden Werte entstehende Ergebnis auf einen Wert zwischen 0,0 und 1,0 begrenzt. Welchen genauen Wert die GetSwitchToProbability-Funktion liefert, ist von den Implementationen der einzelnen Blickverhalten abhängig und wird dementsprechend auch individuell von diesen bestimmt.

Die dritte der Grundfunktionen wird durch die Funktion ShouldChangeBehaviour dargestellt. Diese Funktion führt keinerlei Berechnungen durch, sondern gibt lediglich einen zuvor gesetzten Boolean zurück. Das Bereitstellen dieser Funktion sorgt dafür, dass sich das Blickverhalten nicht selbst um den Wechsel zum nächsten Blickverhalten kümmern muss, sondern dies der Blick-Komponente des Charakters überlassen kann.

Das momentane Ziel des Blickes wird in der Form eines Vektors als Position im Raum darstellt. Ein Positionsvektor eignet sich dabei besser als eine Referenz auf ein Objekt der momentanen Szene, da dadurch auch beliebige Positionen im Raum als Ziel des Blickes verwendet werden können, auch wenn dort kein Objekt existiert. Die vierte und fünfte Grundfunktion werden jeweils durch die Funktionen UpdateGazeTarget und GetGazeTarget dargestellt. Während letztere den Blickziel repräsentierenden Positionsvektor zurückgibt, wird die Art und Weise wie das Ziel in UpdateGazeTarget aktualisiert wird von den abgeleiteten Blickverhalten entschieden.

Die letzte der Grundfunktionen, welche sich um das allgemeine Aktualisieren des Blickverhaltens und das Ausführen der verschiedenen Funktionen kümmert, ist die Funktion UpdateBehaviour. Bei einem Blickverhaltenswechsel müssen die Knochen des Charakters, in den meisten Fällen, erst einmal in die Richtung des Blickziels des neuen Blickverhaltens ausgerichtet werden. In dieser Wechselphase sollte das Aktualisieren der Zeit, die der Charakter in diesem Verhalten verbracht hat und das eventuelle Setzen des Booleans zum Verhaltenswechsel natürlich noch nicht stattfinden, sondern erst, sobald sich die Blickrichtung des Charakters nahe genug an der Position des Zieles befindet. Die Lösung dieses Problems kann über verschiedene Wege, wie zum Beispiel das Nutzen eines speziellen Wechselblickverhaltens, welches der Charakter zwischen Blickverhaltenswechseln einnimmt, gelöst werden. Die Implementation in dieser Arbeit nutzt jedoch lediglich einen Boolean, der in der UpdateBehaviour-Funktion gesetzt wird, sobald der Charakter das Ziel erreicht hat und in dessen Richtung sieht. Daraufhin beginnt sowohl das Aktualisieren der in diesem Blickverhalten verbrachten Zeit als auch die Berechnung und Entscheidung, ob das Blickverhalten geändert werden sollte. Die Entscheidung letzterem kann auf zwei Arten erfolgen. Kann sich der Charakter nicht mehr in diesem Blickverhalten befinden, zum Beispiel auf Grund des Überschreitens der maximalen Blickzeit, wird der

Boolean zum Wechsel des Blickverhaltens direkt gesetzt, ist dies jedoch nicht der Fall, wird der Boolean auf Grund der Wechselwahrscheinlichkeit von diesem Blickverhalten multipliziert mit der seit dem letzten Frame vergangenen Zeit gesetzt. Die Entscheidung, ob der Charakter ein bestimmtes Blickverhalten haben kann, ist grundsätzlich abgängig davon, ob die maximale Blickzeit erreicht ist oder ob das Blickverhalten momentan einen Cooldown hat.

```
public void UpdateBehaviour()
{
    //Aktualisierung des Blickziels
    UpdateGazeTarget();
    //Hat der Charakter das Ziel erreicht?
    if (m_CharacterArrivedAtGazeTarget)
    {
        m_CurrentGazeTime += Time.deltaTime;
        //Kann der Charakter das Blickverhalten nicht mehr haben?
        if (!CanHaveBehaviour())
        {
            m_ShouldChangeGazeBehaviour = true;
        }
        else
        {
            //Sollte das Blickverhalten auf Grund der
            //Wechselwahrhscheinlichkeit gewechselt werden?
            if (GetSwitchFromProbability() * Time.deltaTime > Random.value)
            {
                m_ShouldChangeGazeBehaviour = true;
            }
        }
    }
    else
    {
        //Sieht der Charakter das Ziel an?
        if (m_CharacterGaze && m_CharacterGaze.IsLookingAtPosition(m_GazeTarget))
        {
            m_CharacterArrivedAtGazeTarget = true;
        }
    }
}
```

ABBILDUNG 12: DIE IMPLEMENTATION DER UPDATEBEHAVIOUR-FUNKTION IN DER BASISKLASSE DER BLICKVERHALTEN, WOBEI DIE KOMMENTARE DIE GRUNDLEGENDEN PRÜFUNGEN BESCHREIBEN.

Zusätzlich existieren zwei weitere Funktionen, um das Setzen von Variablen beim Wechsel von und zu einem Blickverhalten zu erleichtern. Beiden kann das nächste beziehungsweise vorherige Blickverhalten als Parameter übergeben werden. Während sich die OnExitBehaviour-Funktion hauptsächlich um das Zurücksetzen der meisten Variablen und das Starten des Cooldowns kümmert, ist die OnEnterBehaviour-Funktion dafür zuständig, eventuell andere Variablen, die von dem vorherigen Verhalten übernommen oder erst beim Wechsel zu diesem Blickverhalten gesetzt werden sollten, zu setzen.

Neben den drei Blickverhalten, die zur grundlegenden Erfüllung der Ziele dieser Arbeit nötig sind, sind ebenfalls zwei weitere Blickverhalten implementiert, um die Möglichkeiten, über die nonverbal mit dem Charakter interagiert werden kann, zu erhöhen. Diese fünf Blickverhalten werden dabei durch die folgenden Klassen repräsentiert:

- LookAtPlayerGazeBehaviour
- LookAtFocusPointOfPlayerGazeBehaviour
- WanderGazeBehaviour
- LookAtPointGazeBehaviour
- LookAtEventGazeBehaviour

Um im Folgenden die Lesbarkeit zu erhöhen und das Wiederholen der im Grunde gleichbedeutenden Wörter GazeBehaviour und Blickverhalten zu vermeiden, werden die Blickverhalten bei ihrer Beschreibung mit LookAtPlayer, LookAtFocusPointOfPlayer, Wander, LookAtPoint und LookAtEvent abgekürzt.

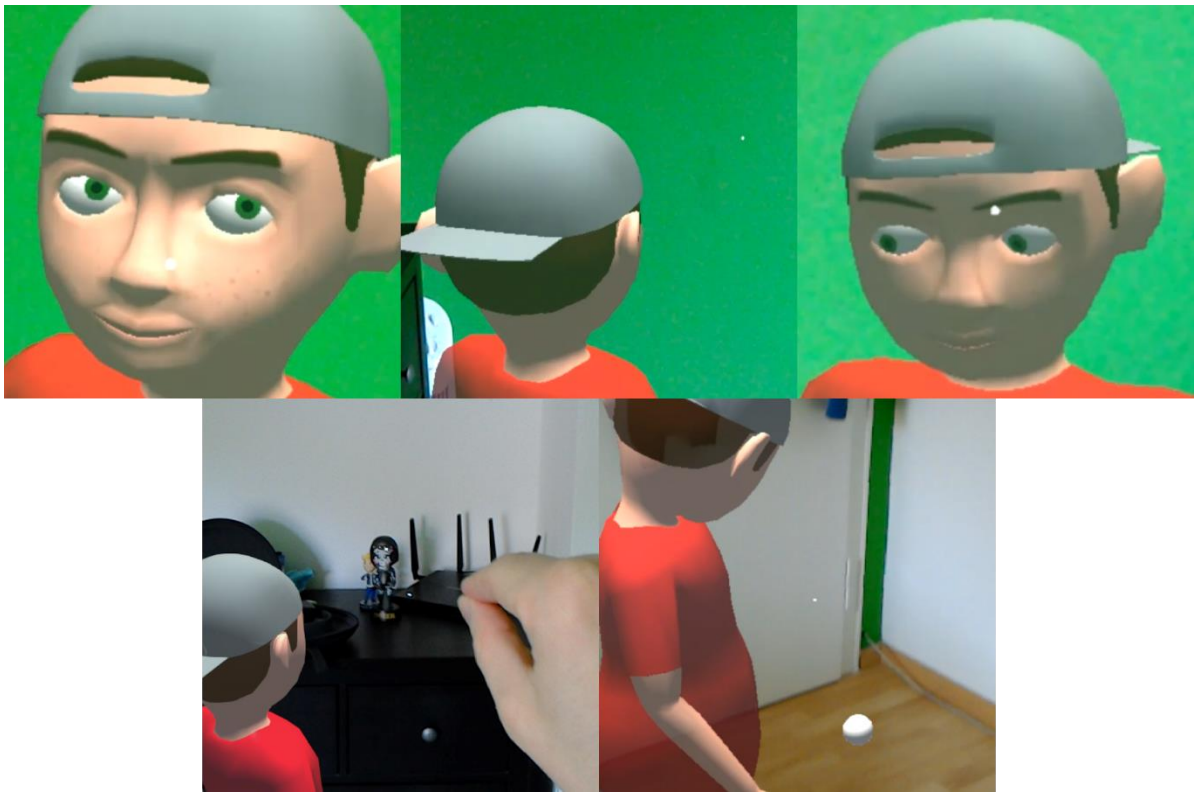

ABBILDUNG 13: DIE FÜNF BLICKVERHALTEN DES CHARAKTERS IN DER ANWENDUNG.

LookAtPlayer

Über das erste von der Basisklasse abgeleitete Blickverhalten LookAtPlayer sieht der Charakter den Spieler an. Die Klasse dieses Verhaltens überschreibt dabei die drei Funktionen UpdateGazeTarget, CanHaveBehaviour und GetSwitchToProbability. Die Liste der möglichen Blickverhalten besteht in diesem Blickverhalten aus Referenzen auf die beiden Blickverhalten LookAtFocusPointOfPlayer und Wander.

Um den Spieler anzusehen wird das Ziel dieses Blickverhaltens in UpdateGazeTarget auf die Position der Augen der Blick-Komponente des Spielers gesetzt. Im Normalfall ist dies die Position der Kamera, wodurch es, wie in Kapitel 3.1 erwähnt, so wirkt, als ob der Charakter direkt in die Augen des Spielers sehen würde.

In der überschriebenen CanHaveBehaviour-Funktion wird zusätzlich geprüft, ob die PlayerGaze-Komponente des Spielers existiert. Dies ist zwar nicht zwingend notwendig, aber dennoch von Vorteil, um sicherzustellen, dass bei den verschiedenen Berechnungen in diesem Blickverhalten keine Probleme auftreten.

GetSwitchToProbability liefert die korrekte Wechselwahrscheinlichkeit zu diesem Blickverhalten, indem als erstes geprüft wird, ob überhaupt in dieses Verhalten gewechselt werden kann. Ist dies nicht möglich, wird direkt die Wahrscheinlichkeit 0 % zurückgegeben. Kann theoretisch in das Verhalten gewechselt werden, beginnt die Berechnung dessen Wahrscheinlichkeit. Dazu wird die Wechselwahrscheinlichkeit zunächst auf eine bestimmte Grundwahrscheinlichkeit gesetzt. Anschließend wird geprüft, ob der Spieler den Charakter ansieht und dementsprechend über die Blick-Komponente des Charakters die Wahrscheinlichkeit berechnet, dass der Charakter den Blick des Spielers wahrnimmt beziehungsweise sieht. Ist diese Sehwahrscheinlichkeit größer als die zuvor gesetzte Grundwahrscheinlichkeit, wird die Wechselwahrscheinlichkeit auf diese Sehwahrscheinlichkeit gesetzt. Die Grundwahrscheinlichkeit erfüllt demnach zwei Aufgaben. Einerseits stellt sie ein unteres Limit für die Wahrscheinlichkeit dar, den Spieler anzusehen und andererseits, die grundlegende Wahrscheinlichkeit, den Spieler anzusehen, auch wenn dieser den Charakter momentan nicht ansieht. Damit die Equilibrium Theory ebenfalls in die Berechnung mit eingebracht wird, wird die bisherige Wechselwahrscheinlichkeit am Ende der Funktion ebenfalls mit der Wahrscheinlichkeit, den Spieler auf Grund der Distanz anzusehen, multipliziert. Diese Wahrscheinlichkeit wird dabei über die Komponente der zwischenmenschlichen Distanz des Charakters berechnet.

```
public override float GetSwitchToProbability()
{
    //Kann das Blickverhalten nicht eingenommen werden?
    if (!CanHaveBehaviour())
    {
        return 0.0f;
    }

    float probability = m_BaseProbability;
    //Sieht der Spieler den Charakter an?
    if (m_PlayerGaze.IsLookingAtObject(gameObject))
    {
        float seeingProbability = m_CharacterGaze.ProbabilityOfSeeingGaze(m_PlayerGaze);
        if (probability < seeingProbability)
        {
            probability = seeingProbability;
        }
    }

    float distanceToPlayer = m_InterpersonalDistance.GetHorizontalDistanceToTarget();
    float distanceProbability =
        m_InterpersonalDistance.GetMutualGazeProbabilityAtDistance(distanceToPlayer);
    probability *= distanceProbability;

    return probability;
}
```

ABBILDUNG 14: DIE GETSWITCHTOPROBABILITY-FUNKTION DES BLICKVERHALTENS LOOKATPLAYER.

LookAtFocusPointOfPlayer

Das nächste abgeleitete Blickverhalten ist LookAtFocusPointOfPlayer. In diesem Verhalten sieht der Charakter den fokussierten Punkt des Spielers an. Grundsätzlich ist es dazu nötig die beiden Funktionen CanHaveBehaviour und GetSwitchToProbability, aber zusätzlich auch die Funktion OnEnterBehaviour zu überschreiben. Der Wechsel ist in diesem Blickverhalten grundsätzlich zu den Verhalten LookAtPlayer und Wander möglich.

Das Überschreiben der UpdateGazeTarget-Funktion ist hier nicht nötig, da sich der fokussierte Punkt des Spielers jederzeit ändern kann, der Charakter aber nur die Position des Punktes, der zur Zeit des Blickverhaltenswechsel fokussiert ist, ansehen sollte. Das führt dazu, dass die Zielposition nur einmalig beim Wechsel zu diesem Verhalten gesetzt werden muss. Dies geschieht dementsprechend in OnEnterBehaviour.

Wie beim Blickverhalten LookAtPlayer wird CanHaveBehaviour auch hier überschrieben, um sicherzustellen, dass die Blick-Komponente des Spielers existiert, aber auch zusätzlich, um zu überprüfen, ob der Spieler momentan überhaupt einen gültigen fokussierten Punkt hat.

Die tatsächliche Entscheidung der Wechselwahrscheinlichkeit zu diesem Blickverhalten findet hierbei ebenfalls erst statt, sobald der Charakter das Blickverhalten einnehmen kann, anderenfalls wird entsprechend direkt 0 % zurückgegeben. Die Wahl der Wechselwahrscheinlichkeit fällt dabei auf eine von zwei eingestellten Wahrscheinlichkeiten, je nachdem, ob der Spieler den Punkt erst seit Kurzem fokussiert oder nicht.

```
public override float GetSwitchToProbability()
{
    //Kann das Blickverhalten nicht eingenommen werden?
    if (!CanHaveBehaviour())
    {
        return 0.0f;
    }

    float probability = 0.0f;

    FocusPoint focusPoint = m_PlayerGaze.GetFocusPoint();
    if (focusPoint)
    {
        //Ist der fokussierte Punkt vor Kurzem erstellt worden?
        if (focusPoint.IsRecent())
        {
            probability = m_FocusPointIsRecentProbability;
        }
        else
        {
            probability = m_FocusPointIsNotRecentProbability;
        }
    }

    return probability;
}
```

ABBILDUNG 15: DIE GETSWITCHTOPROBABILITY-FUNKTION DES BLICKVERHALTENS LOOKATFOCUSPOINTOFPLAYER.

Wander

Das Standardblickverhalten eines virtuellen Charakters kann verschiedene Formen einnehmen. In dieser Anwendung wird es durch das Blickverhalten Wander repräsentiert. Der Blick des Charakters wandert dabei in der Umgebung. Dazu werden die vier Funktionen UpdateGazeTarget, GetSwitchToProbability, OnEnterBehaviour und OnExitBehaviour der Basisklasse überschrieben. Der Wechsel ist standardmäßig nur zum Blickverhalten LookAtPlayer möglich, da der Spieler, wie in Kapitel 4.1 beschrieben, zuerst angesehen werden muss, bevor der Charakter beurteilen kann welchen Punkt er fokussiert.

Das Aktualisieren der Zielposition erfolgt in diesem Blickverhalten allerdings nicht über das Setzen auf die Position eines anderen Objektes der Szene, sondern über die Berechnung eines Punktes in Raum. Die Berechnung selbst erfolgt, indem zunächst die nach rechts und oben zeigenden Richtungsvektoren des Knochens, mit dem der Charakter sieht (normalerweise die der Augen), mit einer Gleitkommazahl für die jeweilige Stärke der Richtung und einer weiteren für die maximale Geschwindigkeit in diesem Aktualisierungsschritt multipliziert und anschließend auf die momentane Zielposition addiert werden. Die Werte für die Stärken der Richtungen werden, in jedem Aktualisierungsschritt, jeweils durch einen zufälligen Wert aus einem Intervall ergänzt und auf einen Wert zwischen -1,0 und 1,0 begrenzt. Die Grenzen der Intervalle, aus dem diese zufälligen Werte gewählt werden, können dabei durch die Angabe maximaler Änderungswerte beeinflusst werden. Der Wert für die maximale Geschwindigkeit besteht aus einer Variablen für die maximale Bewegungsgeschwindigkeit des Zieles multipliziert mit der seit dem letzten Frame vergangenen Zeit.

Damit der Einfluss der Addition der beiden entstehenden Richtungsvektoren auf die letzte Zielposition möglichst konstant bleibt und sich nicht mit erhöhter Entfernung zum Charakter verringert, wird der Richtungsvektor von den Augen des Charakters zu der neu berechneten Zielposition normiert und auf die Position der Augen addiert. Daraus berechnet sich eine Zielposition, welche immer den gleichen Abstand zu den Augen des Charakters hat.

Ein weiteres Problem würde allerdings momentan noch dadurch entstehen, dass sich die Zielposition des Blickes unbegrenzt nach oben und unten bewegen kann. Anders als bei der Bewegung nach links und rechts, kann es dadurch dazu kommen, dass der Charakter relativ weit und unnatürlich lange nach oben beziehungsweise unten sieht. Um dies möglichst leicht zu verhindern, werden, nach dem Berechnen und Setzen der neuen Zielposition, die Vektoren von der Position des Charakters zu der Zielposition und der Position dessen Augen berechnet und durch mehrere Ebenenprojektionen, auf die nach oben zeigende Achse des Charakters projiziert. Mit dem Längenunterschied zwischen den resultierenden Vektoren kann danach entschieden werden, ob sich die Zielposition über oder unter einer bestimmten oberen oder unteren Grenze befindet. Liegt die Zielposition nicht innerhalb dieser beiden Grenzen, wird die Gleitkommazahl für die Stärke der vertikalen Richtung um den maximalen Änderungswert erhöht oder verringert und wieder zwischen -1,0 und 1,0 begrenzt, um im nächsten Aktualisierungsschritt dafür zu sorgen, dass sich die Zielposition langsam wieder zwischen die beiden Grenzen bewegt.

ABBILDUNG 16: DIE VISUALISIERTE PROJEKTION DER ZIELPOSITION AUF DIE UP-ACHSE DES CHARAKTERS. HIERBEI STELLT DER WEIßE PUNKT DIE PROJIZIERTE POSITION DER AUGEN DAR UND DIE ROTEN PUNKTE DIE OBERE UND UNTERE GRENZE. DIE BEIDEN GELBEN PUNKTE STELLEN DIE POSITION DES ZIELES DAR, WOBEI DER RECHTE DIE TATSÄCHLICHE UND DER LINKE DIE PROJIZIERTE IST.

Damit beim nächsten Wechsel zu diesem Blickverhalten die Gleitkommazahlen für die Stärken der beiden Richtungen nicht dieselben Werte wie zuvor haben werden sie in der überschriebenen OnExitBehaviour-Funktion zusätzlich wieder auf den Wert 0,0 gesetzt.

Beim Wechsel zu diesem Blickverhalten würde es momentan noch der Fall sein, dass das Ziel des Blickes auf die letzte Zielposition dieses Verhaltens gesetzt wird. Bei den meisten Übergängen würde das dazu führen, dass die Augen schlaghaft in eine bestimmte Richtung springen und im schlechtesten Fall, dass der Charakter sich in eine komplett andere Richtung dreht. Um dies zu vermeiden wird das Ziel in der OnEnterBehaviour-Funktion auf die Zielposition des vorherigen Blickverhaltens gesetzt und von dieser Position angefangen den Blick des Charakters wandern zu lassen.

Die Berechnung der Wechselwahrscheinlichkeit zu diesem Blickverhalten, setzt sich aus den einzelnen Wechselwahrscheinlichkeiten der anderen möglichen Blickverhalten, zu denen vom momentanen gewechselt werden kann, grundlegend wie folgt zusammen:

$$P(Wander) = \frac{1}{n}\sum_{i=1}^{n} 1 - P(\text{mögliches } Blickverhalten_i)$$

n ist hierbei die Anzahl an möglichen nächsten Blickverhalten, in die von dem momentanen gewechselt werden kann, ohne die Wander-Blickverhalten. Gleicherweise sind die Wahrscheinlichkeiten $P(\text{mögliches } Blickverhalten_i)$ die Wechselwahrscheinlichkeiten zu den möglichen nächsten Blickverhalten ohne die des Typen Wander. Sollte der Wechsel nur zu Wander-Blickverhalten möglich sein, liegt die Wechselwahrscheinlichkeit bei 100 %.

```
public override float GetSwitchToProbability()
{
    //Kann das Blickverhalten nicht eingenommen werden?
    if (!CanHaveBehaviour())
    {
        return 0.0f;
    }

    float probability = 0.0f;
    List<GazeBehaviour> possibleGazeBehaviours =
        m_CharacterGaze.GetCurrentGazeBehaviour().GetPossibleNextGazeBehaviours();
    int behaviourCount = possibleGazeBehaviours.Count;
    for (int i=0; i < possibleGazeBehaviours.Count; i++)
    {
        //Ist das Blickverhalten kein Wander-Blickverhalten?
        if(possibleGazeBehaviours[i].GetType() != GetType())
        {
            probability += 1.0f - possibleGazeBehaviours[i].GetSwitchToProbability();
        }
        else
        {
            behaviourCount--;
        }
    }
    //Ist der Wechsel nicht nur zu Wander-Blickverhalten möglich?
    if(behaviourCount > 0)
    {
        probability /= behaviourCount;
    }
    else
    {
        probability = 1.0f;
    }

    return probability;
}
```

ABBILDUNG 17: DIE GETSWITCHTOPROBABILITY-FUNKTION DES BLICKVERHALTENS WANDER.

LookAtPoint

Das Blickverhalten LookAtPoint lenkt den Blick des Charakters an einen bestimmten Punkt im Raum. Im Vergleich zu den vorherigen drei Blickverhalten, findet der Wechsel nicht auf Grund einer bestimmten Wahrscheinlichkeit statt, sondern manuell, sobald der Spieler über das Halten der Air-Tap-Geste für eine kurze Zeit in eine bestimmte Richtung zeigt. Die Zielposition, die dabei gesetzt wird, ist entweder der über einen Raycast in die Blickrichtung des Spielers getroffene Punkt eines Objektes oder der Punkt am Ende dieses Raycasts. Damit aus diesem Blickverhalten auch wieder gewechselt werden kann, besteht die Liste der Blickverhalten, in die gewechselt werden kann, aus den beiden Blickverhalten LookAtPlayer und Wander.

LookAtEvent

Über das Blickverhalten LookAtEvent kann der Charakter auf unterschiedliche Ereignisse in der Umgebung reagieren. Wie bei LookAtPoint, findet der Wechsel zu diesem Blickverhalten ebenfalls nicht wie bei den ersten drei Blickverhalten statt, stattdessen wird darauf gewartet, dass verschiedene Ereignisse in der Umgebung ausgelöst werden. Die Liste der möglichen nächsten Blickverhalten besteht auch bei diesem Blickverhalten sowohl aus LookAtPlayer als auch Wander.

Um die unterschiedlichen Ereignisse in der Umgebung zu definieren, werden jedem Objekt, dass solche Ereignisse auslösen sollte, entsprechende LookAtEvent-Komponenten angehangen. Die Basisklasse dieser Komponentenart enthält, unter anderem, Funktionen, um die Position des Events,

also die Position, die der Charakter ansehen sollte, sobald er dieses Event ansieht und die Wahrscheinlichkeit dieses Event überhaupt anzusehen zu liefern. Jedes von dieser Basisklasse abgeleitete LookAtEvent überschreibt folglich diese beiden Funktionen, um die richtigen Werte zu liefern. Des Weiteren enthält die Basisklasse eine Funktion, die ein Event auslöst, über das das Blickverhalten eine Referenz auf das auslösende LookAtEvent bekommt und dementsprechend entscheiden kann, ob dieses angesehen werden sollte. So löst zum Beispiel die HitSomethingLookAtEvent-Komponente das Event aus, sobald das Objekt, an dem es hängt, ein anderes Objekt mit einer Geschwindigkeit trifft, die hoch genug ist, liefert die Position des Objektes, an dem es hängt und gibt als Wahrscheinlichkeit, das Event anzusehen, entweder die Geschwindigkeit mit dem das Objekt getroffen wird, begrenzt auf maximal 1,0 oder eine eingestellte Grundwahrscheinlichkeit zurück.

Damit das Blickverhalten aber auch wissen kann, wann ein solches LookAtEvent ausgelöst wird, abonniert dieses Blickverhalten ein Event des Game-Managers, das ausgelöst wird, sobald ein neues LookAtEvent registriert wird. Jede LookAtEvent-Komponente registriert sich dementsprechend bei ihrer Erstellung im Game-Manager. Der Game-Manager löst daraufhin das Event aus. Das Blickverhalten erhält eine Referenz auf das neue LookAtEvent und kann demensprechend dessen Event abonnieren.

Wird das Event eines LookAtEvents ausgelöst, berechnet sich die Wahrscheinlichkeit dieses anzusehen aus der Multiplikation der Ansehwahrscheinlichkeit des Events und einer bestimmten Grundwahrscheinlichkeit des Charakters, dass Event zu bemerken.

In der UpdateGazeTarget-Funktion wird die Zielposition dieses Blickverhaltens auf die vom momentan angesehenen LookAtEvent gelieferte Position gesetzt.

5.5 ROTATION DER KNOCHEN

Die Rotation der verschiedenen Knochen des Charakters in die Richtung des momentanen Blickzieles wird von der Komponentenart GazeBone übernommen. Dabei wird jedem Knochen des Charakters, der sich in die Zielrichtung rotieren sollte, eine dieser Komponenten angehangen. Im Falle des Charakters in dieser Implementation sind dies die Augen, der Kopf, der Hals, die drei Knochen des Rückgrats und der Root-/Hüftknochen. Als Grundlage für die verschiedenen Knochenarten dient dabei die ebenfalls gleichnamige Klasse GazeBone. Diese definiert eine Reihe an Funktionen, die in allen Knochenarten nötig sind. Dazu zählen vor allem die, in Kapitel 4.2 erwähnten, grundlegenden Funktionen LookAtGazeTarget, mit der der Knochen in Richtung einer Zielposition rotiert werden kann und ReachedMaxAngle, welche einen entsprechenden Boolean zurückgibt, je nachdem, ob einer der maximalen Winkel erreicht ist. Um zusätzlich die Position und Ausrichtung des Knochens, unabhängig von dessen Art, zu erhalten, werden vier weitere Funktionen definiert, die die drei Richtungsvektoren und die Position des Knochens zurückgeben.

GenericGazeBone

Da die meisten Knochen des Charakters nur die Aufgabe haben, sich in eine bestimmte Richtung orientieren zu können, ohne dabei allzu spezielle Abhängigkeiten oder Funktionen zu haben, kann die Klasse GenericGazeBone verwendet werden, um die meisten Knochen des Charakterskeletts zu rotieren.

Damit die Rotation des Knochens limitiert werden kann, enthält diese Klasse eine Referenz auf ein Objekt, welches die Grundrotation und -position des Knochens speichert. Dieses Grundtransformationsobjekt wird automatisch in Unitys Start-Funktion erstellt, übernimmt sowohl die initialen Positions- als auch die Rotationswerte des Knochens und wird anschließend an das nächste übergeordnete Knochenobjekt gehangen. Es repräsentiert in anderen Worten die normale

Ausrichtung des Knochens und wird genutzt, um zu bestimmen, wann die maximalen Rotationswinkel erreicht sind.

Nachdem Unity die Rotation des Knochens in jedem Frame mit der in der Animation definierten Rotation überschreibt, muss der Knochen zuerst mit der letzten Blickrotation aktualisiert werden, bevor er in die Zielrichtung rotiert werden kann. Dies geschieht über eine Funktion in der die lokale Rotation durch die Animation gespeichert und anschließend die momentane Knochenrotation um die letzte zusätzliche Blickrotation ergänzt wird.

Das Rotieren des Knochens, in der Funktion LookAtGazeTarget, erfolgt, indem zunächst die tatsächliche Richtung zu der übergebenen Zielposition in diesem Frame berechnet wird. Diese Richtung ergibt sich aus der Kombination mehrere Schritte. Als Erstes wird ein lokaler Offsetvektor auf die übergebene Zielposition addiert. Über diesen Offsetvektor kann die Zielposition bestimmter Knochen verschoben werden. Dies wird zum Beispiel beim Kopfknochen des Charakters genutzt, um diesen immer etwas unterhalb des übergebenen Zieles blicken zu lassen. Als Zweites wird die Richtung von der Position des Knochens zu dieser verschobenen Zielposition berechnet und geprüft, ob der Winkel zwischen der momentanen Ausrichtung des Knochens und der neuen Zielrichtung um die lokale x-Achse des Knochens größer als 90° ist. Ist dies der Fall, wird nur die horizontale Komponente des Zielrichtungsvektors genutzt. Dies ist nötig, da der Knochen stets versucht sich in Richtung des Zieles zu rotieren. Bei manchen Zielpositionen (zum Beispiel einer Position in der umgekehrten Blickrichtung zu der des Knochens) kann es deswegen dazu kommen, dass sich der Knochen vertikal nach oben oder unten rotiert, nur um sich, nachdem er sich horizontal weit genug rotiert hat, wieder zurück zu rotieren. Als Letztes berechnet sich der letztendliche Richtungsvektor, aus der Rotation des momentanen Vorwärtsvektors des Knochens zu der Richtung des aus den vorherigen Schritten entstandenen Zielrichtungsvektors in Abhängigkeit der maximalen Drehgeschwindigkeit des Knochens in diesem Frame. Die Rotation des Knochens in die berechnete Richtung erfolgt danach nur, wenn einer von zwei Fällen einritt. Entweder wenn alle vorherigen Knochen ihre maximalen Winkel erreicht haben oder wenn die Rotation, in die berechnete Richtung, den Knochen näher an seine initiale Ausrichtung bringt.

```
public Vector3 GetOffsetTargetDirection(Vector3 targetLocation)
{
    //Offset im lokalen Raum des Knochens
    Vector3 offset =
          transform.right * m_TargetOffset.x  +
          transform.up * m_TargetOffset.y  +
          transform.forward * m_TargetOffset.z;
    return targetLocation + offset - transform.position;
}

2 references
public Vector3 GetTargetDirectionThisFrame(Vector3 targetLocation)
{
    Vector3 possibleDirection = GetOffsetTargetDirection(targetLocation);
    //Wäre die Rotation zur Zielrichung um die X-Achse des Knochens größer als 90 Grad?
    if(Mathf.Abs(Vector3.SignedAngle(transform.forward, possibleDirection, transform.right)) > 90.0f)
    {
        float angleYAxis = Vector3.SignedAngle(transform.forward, possibleDirection, transform.up);
        possibleDirection = Quaternion.AngleAxis(angleYAxis, transform.up) * transform.forward;
    }

    float maxRadiansDelta = m_MaxTurnSpeed * Mathf.Deg2Rad * Time.deltaTime;
    return Vector3.RotateTowards(transform.forward, possibleDirection, maxRadiansDelta, 0.0f);
}
```

ABBILDUNG 18: DIE BEIDEN FUNKTIONEN ZUR BERECHNUNG DER ZIELRICHTUNG IN DER GENERICGAZEBONE-KOMPONENTE. DIE OBERE FUNKTION FÜGT DEN OFFSETVEKTOR ZUR ZIELPOSITION HINZU UND DIE UNTERE BERECHNET DARAUS DIE LETZTENDLICHE RICHTUNG, IN DIE DER KNOCHEN IN DIESEM FRAME AUSGERICHTET WERDEN KANN.

Die Limitierung auf die eingestellten maximalen Rotationswinkel erfolgt nach der Rotation des Knochens. Dazu werden die minimalen und maximalen Rotationswinkel der jeweiligen Rotationsachsen mit den Winkeln der initialen Rotation des Knochens verrechnet und mit den neuen Winkeln nach der Rotation verglichen. Wird einer dieser Winkel unter- oder überschritten wird sowohl der Rotationswinkel um die entsprechende Achse auf den jeweiligen minimalen oder maximalen Winkel gesetzt als auch der dazugehörige Boolean, um anzuzeigen, dass der maximale Winkel um diese Achse erreicht ist, gesetzt. Am Ende wird die lokale Rotation des Knochens auf die eventuell limitierte Rotation gesetzt und die zusätzliche Blickrotation für das Aktualisieren des Knochens im nächsten Frame berechnet und gespeichert.

```
protected void LimitEulerRotation()
{
    Vector3 euler = transform.localEulerAngles;
    Vector3 eulerBase = m_BaseTransformObject.transform.localEulerAngles;

    //Limitierung der Rotationen auf Werte zwischen -180.0 und 180.0
    if (euler.x > 180.0f) { euler.x -= 360.0f; }
    if (euler.y > 180.0f) { euler.y -= 360.0f; }
    if (euler.z > 180.0f) { euler.z -= 360.0f; }
    if (eulerBase.x > 180.0f) { eulerBase.x -= 360.0f; }
    if (eulerBase.y > 180.0f) { eulerBase.y -= 360.0f; }
    if (eulerBase.z > 180.0f) { eulerBase.z -= 360.0f; }
    //Berechnung der minimalen und maximalen Winkel
    Vector3 min = new Vector3(eulerBase.x + m_MinAngles.x, eulerBase.y + m_MinAngles.y, eulerBase.z + m_MinAngles.z);
    Vector3 max = new Vector3(eulerBase.x + m_MaxAngles.x, eulerBase.y + m_MaxAngles.y, eulerBase.z + m_MaxAngles.z);
    //Limitation der Rotation entlang der X-Achse
    m_ReachedMaxXAxisAngle = false;
    if (euler.x < min.x)
    {
        euler.x = min.x;
        m_ReachedMaxXAxisAngle = true;
    }
    else if (euler.x > max.x)
    {
        euler.x = max.x;
        m_ReachedMaxXAxisAngle = true;
    }

    //Limitation der Rotation entlang der Y- und Z-Achse
    //...

    transform.localEulerAngles = euler;
}
```

ABBILDUNG 19: DIE LIMITIERUNG DER ROTATIONSWINKEL NACH DER ROTATION DES KNOCHENS. DIE LIMITATIONEN DER WINKEL UM DIE Y- UND Z-ACHSE FINDEN GLEICH ZU DENEN DER X-ACHSE STATT, SIND HIER ALLERDINGS DER ÜBERSICHTLICHKEIT HALBER WEGGELASSEN.

EyesGazeBone

Zwei der drei Knochen, die nicht direkt über die GenericGazeBone-Komponente rotiert werden können, sind die beiden Augen des Charakters. Der Grund dafür ist, dass die Augen des Charakters in vielen Fällen abhängig voneinander sind. Stattdessen wird die EyesGazeBone-Komponente genutzt, um die Augen zu rotieren. Die Abläufe und Berechnungen in dieser Komponente finden dabei weitestgehend gleich zu denen der GenericGazeBone-Komponente statt, jedoch einmal je Auge.

Einerseits enthält diese Komponente dementsprechend einige Variablen doppelt, um das zweite Auge zu repräsentieren, andererseits werden sowohl die maximalen Winkel als auch die maximale Rotationsgeschwindigkeit und der Offsetvektor von beiden Augen geteilt, jedoch mit leichten Unterschieden. So ist der minimale und maximale Winkel, den sich die Augen um die y-Achse rotieren können, für das zweite Auge vertauscht und ebenso die x-Komponente des Offsetvektors negiert. Weitere Unterschiede bestehen zum einen darin, dass der maximale Winkel entlang einer Rotationsachse als erreicht gilt, sobald mindestens eines der beiden Augen den maximalen Winkel

entlang dieser Achse erreicht und zum anderen darin, dass die vier Funktionen zur Abfrage der Richtungsvektoren und Position des Knochens, die normierten Richtungsvektoren aus der Addition der jeweiligen Richtungsvektoren beider Augen und die Position auf halbem Weg zwischen den beiden Augen zurückgeben.

RootGazeBone

Der dritte Knochen, der nicht über die GenericGazeBone-Komponente rotiert werden kann, ist der Rootknochen des Charakters. Dieser muss sich um das komplette Rotieren des Charakters kümmern, sobald alle anderen Knochen ihre maximalen Winkel erreicht haben und benötigt daher eine eigene Komponente, um den zu rotierenden Winkel, in diesem Fall, an die Movement-Komponente des Charakters zu übergeben. Dazu wird die RootGazeBone-Komponente verwendet.

Vor der Berechnung des zu rotierenden Winkels muss, zusätzlich zu der bereits erwähnten Prüfung, ob die vorherigen Knochen ihre maximalen Winkel erreicht haben, geprüft werden, ob sich der Charakter nicht bereits dreht oder bewegt. Letzteres ist nötig, da die Rotation während der Bewegung anders gehandhabt wird. Sind alle drei Anforderungen erfüllt, wird der Winkel zwischen der Vorwärtsrichtung und der Richtung zum Blickziel um die y-Achse des Rootknochens berechnet. Ist der absolute Wert dieses Winkels größer als ein gewisser Winkel, wird die Rotation des Charakters um den berechneten Winkel gestartet.

Zur Rotation während der Bewegung wird die Zielposition direkt an die Movement-Komponente des Charakters übergeben. Diese rotiert dabei nicht nur die untere Hälfte des Charakters, sondern den kompletten Charakter in die Richtung des Zieles.

```
public override void LookAtGazeTarget(Vector3 targetLocation, bool previousGazeBonesReachedMaxAngle = true)
{
    //Haben alle vorherigen Knochen ihre maximalen Winkel erreicht?
    if (previousGazeBonesReachedMaxAngle)
    {
        //Dreht und bewegt sich der Charakter momentan nicht?
        if (!m_Movement.IsTurning() && !m_Movement.IsMoving())
        {
            Vector3 directionToTarget = targetLocation - GetPosition();

            float angle = Vector3.SignedAngle(GetForward(), directionToTarget, GetUp());
            //Ist der Winkel größer als der Winkel, ab dem die Rotation gestartet werden sollte?
            if (Mathf.Abs(angle) > m_StartTurningAngle)
            {
                m_Movement.StartTurning(angle);
            }
        }
    }
    //Bewegt sich der Charakter?
    if (m_Movement.IsMoving())
    {
        m_Movement.TurnToPositionWhileMoving(targetLocation);
    }
}
```

ABBILDUNG 20: DIE LOOKATGAZETARGET-FUNKTION ZUR AUSRICHTUNG DES ROOTKNOCHENS UND DER ROTATION DES KOMPLETTEN CHARAKTERS.

5.6 BLICK DES CHARAKTERS UND SPIELERS

Die Blicke des Spielers und Charakters werden jeweils durch die Klassen PlayerGaze und CharacterGaze simuliert. Die Basisklasse der beiden definiert, wie bei den anderen Komponentenarten des Blicksystems, ebenfalls verschiedene grundlegende Variablen und

Funktionen. Dazu zählen, unter anderem, ein Winkel, um den die Blickrichtung beim Ansehen einer Position abweichen darf, die Position aber trotzdem noch als angesehen gilt und verschiedene Funktionen, um, wie bei den GazeBone-Komponenten, die drei Richtungsvektoren und die Position der Augen, unabhängig von der Art des Blickes, zu liefern.

PlayerGaze

Um den Blick des Spielers in dieser Implementation zu simulieren, muss die PlayerGaze-Komponente bestimmen können welcher Punkt vom Spieler momentan fokussiert wird. Der fokussierte Punkt wird dabei durch eine weitere Komponente dargestellt. Diese enthält sowohl eine Variable für deren momentan gesetzte Position als auch zwei weitere Variablen, um die Zeiten, die seit dem letzten Setzen der Position vergangen sein müssen, zu bestimmen, ab der der fokussierte Punkt als nicht mehr vor kurzem erstellt gilt oder zu alt ist, um als fokussiert zu gelten.

Das Bestimmen und Verwalten dieses fokussierten Punktes in der PlayerGaze-Komponente erfolgen, indem der vom Spieler zu einem bestimmten Zeitpunkt angesehene Punkt in der Umgebung in kleinen zeitlich variierenden Abständen in eine Liste eingefügt wird. Die Anzahl an Einträgen in dieser Liste ist begrenzt, wobei die jeweils ältesten Einträge gelöscht werden, sobald die maximale Anzahl überschritten ist. Vor jedem Einfügen eines neuen Punktes wird überprüft wie viele zuvor angesehene Punkte in dieser Liste sich nahe genug am neusten Punkt befinden. Überschreitet diese Menge eine variierende Mindestanzahl, gilt der Punkt als fokussiert und wird als Position des fokussierten Punktes des Spielers gesetzt. Der fokussierte Punkt des Spielers ist gültig, solange er existiert und nicht zu alt ist.

```
void Update()
{
    m_LookAtPointCreationTimer -= Time.deltaTime;
    if (m_LookAtPointCreationTimer < 0.0f)
    {
        if (!IsLookingAtObjectWithTag("Character"))
        {
            Vector3 position = GetLookedAtPosition();
            UpdateFocusPoint(position);
            AddLookAtPoint(position);
        }

        m_LookAtPointCreationTimer = Random.Range(m_TimeBetweenLookAtPointCreationMin, m_TimeBetweenLookAtPointCreationMax);
    }
}
1 reference
public void UpdateFocusPoint(Vector3 position)
{
    int closeEnoughCount = 0;
    for (int i = 0; i < m_LookAtPoints.Count; i++)
    {
        if (Vector3.Distance(m_LookAtPoints[i], position) < m_MaxDistanceBetweenLookAtPointsForFocusPoint)
        {
            closeEnoughCount++;
        }
    }

    if (closeEnoughCount >= m_LookAtPointsNeededForFocusPoint)
    {
        m_FocusPoint.SetPosition(position);
    }
}
```

ABBILDUNG 21: DIE UPDATE- UND UPDATEFOCUSPOINT-FUNKTION DER PLAYERGAZE-KOMPONENTE IN DENEN DER ANGESEHENE PUNKT ZUR AKTUALISIERUNG DES FOKUSSIERTEN PUNKTES VERWENDET WIRD. DIE ADDLOOKATPOINT-FUNKTION FÜGT LEDIGLICH DEN ANGESEHENEN PUNKT IN DIE LISTE EIN UND LIMITIERT DEREN ANZAHL AN ENTHALTENEN PUNKTEN.

CharacterGaze

Die CharacterGaze-Komponente kümmert sich um die Aktualisierung und Verwaltung der Blickverhalten und Knochen-Komponenten des Charakters. Das Aktualisieren und eventuelle Wechseln des Blickverhaltens erfolgen in jedem Frame. Dazu wird zuerst das momentane Blickverhalten über dessen UpdateBehaviour-Funktion aktualisiert und anschließend abgefragt, ob das Blickverhalten gewechselt werden sollte. Tritt dieser Falle ein, erfolgt die Wahl des nächsten Blickverhaltens in zwei Schritten. Im Ersten werden die Wechselwahrscheinlichkeiten zu den möglichen nächsten Blickverhalten des momentanen berechnet, gespeichert und die Summe dieser gebildet. Im zweiten Schritt wird ein zufälliger Wert zwischen 0,0 und der Summe der Wechselwahrscheinlichkeiten gewählt und anschließend die möglichen Blickverhalten nacheinander durchgegangen. Die Wahl fällt dabei auf das erste Blickverhalten, bei dem die Summe aus dessen gespeicherter Wechselwahrscheinlichkeit und den Wechselwahrscheinlichkeiten der zuvor geprüften Blickverhalten größer als der gewählte Zufallswert ist.

```
protected void SwitchToNextGazeBehaviour()
{
    List<float> probabilities = new List<float>();
    float sumOfProbabilities = 0.0f;
    List<GazeBehaviour> possibleGazeBehaviours = m_CurrentGazeBehaviour.GetPossibleNextGazeBehaviours();
    for (int i = 0; i < possibleGazeBehaviours.Count; i++)
    {
        probabilities.Add(possibleGazeBehaviours[i].GetSwitchToProbability());
        sumOfProbabilities += probabilities[probabilities.Count - 1];
    }

    float randomValue = Random.Range(0.0f, sumOfProbabilities);
    float currentProbability = 0.0f;
    for (int k = 0; k < probabilities.Count; k++)
    {
        currentProbability += probabilities[k];
        if (currentProbability > randomValue)
        {
            SwitchToBehaviour(possibleGazeBehaviours[k]);
            break;
        }
    }
}
```

ABBILDUNG 22: DIE FUNKTION ZUR WAHL DES NÄCHSTEN BLICKVERHALTENS AUS DER LISTE DES MOMENTANEN.

Beim Wechsel zum neuen Blickverhalten selbst wird die OnExitBehaviour-Funktion des momentanen Blickverhaltens und die OnEnterBehaviour-Funktion des nächsten Blickverhaltens gerufen und den beiden Funktionen zusätzlich das nächste beziehungsweise momentane Blickverhalten übergeben.

Nachdem Unity die Rotation der Knochen in jedem Frame auf die Rotation, die sie nach der Animation haben sollten, zurücksetzt, erfolgt das Ergänzen der Knochenrotationen um die zusätzliche Rotation durch den Blick und daraufhin folgende Ausrichtung dieser Knochen in die Richtung des momentanen Blickzieles in Unitys LateUpdate-Funktion. In dieser sind alle Rotationen durch die Animationen des Charakters angewandt und die Knochen können dementsprechend rotiert werden, ohne dass deren Rotationen direkt wieder überschrieben werden würden. Die Reihenfolge, in der die Knochen um die letzte Blickrotation aktualisiert werden, spielt auf Grund des Speicherns und der Nutzung der lokalen Rotation der jeweiligen Knochen, keine Rolle.

Bei der Ausrichtung der Knochen, in die Richtung des momentanen Blickzieles, spielt die Reihenfolge jedoch eine wichtige Rolle, um zu bestimmen, wann sich bestimmte Knochen rotieren sollen. Alle GazeBone-Komponenten der Knochen werden dabei, startend mit der der Augen und endend mit der des Hüftknochens, nacheinander durchgegangen. In jedem Schritt wird, zusätzlich zur Übergabe der

Zielposition, ein Boolean übergeben, der angibt, ob mindestens einer der vorherigen Knochen seine maximalen Winkel nicht erreicht hat. Um den Boolean zu setzen, wird nach der Ausrichtung eines Knochens abgefragt, ob dieser den maximalen Winkel um seine x-Achse oder y-Achse erreicht hat. Ist keines der beiden der Fall, wird der Boolean auf den Wert false gesetzt.

```
protected void GazeBonesLookAtTarget(Vector3 targetPosition)
{
    bool previousBonesReachedMaxAngle = true;
    for (int k = 0; k < m_GazeBones.Count; k++)
    {
        m_GazeBones[k].LookAtGazeTarget(targetPosition, previousBonesReachedMaxAngle);
        //Hat der Knochen den maximalen Winkel weder entlang der X- noch der Y-Achse erreicht?
        if (!m_GazeBones[k].ReachedMaxAngle(true, true, false))
        {
            previousBonesReachedMaxAngle = false;
        }
    }
}
```

ABBILDUNG 23: DIE FUNKTION ZUR AUSRICHTUNG ALLER KNOCHEN ZUR ZIELPOSITION.

Damit der Charakter den Spieler ansieht, sobald letzterer die Air-Tap-Geste ausführt während er den Charakter ansieht, wird ähnlich wie bei dem bereits erwähnten LookAtPoint-Blickverhalten, darauf gewartet, dass das entsprechende Event ausgelöst wird. Dementsprechend findet ein garantierter Wechsel in das LookAtPlayer-Blickverhalten statt.

Damit die Wahrscheinlichkeit im LookAtPlayer-Blickverhalten bestimmt werden kann, dass der Charakter den Blick des Spielers wahrnimmt, wird eine Funktion bereitgestellt, die diese Wahrscheinlichkeit liefert. Die Wahl dieser Wahrscheinlichkeit geschieht in Abhängigkeit des Winkels zwischen der Blickrichtung des Charakters und der Richtung zu den Augen des Spielers. Ist dieser Winkel kleiner als ein bestimmter Winkel, in dem die Augen des zu testenden Blickes als im Fokus gelten, wird die Sehwahrscheinlichkeit des Blickes auf einen voreingestellten Wert gesetzt. Ist der Winkel jedoch größer als der Fokuswinkel, berechnet sich die Wahrscheinlichkeit wie folgt:

$$P(sehen\ des\ Blicks) = 1 - \frac{Winkel\ zu\ den\ Augen\ des\ Zieles\ -\ Fokuswinkel}{\frac{\mathrm{FOV}}{2} - Fokuswinkel}$$

Die Unterscheidung des horizontalen und vertikalen Sichtfeldwinkels (auch field of view oder kurz FOV genannt) ist für den Charakter in dieser Implementation nicht nötig, da beide dieser Winkel durch einen einzigen Winkel repräsentiert werden. Sollte der Winkel zu den Augen des Zieles größer als die halbe FOV des Charakters sein besteht allerdings das Problem, dass die Wahrscheinlichkeit negativ werden würde. Daher wird diese grundlegende Wahrscheinlichkeit anschließend noch auf einen Wert zwischen 0,0 und 1,0 begrenzt. Da es ebenfalls keinen Sinn ergeben würde, wenn die Wahrscheinlichkeit den Blick wahrzunehmen außerhalb des Fokuswinkels größer wäre als innerhalb, wird die berechnete Wahrscheinlichkeit außerhalb des Fokuswinkels auf die Fokuswahrscheinlichkeit gesetzt, fall sie größer als diese ist.

```
public override float ProbabilityOfSeeingGaze(Gaze gazeToSee)
{
    float probability = 0.0f;
    if (gazeToSee)
    {
        Vector3 directionToTargetEyes = gazeToSee.GetEyesPosition() - GetEyesPosition();
        float angleToTarget = Vector3.Angle(GetEyesForward(), directionToTargetEyes);
        //Ist der Winkel zum Ziel kleiner als der Fokuswinkel?
        if (angleToTarget < m_InFocusAngle)
        {
            probability = m_InFocusProbability;
        }
        else
        {
            float halfFOV = GetFOV() / 2.0f;
            probability = 1.0f - ((angleToTarget - m_InFocusAngle) / (halfFOV - m_InFocusAngle));
            probability = Mathf.Clamp01(probability);
            //Ist die Wahrscheinlichkeit größer als die Wahrscheinlichkeit im Fokus?
            if (probability > m_InFocusProbability)
            {
                probability = m_InFocusProbability;
            }
        }
    }

    return probability;
}
```

ABBILDUNG 24: DIE FUNKTION DER CHARACTERGAZE-KOMPONENTE ZUR BERECHNUNG DER SEHWAHRSCHEINLICHKEIT EINES ANDEREN BLICKES.

5.7 ZWISCHENMENSCHLICHE DISTANZ

Das Prüfen der Einhaltung der zwischenmenschlichen Distanz und die eventuelle Berechnung der Position, an die sich der Charakter bewegen sollte, wird durch die InterpersonalDistance-Komponente erledigt. Diese Komponente enthält eine Variable für die Distanz, ab der der Charakter berührt wird und eine List mit InterpersonalDistanceZone-Komponenten, welche die verschiedenen Distanzen, die der Charakter einnehmen kann, darstellen.

Die InterpersonalDistanceZone-Komponenten selbst basieren auf den von Hall definierten Distanzzonen (E. T. Hall, 1966, S. 114), wobei jede dieser Komponenten eine Variable für den Typen der Zone, zwei Variablen für die Distanzen der unteren und oberen Grenze dieser Zone und eine weitere Variable für die Wahrscheinlichkeit, in dieser Zone Blickkontakt mit dem Spieler herzustellen, enthält. Letztere Wahrscheinlichkeit gilt dabei standardmäßig nicht in der kompletten Zone, sondern nur in der Mitte dieser. Die einzige Ausnahme dazu besteht im Falle, dass keine nähere oder weiter entfernte Distanzzone existiert. In diesen Fällen gilt die Wahrscheinlichkeit ab der oberen beziehungsweise unteren Grenze.

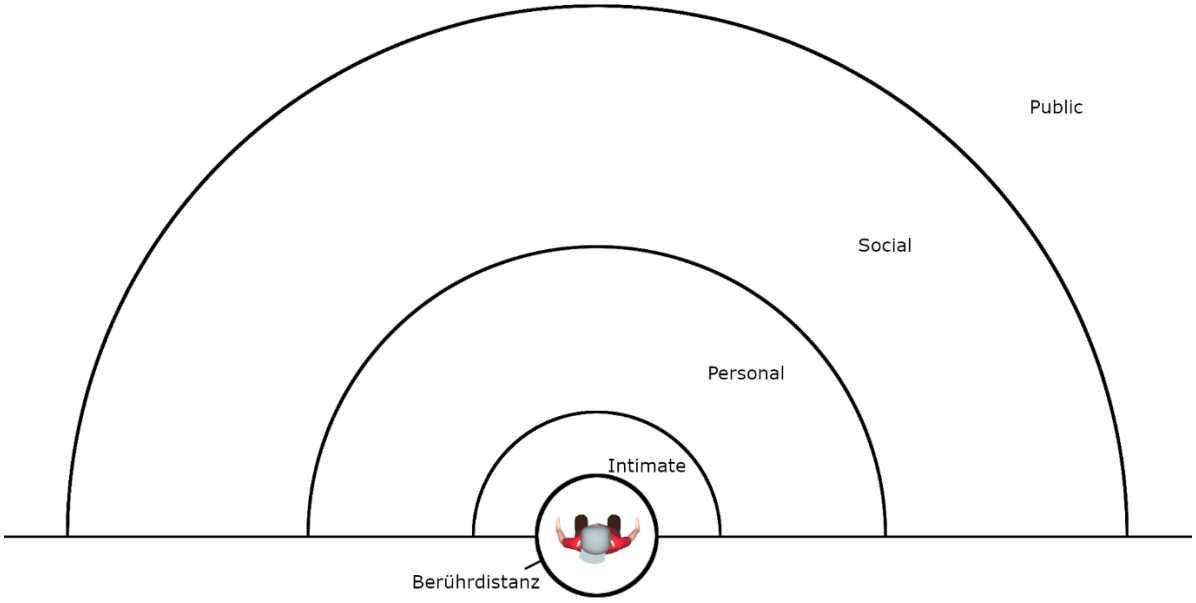

ABBILDUNG 25: DIE GROBE VISUALISIERUNG DER DISTANZZONEN DES CHARAKTERS, INKLUSIVE DES EINFLUSSES DESSEN SICHTFELDES VON 180° AUF DIE WAHRNEHMUNG DES SPIELERS. NUR INNERHALB DER BERÜHRDISTANZ NIMMT DER CHARAKTER DEN SPIELER UNABHÄNGIG VON SEINEM SICHTFELD WAHR.

Das Überprüfen der eingehaltenen Distanz erfolgt in der InterpersonalDistance-Komponente in jedem Frame. Die Entscheidung sich in die momentane Distanzzone zu bewegen, findet dabei auf Grund der horizontalen Distanz zwischen Charakter und Spieler und des Eintretens mindestens einer der folgenden Fälle statt:

- Der Spieler befindet sich in der Berührdistanz des Charakters.
- Der Charakter sieht den Spieler und bewegt sich bereits in die momentane Distanzzone, das Ziel der Bewegung befindet sich allerdings nicht mehr in dieser Zone.
- Der Charakter bewegt sich nicht in die Zone, sieht den Spieler und ist zu nahe an diesem.
- Der Charakter bewegt sich nicht in die Zone und sieht den Spieler an aber ist zu weit entfernt von diesem.

```
public bool ShouldMoveIntoCurrentZone()
{
    float distanceToTarget = GetHorizontalDistanceToTarget();
    //Berührt der Spieler den Charakter?
    if (IsTouchingTarget(distanceToTarget))
    {
        return true;
    }
    //Sieht der Charakter den Spieler?
    if (m_CharacterGaze && m_CharacterGaze.IsPositionInFOV(m_Target.transform.position))
    {
        //Bewegt sich der Charakter bereits in die momentane Distanzzone?
        if (m_IsMovingIntoZone)
        {
            //Ist das Ziel der Bewegung nicht in der momentanen Distanzzone?
            if (!IsCurrentDestinationInCurrentZone())
            {
                return true;
            }
        }
        else
        {
            //Ist der Charakter zu nah am Spieler?
            if (IsTooCloseToTarget(distanceToTarget))
            {
                return true;
            }
            //Ist der Charakter zu weit entfernt vom Spieler?
            if (IsTooFarAwayFromTarget(distanceToTarget))
            {
                //Sieht der Charakter den Spieler momentan an?
                if (m_CharacterGaze.GetCurrentGazeBehaviour() &&
                    m_CharacterGaze.GetCurrentGazeBehaviour().GetType() == typeof(LookAtPlayerGazeBehaviour))
                {
                    return true;
                }
            }
        }
    }

    return false;
}
```

ABBILDUNG 26: DIE FUNKTION DER INTERPERSONALDISTANCE-KOMPONENTE ZUR ENTSCHEIDUNG, OB SICH DER CHARAKTER IN DIE MOMENTANE DISTANZZONE BEWEGEN SOLLTE.

Sollte der Charakter, auf Grund einer dieser Fälle, versuchen sich in die momentane Distanzzone zu bewegen, muss zunächst eine gültige Position gefunden werden, die sich sowohl auf dem Navmesh als auch auf dem Boden und in dieser Zone befindet, bevor diese als Ziel der Bewegung gesetzt werden kann. Das versuchte Finden einer gültigen Position basiert dabei auf dem stichprobenartigen Testen verschiedener zufälliger Positionen in der entsprechenden Distanzzone. Dazu werden zunächst die tatsächlichen Grenzen der Zone berechnet, indem die von der momentanen Distanzzone gelieferten Grenzen jeweils mit der Berührdistanz des Charakters, dem Radius, in dem versucht wird eine Position auf dem Navmesh zu finden und einem Offset, um sicherzustellen, dass der Charakter nicht zu nahe an einer der beiden Grenzen landet, verrechnet werden. Die daraufhin folgende Berechnung der zu testenden Positionen erfolgt nicht ausgehend von der Position des Charakters, sondern der des Spielers. Das hat den Vorteil, dass bei der Suche einer gültigen Position, die zu testende Richtung langsam um den Spieler rotiert werden kann, um eine Position zu finden, die sich möglichst nahe an der Ausgangsposition des Charakters befindet, ohne großartig komplizierte eventuell aufwändige Umrechnungen vornehmen zu müssen.

Als Startrichtung der Tests wird die direkte Richtung vom Spieler zum Charakter gewählt. Diese wird normalisiert, mit der Distanz zur unteren Grenze der Zone multipliziert und anschließend auf die Position des Spielers addiert, um die Position am Anfang der Distanzzone in Richtung des Charakters zu erhalten. Auf diese Startposition wird bei jedem Testschritt ein direkt in die Zone zeigender Vektor

mit einer zufälligen Länge, die innerhalb der Zone liegt, addiert. Es kann allerdings nicht direkt getestet werden, ob sich um die daraus entstehende Position eine gültige Position auf dem Navmesh befindet, da das Umgebungsmesh und dementsprechend ebenfalls das Navmesh nicht zwingend komplett eben sind. Je nach eingestelltem Sampleradius und Höhenunterschied des Navmeshes an dieser Stelle könnte es dazu kommen, dass keine Position gefunden wird, obwohl theoretisch eine gültige Position existieren würde. Um dies zu verhindern wird ein Raycast mit einem bestimmten Offset von oberhalb der zu testende Position nach unten durchgeführt. Wird ein Objekt getroffen und ist dieses getroffene Objekt ein Teil des Bodens, wird geprüft, ob es eine Position auf dem Navmesh gibt, die sich nahe genug an der getesteten Position befindet und dementsprechend zurückgegeben werden kann. Kann nach der eingestellten Anzahl an Tests in dieser Richtung keine gültige Position gefunden werden, wird die Startposition um einen bestimmten Winkel nach jeweils links und rechts um die Spielerposition rotiert und der Vorgang wiederholt. Dies geschieht so lange bis der Winkel 180° erreicht ist. Wird in keinem Test eine gültige Position gefunden, kann sich der Charakter in diesem Frame nicht in Richtung der Zone bewegen.

ABBILDUNG 27: VISUALISIERUNG DER ZUFÄLLIGEN POSITIONSSTICHPROBEN IN DER MOMENTANEN DISTANZZONE DES CHARAKTER AUSGEHEND VON DER POSITION DES SPIELERS. STARTEND IN DER RICHTUNG DES CHARAKTERS IST DIE REIHENFOLGE DER GETESTETEN RICHTUNGEN MITTELS DES FARBVERLAUFS DER PUNKTE VERDEUTLICHT.

Um die Wahrscheinlichkeit den Spieler auf Grund der Distanz anzusehen abzufragen wird eine Funktion bereitgestellt, die diese Wahrscheinlichkeit berechnet. Die Wahrscheinlichkeit, die zurückgegeben wird, ist ein interpolierter Wert zwischen der Blickkontaktwahrscheinlichkeit in der momentanen Distanzzone und der Blickkontaktwahrscheinlichkeit in der nächstgelegenen Distanzzone. Der Interpolationswert selbst ermittelt sich, indem zunächst geprüft wird, in welcher Distanzzone sich der Spieler befindet und ob sich der Spieler unter- oder oberhalb der Summe, der durch diese Zone gelieferten Distanz, bei der die Blickkontaktwahrscheinlichkeit gilt und der Berührdistanz des Charakters befindet. Je nachdem wird entweder die näherliegende und

momentane Distanzzone oder die momentane und entferntere Distanzzone für die Interpolation gewählt. Im Allgemeinen berechnet sich der Interpolationswert daraufhin nach dieser Formel:

$$Interpolationswert = \frac{Distanz_{Spieler} - (Distanz_{\text{nähere Zone}} + Distanz_{Berührt})}{Distanz_{entferntere\ Zone} - Distanz_{\text{nähere Zone}}}$$

Wobei $Distanz_{Spieler}$ die Distanz zum Spieler, $Distanz_{Berührt}$ die Berührdistanz des Charakters und $Distanz_{\text{nähere Zone}}$ und $Distanz_{entferntere\ Zone}$ die Wahrscheinlichkeitsdistanzen der jeweils näheren und weiter entfernten der beiden Distanzzonen darstellt. In den Fällen, dass die Zone, in der sich der Spieler befindet, die nächste oder am weitesten entfernte ist und es daher eventuell keine nähere oder entferntere Zone gibt, wird die Blickkontaktwahrscheinlichkeit der momentanen Zone zurückgegeben.

5.8 SONSTIGE KOMPONENTEN

Um die nonverbalen Interaktionen mit dem Charakter glaubhafter wirken zu lassen, sind eine Reihe an zusätzlichen Komponenten und Effekten nötig. Dazu zählen sowohl das bereits erwähnte Blinzeln des Charakters als auch dessen variierende Idle-Animation. Zu den weiteren noch nicht erwähnten Effekten zählt zum einen das Vermeiden der Kollision des Charakters mit dem Spieler oder den geworfenen virtuellen Objekten bei der Bewegung in die momentane Distanzzone und zum anderen das Hinzufügen von Schatten unter sowohl den geworfenen Objekten als auch unter dem Charakter.

Das Vermeiden der Kollision mit den geworfenen Objekten, ist über das Anhängen einer NavMeshObstacle-Komponente an die jeweiligen Objekte und entsprechende Anpassen dieser relative trivial gelöst. Um die Kollision mit dem Spieler zu vermeiden, muss zusätzlich zum Anhängen einer solchen NavMeshObstacle-Komponente ebenfalls deren Rotation deaktiviert werden, um zu verhindern das sich diese bei der Rotation der Kamera mitrotiert.

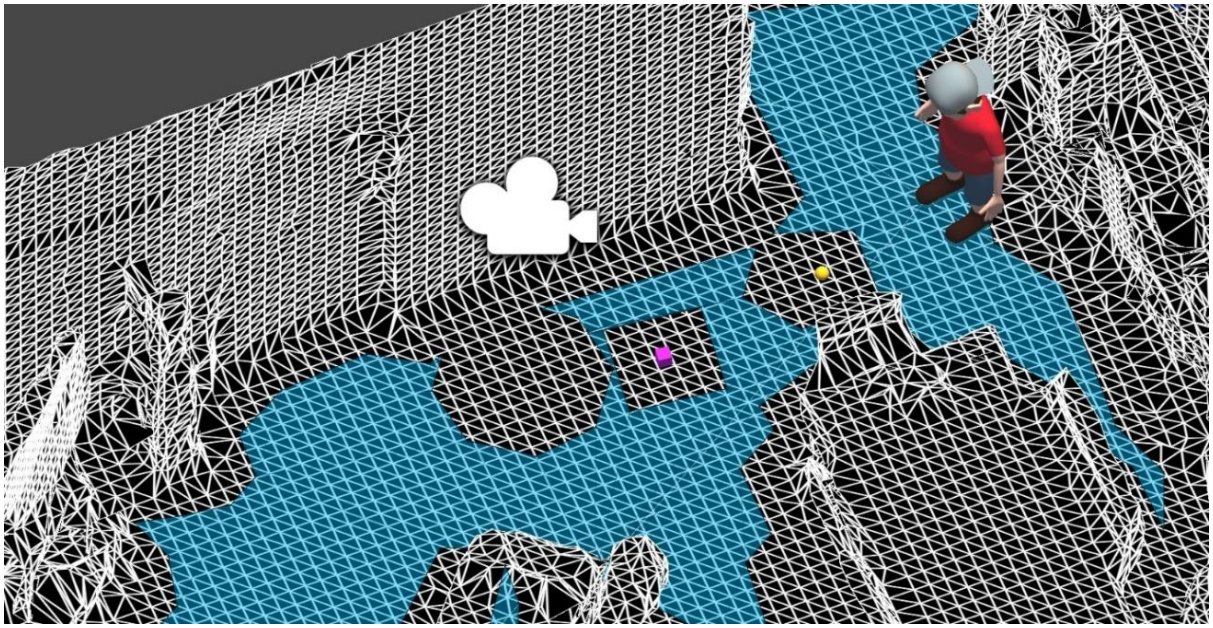

ABBILDUNG 28: DAS GENERIERTE NAVMESH IN EINER DER BEISPIELRAUMMESHE DES HOLOTOOLKITS, INKLUSIVE DER ERZEUGTEN BEREICHE UM DEN SPIELER UND DIE GEWORFENEN OBJEKTE, UM DIE KOLLISION WÄHREND DER BEWEGUNG ZU VERMEIDEN.

Das Hinzufügen von Schatten unter den Objekten und dem Charakter ist in Kombination mit der HoloLens mit verschiedenen Problemen verbunden. Die Verwendung von den in Spielen herkömmlichen Schatten ist auf Grund des additiven Displays der HoloLens, bei dem die Objekte angezeigt werden, indem die entsprechenden Pixel Licht auf das Licht der realen Umgebung addieren, nicht möglich. Zum einen werden die schwarzen oder zumindest ziemlich dunklen Stellen des Schattens nur wenig beziehungsweise gar nicht beleuchtet und sind daher auf der HoloLens nicht sichtbar und zum anderen kann die Umgebung dementsprechend nicht abgedunkelt werden. (Jakl, A., 2017, S. 1).

Bei der Verwendung eines Sprites unter den jeweiligen Objekten, der aus einem Kreis besteht, welcher ausgehend vom Mittelpunkt nach außen hin stetig transparenter wird, tritt bei dunklen Schatten das gleiche Problem auf. Wird die Helligkeit des Schattens erhöht oder der Schatten farbig angepasst, wirkt der Schatten, zumindest im Kontext dieser Anwendung, nicht mehr wie ein glaubhafter Schatten.

Die Lösung dieses Problems in dieser Anwendung nutzt einen Sprite mit einem negativen Schatten. Negative Schatten werden in der Mixed-Reality-Dokumentation[10] von Microsoft wie folgt beschrieben:

> Many designers have found that they can even more believably integrate holograms by creating a "negative shadow" on the surface that the hologram is sitting on. They do this by creating a soft glow on the ground around the hologram and then subtracting the "shadow" from the glow. The soft glow integrates with the light from the real world and the shadow grounds the hologram in the environment.

Damit sich die Sprites der Schatten der geworfenen Objekten, auf dem Boden unterhalb dieser befinden, werden jeweils Raycasts nach unten durchgeführt, treffen diese Raycasts ein Objekt werden die Sprites aktiviert und deren Position auf die des getroffenen Punktes gesetzt, anderenfalls werden die Sprites deaktiviert.

ABBILDUNG 29: DIE VERWENDETEN NEGATIVEN SCHATTEN UNTER DEM CHARAKTER UND EINEM DER WERFBAREN OBJEKTE. IN DEN SCREENSHOTS DER HOLOLENS SIND DIESE NAHEZU NICHT SICHTBAR, DAHER WERDEN SIE HIER ZUR VERDEUTLICHUNG AUF EINEM SCHWARZEN UNTERGRUND DARGESTELLT.

[10] What is a hologram? - Mixed Reality | Microsoft Docs - https://docs.microsoft.com/en-gb/windows/mixed-reality/hologram

6 DISKUSSION UND AUSBLICK

Alle Personen, die die Anwendung ausprobiert haben, sind der Meinung, dass die Nutzung unterschiedlicher Blickverhalten in Kombination mit dem Einhalten einer zwischenmenschlichen Distanz einen positiven Effekt auf die wahrgenommene Präsenz des Charakters hat. Inwiefern die Effekte dieser nonverbalen Verhalten den Nutzer tatsächlich beeinflussen, muss in eventuellen zukünftigen Arbeiten über Studien untersucht werden.

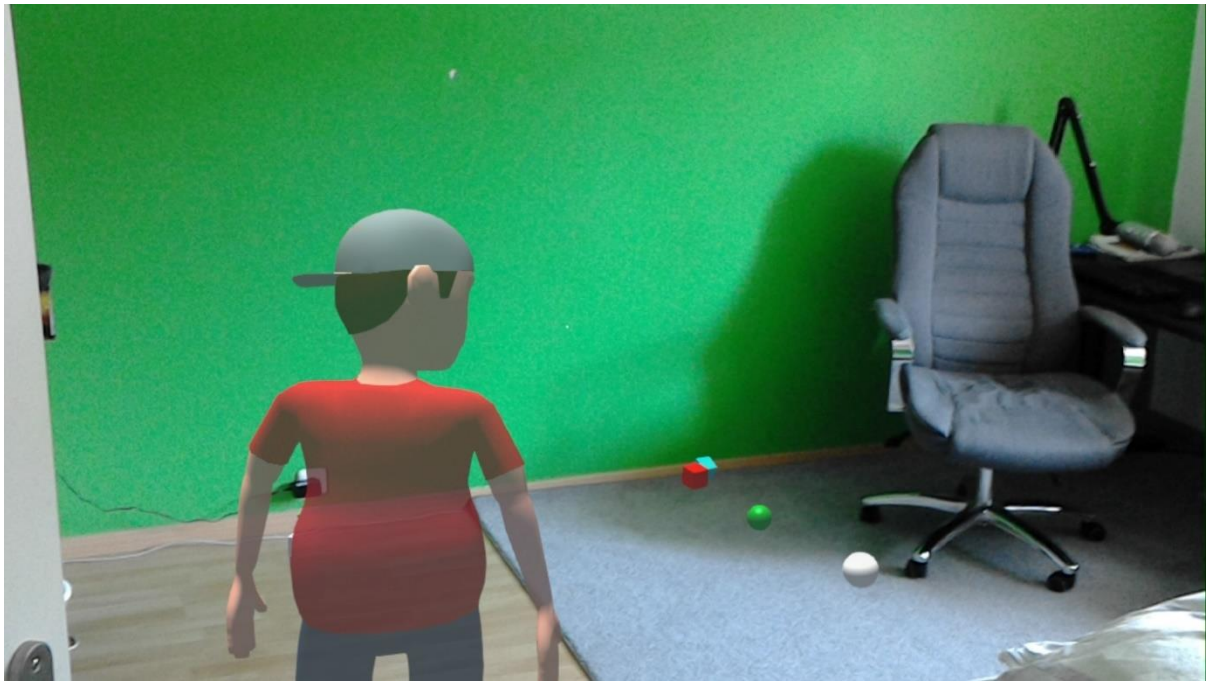

ABBILDUNG 30: EIN WEITERES BILD EINER SPIELSITUATION DER ANWENDUNG AUS DER SICHT DES SPIELERS.

Trotz dieser positiven Ergebnisse sind einigen Testern eine Handvoll Probleme mit der verwendeten Hardware und der momentanen Implementation aufgefallen. Die Größten dieser Probleme inklusive möglicher Lösungen werden im Folgenden kurz beschrieben.

Die meisten der Tester bemängeln vor allem das geringe Sichtfeld der verwendeten HoloLens. Auf Grund dessen ist es bei einigen nonverbalen Verhalten schwer zu sehen, wie der Charakter diese ausführt. Vor allem im Falle, dass der Charakter den Blick des Spielers verfolgt, ist es, je nach der Position des Charakters, schwer für den Spieler einen bestimmten Raumpunkt anzusehen und gleichzeitig den Charakter im Blickfeld zu halten, um dessen Rotation zu sehen. Dieses Problem kann allerdings grundsätzlich einfach gelöst werden. Entweder durch die weitere Anpassung des Charakters und Wahl der Szene, um den kompletten Charakter leichter in das Sichtfeld des Spielers zu rücken oder durch das Verwenden eine AR-Brille mit einem größeren Sichtfeld als die HoloLens, wie zum Beispiel der zweiten Generation der HoloLens.

Ein weiterer Effekt, der von einigen Testern bemängelt wird, ist der jeweilige Schatten unter den geworfenen Objekten und dem Charakter. Obwohl die Darstellung und Nutzung von Schatten in Kombination mit dem additiven Display der HoloLens durch die Verwendung von negativen Schatten zwar grundsätzlich möglich ist, bringen diese ebenfalls eine Reihe an eigenen Problemen mit sich. Das größte Problem ist dabei definitiv die Abhängigkeit von sowohl den Lichtverhältnissen in der Umgebung als auch den Helligkeitseinstellungen des Displays der HoloLens. Ist die Umgebung zu hell, sieht der Nutzer das addierte Licht um die Schatten nahezu oder gar nicht. Ist die Umgebung zu dunkel, sieht er den Schein um die Schatten viel zu deutlich. Vergleichsweise ist ersterer Fall nicht

allzu problematisch, da es dabei lediglich so wirkt, als ob das entsprechende Objekt keinen Schatten wirft. Das hat zwar trotzdem negative Folgen auf die Wahrnehmung des Charakters, ist aber um einiges weniger störend als ein konstanter weißer Schein auf dem Boden unter den Objekten im zweiten Fall. Eine eventuelle Lösung für dieses Problem wäre die automatische Anpassung der Transparenz oder Helligkeit der Schatten, je nach den Lichtverhältnissen in der Umgebung. Alternativ kann dieses Problem komplett umgangen werden, indem eine AR-Brille gewählt wird, die eine herkömmlichere Art der Schattendarstellung unterstützt.

Zu den technischen Verbesserungsmöglichkeiten, bezüglich des Codes, zählen vor allem zwei Dinge. Als Erstes sind viele der Blickverhalten in dieser Implementation auf die Interaktion mit einer einzigen weiteren virtuellen oder realen Person ausgelegt. Die Interaktionen zwischen virtuellen Charakteren und dem Spieler in Videospielen dagegen erfolgen zumeist nicht komplett isoliert voneinander, stattdessen interagieren mehrere NPCs gleichzeitig sowohl mit dem Spieler als auch anderen NPCs. In diesen Fällen kann das Einbeziehen mehrerer Blicke beim Wechsel und Ausführen der Blickverhalten höchstwahrscheinlich erheblich zum Spielerlebnis beitragen.

Ebenfalls verbessert werden kann das Finden einer Position, die sich in der momentanen Distanzzone des Charakters und auf dem Navmesh befindet. Die in Kapitel 5.7 beschriebene Vorgehensweise funktioniert zwar im Kontext des Umfangs dieser Anwendung und der Verwendung eines einzigen Charakters, da sich der Charakter relativ frei in der Umgebung bewegen kann und nur die Distanz zum Spieler beachten muss, bei komplexeren Anwendungen mit mehreren Charakteren, ist es jedoch ratsam effizientere Algorithmen zur Findung einer Zielposition zu verwenden.

Durch diese Arbeit entstehen verschiedene Ideen für Folgeprojekte. Neben der Durchführung von Studien zum Testen des Effektes der nonverbalen Verhalten auf den Nutzer ist es denkbar eine Anwendung zu entwickeln, in der mehrere virtuelle Charaktere sowohl mit dem Spieler als auch miteinander interagieren. Die Implementation weiterer Mittel der nonverbalen Kommunikation ist ebenfalls möglich, um die Interaktion und das Feedback, dass der Spieler bei dieser Interaktion erhält, zu verbessern. Dazu zählt sowohl die Verwendung von unterschiedlichen Gesichtsausdrücken und allgemeinen Variationen in den Bewegungsanimationen des Charakters als auch das eventuelle Aufheben oder versuchte Fangen von virtuellen Objekten.

Auch die Implementation und Nutzung von verschiedenen weiteren Aspekten künstlicher Intelligenz könnten in Folgeprojekten untersucht werden. Eine beispielhafte Idee dafür wäre eine Anwendung mit einem computergesteuerten Charakter dessen Beziehung zu einem bestimmten Spieler sich, in Abhängigkeit der Aktionen dieses Spielers, über einen längeren Zeitraum oder zumindest über mehrere Spielsitzungen ändert und der Charakter dementsprechend seine nonverbalen Verhalten gegenüber dem Spieler anpasst.

Eine weitere mögliche Verwendung künstlicher Intelligenz entsteht dadurch, dass individuelle Faktoren einer Person und deren Umgebung das nonverbale Verhalten beeinflussen. Die Charaktere in Videospielen und anderen Anwendungen haben, im Normalfall, allerdings kein Wissen bezüglich dieser Faktoren des Nutzers und passen daher ihr eigenes Verhalten nicht an das des Nutzers an. Das Entwickeln einer künstlichen Intelligenz die das Verhalten des Nutzers nach und nach lernt und entsprechend das Verhalten virtueller Charaktere an das individuelle Verhalten des Nutzers anpasst, könnte daher zur wahrgenommenen Präsenz dieser virtuellen Charaktere und dem Spielerlebnis sowohl in Unterhaltungsmedien als auch in Serious Games beitragen.

Abschließend ist zu sagen, dass die Verwendung von Blickverhalten und das Einhalten einer zwischenmenschlichen Distanz zum Spieler ein hohes Potenzial für die wahrgenommene Präsenz eines virtuellen Charakters zeigt und daher gegebenenfalls erheblich zum Effekt und Erlebnis von Unterhaltungsmedien und Serious Games beitragen kann. Dementsprechend sollte die Verwendung dieser nonverbalen Verhalten in Kombination mit virtuellen Charakteren definitiv in weiteren wissenschaftlichen Arbeiten untersucht werden.

7 ZUSAMMENFASSUNG

In dieser Arbeit ist die Implementation einer Augmented-Reality-Anwendung beschrieben, in der der Spieler auf verschiedene Arten und Weisen nonverbal mit einem computergesteuerten virtuellen Charakter kommunizieren und interagieren kann. Am Anfang wird auf die Motivation hinter dieser Arbeit, wie der unzureichenden Implementierung von einigen Mitteln der nonverbalen Kommunikation in verschiedenen Medien aber vor allem im Kontext von Videospielen und der Grund für die Wahl der Darstellung über Augmented Reality eingegangen. Daraufhin folgt die kurze Beschreibung der Zielsetzung dieser Arbeit und der Fokus auf die drei Mittel der nonverbalen Kommunikation: Blickkontakt, Blickverfolgung und zwischenmenschliche Distanz. Die Beschreibung einiger themenverwandten Arbeiten und deren Ergebnisse als auch die Beschreibung des Aufbaus der daraufhin folgenden Kapitel dieser Arbeit erfolgt gegen Ende des ersten Kapitels.

Das zweite Kapitel beschäftigt sich mit verschiedenen Grundlagen der nonverbalen Kommunikation im Allgemeinen und in Bezug auf die Nutzung mit virtuellen Charakteren. Genauer wird dabei sowohl auf die für diese Arbeit relevanten Gebiete der Blickverhalten und der zwischenmenschlichen Distanz als auch auf die Equilibrium Theory, welche diese beiden Gebiete in Verbindung miteinander bringt, eingegangen.

Gefolgt wird das Ganze im dritten Kapitel mit der Beschreibung der verwendeten Technologien. Zunächst wird dazu kurz das Gebiet der Augmented Reality erklärt und auf einige Vorteile, die durch die Verwendung dieser Technologie für die Präsenz des virtuellen Charakters entstehen, eingegangen. Im Anschluss wird die HoloLens Mixed-Reality-Brille im Allgemeinen und deren Funktionen, um die Interaktion zwischen virtuellen Objekten und der Realität zu ermöglichen, beschrieben. Gegen Ende dieses Kapitels wird sich mit der Nutzung der Unity Game Engine und der dadurch möglichen Nutzung des HoloToolkits, um das Arbeiten mit der HoloLens zu erleichtern befasst. Als letztes wird kurz erklärt was ein Navmesh ist und warum die NavMeshComponents von Unity Technologies für das Generieren dieses Navmeshes nötig sind.

Das vierte Kapitel dieser Arbeit definiert verschiedene Anforderungen, die das Blicksystem und System zum Einhalten der zwischenmenschlichen Distanz unabhängig von der letztendlichen Implementation erfüllen müssen. In diesem Kapitel wird zusätzlich sowohl das grundlegende Design der beiden Systeme in der Anwendung beschrieben als auch erklärt wie das jeweilige Design der beiden die jeweiligen Anforderungen erfüllt.

Nach der Beschreibung der Anforderungen und des jeweiligen Designs wird im folgenden Kapitel auf die Implementation der Anwendung eingegangen. Dies inkludiert sowohl das Design des Charakters und dessen nötigen Animationen als auch den allgemeinen Aufbau und Ablauf der Szene der Anwendung und die in der Implementation nötigen Komponenten. Dabei werden zuerst der Nutzen und die Funktionsweisen der fünf implementieren Blickverhalten des Charakters LookAtPlayer, LookAtFocusPointOfPlayer, Wander, LookAtPoint und LookAtEvent beschrieben. Nach den Blickverhalten werden die drei nötigen Knochen-Komponenten zur Rotation der beiden Augenknochen, des Rootknochens und der restlichen Knochen nähergebracht. Als letzte Komponentenart des Blicksystems wird auf die Blick-Komponenten des Charakters und Spielers eingegangen. Die Beschreibung der Komponente zum Einhalten der zwischenmenschlichen Distanz und der dafür verwendeten von Hall inspirierten Distanzzonen folgt im Anschluss zu den Komponenten des Blicksystems. Am Ende des Kapitels werden die beiden noch nicht genannten Komponenten zur Vermeidung der Kollision des Charakters mit dem Spieler und der geworfenen Objekte und zum Werfen eines Schattens unter diesen erklärt. In Zusammenhang mit letzterem wird ebenfalls auf die Nutzung von negativen Schatten zur Darstellung auf dem additiven Display der HoloLens eingegangen.

Am Ende der Arbeit findet eine kurze Diskussion statt und ein Ausblick auf mögliche Folgeprojekte wird gegeben. Dazu zählen die Adressierung der von verschiedenen Testpersonen gefundenen

Probleme und das Liefern von Ideen für eventuelle Folgeprojekte, die weitere Mittel der nonverbalen Kommunikation, mit den in dieser Arbeit implementierten kombinieren könnten oder eine künstliche Intelligenz nutzen könnten, um das Verhalten des Charakters individuell an den Spieler anzupassen.

8 ABBILDUNGSVERZEICHNIS

- Abbildung 1-30: Eigene Abbildung.

9 LITERATURVERZEICHNIS

- Anabuki, M., Kakuta, H., Tamura, H. & Yamamoto, H. (2000). Welbo: An Embodied Conversational Agent Living in Mixed Reality Space, CHI '00 Extended Abstracts on Human Factors in Computing Systems, S. 10-11.

- Anonymer Autor (2012). A Primer on Communication Studies[11]

- Argyle, M. & Dean, J. (1965). Eye-Contact, Distance and Affiliation, Sociometry, 10(3), S. 289-304.

- Asché, L. M., Bönsch, A., Habel, U., Kuhlen, T. W., Overath, H., Radke, S., Vierjahn, T. & Wendt, J. (2018). Social VR: How Personal Space is Affected by Virtual Agents' Emotions, 2018 IEEE Conference on Virtual Reality and 3D User Interfaces (VR), S. 199-206.

- Azuma, R. T., (1997). A Survey of Augmented Reality, Presence: Teleoperators and Virtual Environments, 6(4), S. 355-385.

- Bailenson, J., Beall, A. C., Blascovich, J. & Loomis, J. M. (2001). Equilibrium Theory Revisited: Mutual Gaze and Personal Space in Virtual Environments, Presence: Teleoperators and Virtual Environments, 10(6), S. 583-598.

- Bailenson, J., Bruder, G., Bölling, L., Haesler, S., Kim, K. & Welch, G. (2018). Does a Digital Assistant Need a Body? The Influence of Visual Embodiment and Social Behavior on the Perception of Intelligent Virtual Agents in AR, Proceedings of the 17th IEEE International Symposium on Mixed and Augmented Reality (ISMAR 2018), S. 105-114.

- Bailenson, J., Bruder, G., Kim, K., Maloney, D. & Welch, G. (2017). The Effects of Virtual Human's Spatial and Behavioral Coherence with Physical Objects on Social Presence in AR, Computer Animation and Virtual Worlds, 28(3-4), S. 1-16.

- Bailenson, J., Oh, C. S. & Welch, G. (2018). A Systematic Review of Social Presence: Definition, Antecedents, and Implications, Front. Robot. AI, 5, S. 114.

- Baron-Cohen, S., Wheelwright, S., & Jolliffe, T. (1997). Is there a "language of the eyes"? Evidence from normal adults, and adults with autism or Asperger syndrome. Visual Cognition, 4(3), S. 311-331.

[11] Sowohl der Autor dieses Werkes als auch der Herausgeber möchten anonym bleiben. Das Buch selbst ist unter einer Creative Commons by-nc-sa 3.0 Lizenz verfügbar. Die in dieser Arbeit referenzierte Version ist über diesen Link verfügbar:
https://dspace.lib.hawaii.edu/handle/10790/3307
Weitere Informationen zu den Lizenzumständen sind auf dieser Webseite verfügbar:
https://2012books.lardbucket.org/

- Biocca, F., Harms, C., & Burgoon, J. K. (2003). Towards A More Robust Theory and Measure of Social Presence: Review and Suggested Criteria, Presence: Teleoperators and Virtual Environments, 12(5), S. 456-480.

- Blumberg, B., Darrel, T., Maes, P. & Pentland, A. (1995). The ALIVE system: wireless, full-body interaction with autonomous agents, Proceedings Computer Animation'95, S. 11-18.

- Broz, F., Lehmann, H., Nehaniv, C. L. & Dautenhahn, K. (2012). Mutual Gaze, Personality, and Familiarity: Dual Eye-Tracking During Conversation, 2012 IEEE RO-MAN: The 21st IEEE International Symposium on Robot and Human Interactive Communication, S. 858-864.

- Bruder, G., Kim, K., Maloney, D. & Welch, G. (2016). The Influence of Real Human Personality on Social Presence with a Virtual Human in Augmented Reality, ICAT-EGVE '16: Proceedings of the 26th International Conference on Artificial Reality and Telexistence and the 21st Eurographics Symposium on Virtual Environments, S. 115-122.

- Capin, T. K., Guye-Vuillème, A., Pandzic, I. S., Thalmann, D. & Thalmann, N. M. (2009). Nonverbal Communication Interface for Collaborative Virtual Environments, Virtual Reality, 4, S. 49-59.

- Dicke, P. W., Marquardt, K., Ramezanpour, H. & Thier, P. (2017). Following Eye Gaze Activates a Patch in the Posterior Temporal Cortex That Is not Part of the Human “Face Patch” System, eNeuro, 4(2), S. 1-10.

- Ellgring, J. H. (1986). Nonverbale Kommunikation, Körpersprache in der schulischen Erziehung: pädagogische und fachdidaktische Aspekte non-verbaler Kommunikation, S. 7-48.

- Fast, E., Hartholt, A., Liewer, M., Mozgai, S., Reilly, A., Rizzo, A. & Whitcup, W. (2019). Virtual Humans in Augmented Reality: A First Step towards Real-World Embedded Virtual Roleplayers, HAI '19: Proceedings of the 7th International Conference on Human-Agent Interaction, S. 205-207.

- Gifford, R. (2011). The Role of Nonverbal Communication in Interpersonal Relations, Handbook of interpersonal psychology: Theory, research, assessment, and therapeutic interventions, S. 171-190.

- Giri, V. N. (2009). Nonverbal Communication Theories, Encyclopedia of Communication Theory, S. 690-694.

- Hall, E.T. (1966). The Hidden Dimension. NY: Anchor Books.

- Hans, A. & Hans, E. (2015). Kinesics, Haptics and Proxemics: Aspects of Non -Verbal Communication, IOSR Journal of Humanities and Social Science, 20(2), S. 47-52.

- Iizuka, Y. (1994). Gaze in Cooperative and Competitive Games, The Japanese Journal of Experimental Social Psychology, 33(3), S. 237-242.

- Jakl, A. (10.05.2020). How to add Negative Shadows to a HoloLens Scene – andreasjakl.com, https://www.andreasjakl.com/how-to-add-negative-shadows-to-a-hololens-scene/

- Johnson, R. B., Mercado, C. C., Sullivan, L. E. & Terry, K. J. (2009). The SAGE Glossary of the Social and Behavioral Sciences. SAGE Publications.

- Liu, R., Sahin, N., Salisbury, J. & Vahabzadeh, A. (2017). Feasibility of an Autism-Focused Augmented Reality Smartglasses System for Social Communication and Behavioral Coaching, Frontiers in Pediatrics, 5, S. 1- 10.

- n.a., (10.05.2020), What is a hologram? - Mixed Reality | Microsoft Docs, https://docs.microsoft.com/en-gb/windows/mixed-reality/hologram

- Patterson, M. L. (1995). Invited article: A parallel process model of nonverbal communication, Journal of Nonverbal Behavior, 19, S. 3-29.

- Patterson, M. L. (2009). Psychology of Nonverbal Communication and Interpersonal Interaction, PSYCHOLOGY, 3, S. 131-150.

- Tanenbaum, J., Seif El-Nasr, M., & Nixon, M. (2014). Nonverbal Communication in Virtual Worlds. Understanding and Designing Expressive Characters. Pittsburgh, PA: ETC Press.

10 ERKLÄRUNGEN

10.1 SELBSTSTÄNDIGKEITSERKLÄRUNG

Hiermit erkläre ich, dass ich die Bachelorarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die aus fremden Quellen direkt oder indirekt übernommenen Gedanken als solche kenntlich gemacht habe.

Die Arbeit habe ich bisher keinem anderen Prüfungsamt in gleicher oder vergleichbarer Form vorgelegt. Sie wurde bisher nicht veröffentlicht.

Ort, Datum Unterschrift

10.2 ERMÄCHTIGUNG

☐ Hiermit ermächtige ich die Hochschule Kempten zur Veröffentlichung einer Kurzzusammenfassung sowie Bilder/Screenshots und ggf. angefertigte Videos meiner studentischen Arbeit z. B. auf gedruckten Medien oder auf einer Internetseite der Hochschule Kempten zwecks Bewerbung des Bachelorstudiengangs „Game Engineering“ und des Masterstudiengangs „Game Engineering und Visual Computing“.

Dies betrifft insbesondere den Webauftritt der Hochschule Kempten inklusive der Webseite des Zentrums für Computerspiele und Simulation. Die Hochschule Kempten erhält das einfache, unentgeltliche Nutzungsrecht im Sinne der §§ 31 Abs. 2, 32 Abs. 3 Satz 3 Urheberrechtsgesetz (UrhG).

Ort, Datum Unterschrift