Redirected Walking in zufallsgenerierten Virtual Reality Leveln

Autor:
Karim Djemai

Matrikel: 6911548

Mensch-Computer Interaktion
Fachbereich Informatik

Erstgutachter: Prof. Dr. Frank Steinicke
Zweitgutachter: Dr. Eike Langbehn

Hamburg, 1. November 2021

Abstract

One of the greatest challenges in dealing with modern virtual reality (VR) technology is the limitation of the accessible area (tracking space). This is caused by spatially limited tracking of the position and rotation of the headset and the controllers in some technologies (for example, the models of the "HTC VIVE" product line[1]) And also because of limited amount of vacant and safe area the average user has, which is necessary to be able to move around freely. There are different solutions to this problem. One of which are the so called "redirection"-techniques. This is an umbrella term for methods where subtle changes to the visual representation of the virtual environment are made, such that the user can move to areas of it that would otherwise be inaccessable because they would be located beyond the boundaries of the real tracking space. Meanwhile they actually stay inside those boundaries but the illusion they could move freely and without manipulation of the visual environment is kept. These methods can be combined in different ways and thus are adaptable to the context. This work investigates a certain context: procedurally generated levels. By combining so-called "rotational gains" and so-called "impossible spaces" and incorporating them into a level generation algorithm, I have succeeded in creating a level generation method that can produce any length, theoretically even infinitely long levels. The users move trough those levels with the repeated illusion of accessing a new room that would have previously been outside the boundaries of their real tracking space. This thesis also presents an informal pilot study in which subjects walked through levels of different lengths generated by the above-mentioned method using different techniques of locomotion. For one of them, real walking, the above mentioned redirection techniques associated with the method were used, for the other two (joystick control and teleportation) they were not. It was examined whether the test subjects in the real walking condition were better able to estimate the distance they had moved, and whether their sense of presence was higher than in the other two test conditions. No significant differences were found.

[1] https://www.vive.com/

Zusammenfassung

Eine der größten Hürden im Umgang mit moderner “virtual reality” (VR) Technologie ist die Begrenzung des begehbaren Bereiches (Trackingspace). Dies entsteht zum einen durch räumlich limitierte Erfassung (auf Englisch: tracking) der Position und Rotation des Headsets und der Controller bei einigen Technologien (beispielsweise den Modellen der “HTC VIVE”-Produktreihe[2]). Zum anderen durch die, für durschnittliche Nutzer:innen, nur begrenzt zur Verfügung stehende Menge von leerstehendem und sicherem Bereich, der nötig ist um sich darin frei bewegen zu können. Es gibt unterschiedliche Lösungsansätze für dieses Problem. Einen davon stellen die sogenannten “Redirection”-Techniken dar. Hierbei handelt es sich um einen Sammelbegriff für Methoden bei denen die Nutzer:in durch subtile Änderungen an der visuellen Darstellung der virtuellen Welt den Eindruck bekommen kann, sich über die Grenzen des realen Trackingspaces hinausbewegen zu können und dies auch zu tun. Dabei bleibt sie innerhalb dieser Grenzen, es wird aber die Illusion aufrecht erhalten, sie würde sich frei und ohne Manipulation der Darstellungsart der virtuellen Welt fortbewegen. Diese Methoden lassen sich auf unterschiedliche Weisen kombinieren und sind so dem Kontext flexibel anpassbar. Diese Arbeit untersucht dabei einen konkreten Kontext: Prozedural generierte Level. Durch die Kombination zweier dieser Redirection-Techniken, den so genannten “Rotation-Gains” und den sogenannten “Impossible-Spaces”, und durch die Inkorporation davon in einen Levelgenerirungsalgorithmus ist es mir gelungen, eine Level-Generierungs-Methode zu definieren, mit der beliebig lange, theoretisch sogar endlos lange Level erschaffen werden können. Dabei erhalten Nutzer:innen immer wieder die Illusion einen Raum zu betreten, der eigentlich außerhalb der Grenzen ihres Trackingspaces liegt, ohne die realen Grenze des Trackingspaces zu überschreiten. Die Thesis stellt zudem eine informelle Pilotierungsstudie vor, bei der Proband:innen verschieden lange Level, die mit der eben genannten Methode generiert wurden, mit verschiedenen Fortbewegungsarten durchschritten. Bei einer davon, dem natürlichen Gehen (Real-Walking) wurden dabei auch die der Methode zugehörigen Redirection-Methoden eingesetzt, bei den anderen beiden (Joystick-Steuerung und Teleportation) nicht. Dabei wurde untersucht ob die Proband:innen bei der ersten Versuchsbedingung die von ihnen fortbewegte Distanz im Nachhinein besser schätzen können, und ob ihr Präsenzgefühl höher war als bei den anderen beiden Versuchsbedingungen. Es konnten keine signifikanten Unterschiede festgestellt werden.

[2] https://www.vive.com/

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einführung

Zweibeiniges Gehen bietet als Fortbewegungsart durch virtuelle Umgebungen viele Vorteile gegenüber alternativen Fortbewegungsarten. Usoh et al. [31] beschreiben beispielsweise, ein höheres Präsenzgefühl der Nutzer:innen, wenn sich diese gehend durch die virtuelle Umgebung fortbewegten, als wenn diese per Knopfdruck durch die Szene flogen. Eine der drei von LaViola [13] zusammenfassten Theorien, die "Sensory Conflict Theory" (Sensorische Konflikt Theorie) erklärt die Ursache für so genannte "Cybersickness" oder auch Simulatorkrankheit: Nutzer:innen erleben in bestimmten, durch virtuelle Umgebungen hervorgerufenen, Situationen teilweise starke Übelkeit, weil Informationen verschiedener Sinnesmodalitäten widersprüchlich sind.

Demnach wäre es also Cybersickness senkend, wenn andere Sinnesmodalitäten, wie der Gleichgewichtssinn oder die Propriozeption zusätzlich zum visuellen Sinn die Eigenbewegung der Nutzer:in wahrnehmen. Und tatsächlich konnten Langbehn et al. [10] zeigen, dass Nutzer:innen, die sich in virtuellen Umgebungen fortbewegen signifikant weniger Cybersickness erleben, als bei der Fortbewegung mithilfe von Joysticks.

Leider bringt das natürliche Gehen (Real-Walking) auch den großen Nachteil mit sich, dass es in der Regel auf einen einzelnen Raum (den Trackingspace) beschränkt ist.

Dies entsteht zum einen, durch räumlich limitierte Erfassung (auf Englisch: tracking) der Position und Rotation von Headset und Controllern bei einigen Technologien, zum anderen durch die Raumgröße der meisten VR-Setups.

Zwar gibt es dazu auch Ausnahmen, (siehe beispielsweise [3]), jedoch sind diese dann mit großem Aufwand verbunden und nicht für jede Endnutzer:in umzusetzen.

1.1 Redirection Methoden

Eine Herangehensweise an dieses Problem sind so genannte "Redirection Techniques" (besser bekannt als Redirected Walking. Diese Begriffe werden oft austauschbar verwendet, siehe [12]). Dies ist ein Sammelbegriff für Methoden bei denen die Nutzer:in durch subtile Manipulationen an der Darstellung der virtuellen Umgebung, auf virtuelle Gebiete zugreifen kann, die sonst außerhalb des begehbaren Bereichs wären. Tatsächlich bleibt sie dabei aber nur innerhalb des realen Trackingspaces. Im Idealfall wird dabei die Illusion aufrecht erhalten sie würde sich unmanipuliert, frei, und in virtueller

und echter Umgebung identisch bewegen. Tatsächlich werden bei den meisten dieser Methoden gewisse Veränderungen zwischen der Nutzer:in und ihrem virtuellen Avatar etabliert (beispielsweise die Geh-Geschwindigkeit), sodass die Bewegung sich zwischen den beiden Umgebungen unterscheidet. Bei manchen dieser Methoden sorgt diese Veränderung dafür, dass die Bewegungen der Nutzer:in in gewisser Weise manipuliert werden, beispielsweise indem diese unbewusst versucht die Veränderungen auszugleichen.

Mit diesen Methoden lassen sich also einige Effekte erzielen, wie unter anderem die virtuell begehbare Fläche zu vergrößern. Im Folgenden werde ich nun zwei dieser Methoden genauer vorstellen.

1.1.1 Rotation-Gains

Rotation-Gains werden Kopfrotationen hinzugefügt, sodass sich die virtuelle Kamera leicht schneller oder langsamer dreht als der reale Kopf mit dem VR-Headset. Wie diese dargestellt werden können findet sich wie folgend bei Steinicke et al. [24]:

Kopfrotationen lassen sich mit der Schreibweise

$$R_{real} := (pitch_{real}, yaw_{real}, roll_{real})$$

darstellen, wobei *pitch*, *yaw* und *roll* die Eulerschen Winkel der Kopfrotation repräsentieren. Ein Rotation-Gain wird als Quotient des virtuellen Winkels und des realen Winkels einer Rotationsdimension definiert:

$$gR := \frac{R_{virtual}}{R_{real}}$$

Für alle 3 Winkel kann ein Rotation-Gain angewandt werden. Statt der eigentlichen Kopfrotation α, wird die Multiplikation aus α mit dem Rotation-Gain:

$$gR * \alpha$$

durchgeführt. Folglich rotieren virtuelle Kameras mit einem Rotation-Gain von $gR > 1$ schneller in die Richtung der realen Kopfdrehung als der reale Kopf, während solche mit einem Rotation-Gain von $gR < 1$ langsamer die Richtung drehen. Da für jeden Winkel der Kopfrotation ein Rotation-Gain definiert werden kann, können Rotation-Gains folgendermaßen dargestellt werden:

$$(gR_{pitch}, gR_{yaw}, gR_{roll})$$

Nach Steinicke et al. [24] wird generell für Redirection-Zwecke ein Rotation-Gain auf den yaw_{real} Winkel der Kopfrotation angewandt.

Durch Anwenden eines positiven Rotation-Gains auf Yaw-Rotationen nach rechts und eines neutralen oder sogar negativen Rotation-Gains für Rotationen nach links kann

der virtuelle Bereich, der dem Trackingspace entspricht um den realen Trackingspace nach rechts rotiert werden, mit dem Drehpunkt der Nutzer:innenposition. Dies gilt auch für die andere Richtung. Indem im vorhinein ein Winkel β, für die Ziel-Verdrehung der beiden Realitäten definiert wird, lässt sich beim Erreichen dieses Winkels das Anwenden des Rotation-Gains beenden, sodass die Welten sich nun um genau β Grad verdreht haben.

Dies ermöglicht Prozesse, bei denen in bestimmten Bereichen der virtuellen Realität ein solcher "gerichteter" Rotation-Gain bis zu einem bestimmten Ziel-Winkel um einen vorher definierten Drehpunkt[1] angewendet wird.

In solchen Prozessen ist die Verdrehung zwischen den beiden Realitäten designbar, da sie so deterministisch wird. Da sich nun die Realitäten verdreht haben, liegen nun virtuelle Orte innerhalb des durch den realen Trackingspace begrenzten, für die Nutzer:in begehbaren Bereichs, die es vorher nicht taten. Für die Nutzer:in kann so die Illusion entstehen, sie würde über die Grenzen des Trackingspaces hinaus schreiten können, ohne dies zu tun.

1.1.2 Impossible-Spaces

Um den begehbaren Bereich eines Trackingspace noch weiter zu vergrößern wurde eine Redirection Methode von Suma et al. [25] vorgestellt, bei der zwei oder mehr Räume in überlappenden Gebieten liegen, allerdings nur einer zur Zeit angezeigt wird. Es gibt dann unterschiedliche Bedingungen, wann welcher der Räume angezeigt wird. Beispielsweise wird Raum A nur angezeigt wenn die Nutzer:in den überlappenden Raum durch Tür a betritt und Raum B, wenn sie ihn durch Tür b betritt. Dafür ist es also notwendig, dass x verschiedene Zustände programmatisch ermöglicht werden, sodass immer einer von x verschiedenen möglichen Räumen angezeigt wird. Des Weiteren ist es notwendig ein Gebiet in der virtuellen Umgebung bereitzustellen, in dem zwischen den Zuständen gewechselt werden kann, ohne dass die Nutzer:in es merkt.

1.2 Generierte Level

Klassischer Weise werden Level in Computerspielen und virtuellen Umgebungen von Leveldesignern gestaltet. Dies erfordert Zeit und Fachwissen. Der Arbeitsaufwand wächst mit der Größe des Levels. Deshalb ist es unmöglich auf diese Art endlos große Level zu erschaffen. Eine alternative Levelerstellungsweise ist die so genannte "Prozedurale

[1] Es sollte sichergestellt werden, dass sich die Kopfposition der Nutzer:in möglichst nah an diesem Drehpunkt befindet, da sonst nicht der Eindruck entsteht, die Rotation wäre mit der Kopfrotation der Nutzer:in identisch. Außerdem kann es für große Verwirrung sorgen, wenn sich die Welt auf einmal nicht um die eigene Achse dreht, aber dennoch genau dann, wenn der Kopf bewegt wird.

Generierung", auch "Prozedurale Synthese" genannt. Dabei wird das Level von einem Algorithmus erschaffen, und kann somit theoretisch endlos groß werden.

Dies ermöglicht mit Hinblick auf die Erweiterung des begehbaren Bereichs durch Redirection Methoden einen weiteren spannenden Effekt. Mit Redirection-Maßnahmen ausgestattet, welche wiederholt angewendet werden können um immer wieder den begehbaren Bereich durch die virtuelle Umgebung zu transformieren, ist es möglich theoretisch endlos große, begehbare virtuelle Umgebungen zu erzeugen. Eine solche "Infinite-Walking" Methode wird in dieser Arbeit vorgestellt.

1.3 Infinite-Walking Methode

Die in dieser Arbeit vorgestellte Infinite-Walking Methode basiert darauf aufeinander folgende trackingspacegroße Räume zu generieren, die genau so angeordnet sind, dass ein 90°, gerichteter Rotation-Gain an einer bestimmten Stelle, in der Nähe einer der Ecken des Raumes, die virtuelle und die reale Umgebungen, so umeinander dreht, dass der reale Trackingspace nach Abschluss des gerichteten Rotation-Gains genau kongruent mit den Außenwänden des nächsten Raumes liegt. Die Nutzer:in kann diesen dann betreten und den gleichen Effekt an einer anderen Ecke des Raumes, in dem die sich nun befindet, wiederholen um den begehbaren Bereich auf den wieder nächsten Raum zu transformieren. Dies kann dann theoretisch endlos so wiederholt werden. Die Räume sind durch Korridore miteinander verbunden. Jeder Raum hat genau einen Korridor. Diese sind so gestaltet, dass der Drehpunkt die Mitte des Korridors kurz vor dessen Ende darstellt. Durch Zufallsfaktoren in der Levelgenerierung, ist es mit dieser Methode möglich, viele unterschiedliche Level, die auf dem eben genannten Prinzip basieren, zu generieren.

Wie die Generierung solcher Level funktioniert, wird in Kapitel 4 genauer beschrieben.

1.4 Information

Bei allen in dieser Arbeit abgebildeten Grafiken, Bildern, Screenshots und Tabellen handelt es sich um Eigengrafiken, sofern nicht explizit eine Bildquelle dazu angegeben ist.

Kapitel 2

Verwandte Arbeiten

In diesem Kapitel werden wissenschaftliche Arbeiten vorgestellt mit denen diese Arbeit zusammenhängt. Dabei werde ich zunächst auf solche Arbeiten eingehen, die sich mit dem Thema Real-Walking in virtuellen Umgebungen beschäftigen, danach verschiedene Redirection-Techniken vorstellen und dann auf das Thema der Level-Generierung eingehen. Zunächst stelle ich generelle Arbeiten zu dem Thema Real-Walking und dann zu Redirection Techniken vor, danach gehe ich konkreter auf die in diesem Projekt sehr im Fokus liegenden Rotation-Gains ein, um danach die auch genutzten Impossible-Spaces vorzustellen. Als nächstes stelle ich der Leser:in Arbeiten zu dem Thema der Prozeduralen Levelgenerierung vor. Dabei werde ich mich sowohl mit Artikeln über die genaue Definition dieses Bereichs beschäftigen, als auch eine Taxonomie zur Einordnung von Prozeduren zur Inhaltsgenerierung zitieren. Die Kombination von generierten Leveln und Redirection-Techniken führen zu dem sogenannten "Infinite-Walking". Mit den Arbeiten zu diesem Thema wird das Kapitel abgeschlossen.

2.1 Real-Walking

1995 zeigen Slater et al. [22], dass Proband:innen ein höheres Präsenzgefühl angaben, wenn sie die von ihnen vorgestellte Technik "Walking-In-Place" nutzten, als wenn sie sich per Knopfdruck durch die Welt bewegten. Hierbei handelte es sich um eine Virtual-Walking Technik, bei der die Proband:innen eine Gehbewegung simulierten die dann digital erfasst und in virtuelle Fortbewegung umgewandelt wurde. Dieses Experiment wurde 1999 von Usoh et al. [31] repliziert, wobei nun die Option wirklich zu gehen ("Real-Walking") gegeben war. Dabei hatten die Proband:innen nochmal ein signifikant höheres Präsenzgefühl, als bei den beiden anderen Optionen (Virtual-Walking und Push-Button-Fly).

Des Weiteren zeigen Arbeiten, dass virtuelle Fortbewegungsarten, die anders als Real-Walking nicht den vestibulären Sinn und die Propriozeption stimulieren, wahrscheinlicher die sogenannte "Simulatorkrankheit" (simulator sickness) auslösen [2] und dass die Nutzer:innen damit weniger effektiv navigieren. [20]

Wenn Designer eine Real-Walking-Umgebung erstellen, müssen sie dabei schon die Dimensionen des Trackingspaces kennen. Da man aber nicht davon ausgehen kann, dass unterschiedliche Nutzer:innen gleiche Trackingspacedimensionen zur Verfügung haben,

entsteht ein Problem. Dies versuchen Marwecki et al. in ihrer Arbeit [14] zu Lösen. Sie stellen dabei das Softwaresystem "Scenograph" vor, welches große virtuelle Umgebungen in mehrere kleinere, teilweise anders geformte Umgebungen, mit prozedural generierten Verbindungen aufteilt ohne dabei die narrative Struktur der ursprünglichen Umgebung zu verändern.

Ein anderer Ansatz um Nutzer:innen mit begrenztem Trackingspaceplatz Real-Walking-Erfahrungen zu ermöglichen ist es den von der Nutzer:in begehbaren Bereich, in der virtuellen Umgebung, zu vergrößern. Eine vielversprechende Art dies zu erreichen sind Redirection-Techniken.

2.2 Redirection Techniken

Razzaque et al. [18] stellten 2001 die Methode des "Redirected-Walking" vor, bei der die Nutzer:innen unwissentlich durch den Trackingspace gelenkt werden, dabei aber die Illusion entsteht, sie würden sich über die Grenzen dessen hinausbewegen. Die Technik basiert darauf, dass der visuelle Sinn dominanter ist als andere Sinne, mit denen man seine Orientierung im Raum bestimmen kann. Seit dem gibt es zahlreiche weitere Techniken um den selben Effekt zu erzielen oder um ihn weiterzuentwickeln. Der Ansatz, die verschiedenen Manipulationseffekte als "Gains" zu beschreiben, findet sich bei Steinicke et al. [24]. Dort wird untersucht, wie subtil diese Manipulationen sein müssen um nicht von der Nutzer:in erkannt zu werden.

Es konnte gezeigt werden, dass Proband:innen in virtuellen Umgebungen, welche Redirection-Techniken nutzten um Real-Walking zu ermöglichen, signifikant besser unbewusst räumliches Wissen über diese Umgebungen sammelten, signifikant bessere Navigation und Wegfindung aufwiesen und die Größe der Umgebung signifikant besser einschätzen konnten als in Umgebungen, die andere Fortbewegungsarten nutzen, wie Walking-In-Place, Joystick-Steuerung oder Teleportation [17], [10].

Eine Taxonomie über die verschiedenen Redirection Techniken stellten 2012 Suma et al. [26] vor. Die unterschiedlichen Techniken werden in die Kategorien: "Repositioning" (Repositionierung) oder "Reorientation" (Reorientierung), "Subtle" (subtil) oder "Overt" (unverborgen), und "Discrete" (diskret) oder "Continuous" (kontinuierlich) unterteilt.

2.2.1 Rotation Gains

Bei Rotation-Gains handelt es sich nach Sumas Taxonomie [26] um eine kontinuierliche, subtile Reorientierungstechnik. In der Arbeit [24] untersuchten Steinicke et al. verschiedene subtile Redirection-Techniken darauf, wie stark die Manipulation sein darf, bevor Proband:innen erkennen, ob sie eingesetzt wurde oder nicht. Dazu teilten sie die verschiedenen Elemente, die für Redirected Walking eingesetzt werden, in drei verschiedene Gains ein: "Translation-Gains", "Rotation-Gains" und "Curvature-Gains". Es stellte sich

heraus, dass Nutzer:innen physisch um bis zu 49% mehr oder um bis zu 20% weniger als die wahrgenommene virtuelle Rotation, rotiert werden können, ohne die Diskrepanz zu bemerken, also Rotation-Gains $0,8 < gR < 1,49$ unentdeckt bleiben.

Des Weiteren wurde festgestellt, dass Geh-Distanzen unbemerkt um bis zu 14% herunter- oder um bis zu 26% heraufskaliert werden können und dass Nutzer:innen erst bemerken, dass sie in einem kreisförmigem Bogen durch den Trackingspace geleitet werden, wenn dessen Radius 22 Meter oder kleiner ist.

2.2.2 Impossible-Spaces

Bei "Impossible-Spaces" handelt es sich um eine von Suma et al. [25] vorgestellte Redirection-Technik, bei der sich die Architektur der virtuellen Umgebung auf nicht-euklidische Weise verändert, sodass solche Gebiete in der Realität nicht existieren könnten. Die Räume überlappen einander, allerdings wird jeweils nur einer der überlappenden Räume angezeigt. Hierbei handelt es sich nach der schon erwähnten Taxonomie um eine subtile diskrete Redirection-Technik.

In einer Forschungsdemonstration [11] stellten Langbehn et al. eine Weise vor mit der Impossible-Spaces mit traditionelleren Redirected-Walking Methoden kombiniert werden können, sodass beide Methoden ihren Effekt beitragen können.

2.3 Prozedural generierte Level

2.3.1 Definition

Der Artikel [28] von Togelius et al. definiert prozedurale Generierung von Spiel-Inhalten (procedural (game-)content generation oder auch PCG) als:

> "[...] creating game content automatically, through algorithmic means."
>
> ("[...] algorithmisch, automatisch, (Computer-)spiel Inhalte erstellen.")

In ihrer späteren Arbeit hingegen [27] definieren Togelius et al. PCG folgendermaßen neu:

> "We can therefore tentatively redefine PCG as the algorithmical creation of game content with limited or indirect user input."
>
> ("Wir können PCG daher versuchsweise als die algorithmische Erstellung von Spielinhalten mit begrenzter oder indirekter Benutzereingabe neu definieren.")

um unter anderem miteinzubeziehen, dass einige PCG-Algorithmen Nutzer- oder Designerinput miteinbeziehen können und somit nicht mehr "automatisch" Inhalte generieren. Außerdem wollen sie an der Definition festhalten, dass Nutzerinput typischerweise zumindest indirekt (mindestens durch Druck eines Startknopfes) erforderlich ist, um Inhalte zu generieren.

Mit (Spiel-)Inhalten sind in diesen Definitionen unterschiedlichste Elemente in Videospielen gemeint, unter anderem Texturen und Musik. Auch die Geschichte des Spiels kann prozedural generiert werden. Im Rahmen dieser Arbeit hingegen beschäftige ich mich lediglich mit PCG zur Erstellung von Leveln.

2.3.2 Taxonomie

In ihrer Arbeit [28] stellen Togelius et al. eine Taxonomie für PCG vor, die aus folgenden Kategorien besteht:

"Online versus offline" (Zur Laufzeit versus während der Entwicklung),

"Necessary vs optional" (Müssen die Spieler:innen den generierten Bereich des Spiels absolvieren oder nicht?),

"Random seeds versus Parameter Vectors" (auch: "degrees of control": Wieviel Einfluss hat die Spieler:in auf den generierten Inhalt, wird nur ein zufälliger RNG-Seed (Random-Number-Generator-Seed) als Eingabe in den Zufallsgenerator genutzt oder wird sein bisheriges Spielverhalten analysiert und bei der Generierung beachtet?),

"Stochastic vs deterministic" (Wird bei gleicher Eingabe (abgesehen vom RNG-Seed) auch der gleiche Inhalt generiert?) und

"Constructive vs generate-and-test" (Generiert der Algorthmus direkt nur korrekte Ausgaben, oder funktioniert er so, dass er fortlaufend Versuche generiert und dann validiert ob sie korrekt sind und sie dann erst ausgibt.)

Der in dieser Arbeit beschriebene PCG-Algorithmus lässt sich dementsprechend eher in diese Kategorien der Taxonomie einordnen, als in ihre jeweiligen Alternativen: Online, necessary, random seeds, stochastic and constructive.

2.4 Infinite walking

Viele der Redirection-Techniken simulieren die Vergrößerung des begehbaren Bereichs, doch dennoch bleibt die begehbare Fläche limitiert. Solange die virtuelle Umgebung von menschlichen Designern erschaffen werden muss, ist sie begrenzt. Wenn jedoch PCG genutzt wird um die virtuelle Umgebung zu erschaffen lässt sie sich theoretisch endlos weit durchschreiten, weil die Generierung während der Erkundung der Welt fortgeführt werden kann. Wenn die virtuelle Umgebung also, mit Hilfe von PCG, theoretisch endlos

weit erkundet werden kann, spricht man vom “Infinite-(Real)-Walking”. In der Regel lässt sich dieser Zustand erreichen, indem man Redirection-Techniken (um über den Trackingspace hinaus gehen zu können) mit prozeduraler Levelgenerierung (um die Welt während der Laufzeit weiter zu generieren) kombiniert.

Ein Beispiel für eine solche Technik stellen Vasylevska et al. in ihrer Arbeit [33] vor. Ihr Algorithmus generiert fortlaufend Räume, innerhalb des Trackingspaces, die einander überlappen können (Impossible-Spaces) und verbindet sie mit Korridoren, sodass die Nutzer:in von einem Raum zum nächsten gehen kann. Besonders bei Umgebungen in denen der Inhalt der Räume mehr im Fokus steht als das spezifische Layout der Räume (beispielsweise ein Museeum) ist Technik sehr geeignet.

Einen sehr ähnlichen Ansatz nutzt das VR-Spiel “Tea for God” [6] bei dem die Nutzer:in durch ein endlos groß scheinendes Labyrinth von Korridoren gehen kann. Der Entwickler Jarosław (Void Room) Ciupiński erklärt in seinen Devlogs (beispielsweise [5] oder [4]) genauer wie der Ansatz funktioniert. Die Welt besteht aus einem prozedural generierten Netz von verbundenen Zellen, die jeweils einen Raum repräsentieren und mit Korridoren verbunden sind. Auch hier basieren die Räume auf den vorher erwähnten Impossible-Spaces.

Einen anderen Ansatz verfolgt das in der Arbeit [3] von Cheng et al. vorgestellte Projekt “VRoamer”. Hier erkundet die Nutzer:in eine On-The-Fly generierte virtuelle Umgebung (auch hier besteht diese aus Räumen und Korridoren), während er durch die reale Welt läuft. Die Generierungssoftware erhält einen 3D-Kamera Input und kann so Wände, Säulen, Gegenstände, andere Menschen etc. beachten und dementsprechend die virtuelle Welt anpassen. Dort wird dann ein virtueller Gegenstand platziert, sodass die Nutzer:in nicht mit den Hindernissen der realen Welt kollidiert. Diese Technik ist nur bei VR-Systemen anwendbar, die nicht auf einen Trackingspace beschränkt sind, sondern (zum Beispiel mit Kameras am HMD (Head-Mounted-Display)) ihre Umgebung, und somit auch ihre eigene Position und Orientierung benötigen. Die Möglichkeiten, die virtuelle Umgebung zu erkunden, sind hier also nur durch den realen Platz, den die Nutzer:in zur freien Begehung zur Verfügung hat, limitiert. Streng genommen gilt die Definition von Infinite-Walking hier also nicht, sie sollte an dieser Stelle aber dennoch Erwähnung finden.

2.5 Einordung dieser Arbeit

Ähnlich zu der Arbeit [33] werde ich in dieser Arbeit eine Methode vorstellen, wie mit verschiedenen Redirection-Techniken und einem PCG-Algorithmus eine virtuelle Umgebung mit Infinite-Walking erstellt werden kann. Vergleichbar mit den Arbeiten [17] und [10] werde ich diese Methode dann in einem Experiment unter Testbedingungen mit alternativen Fortbewegungsarten, die dementsprechend kein Real-Walking ermöglichen, auf verschiedene Faktoren vergleichen.

Kapitel 3

Implementierung

In diesem Kapitel werde ich die technischen Elemente für die Umsetzung des in dieser Arbeit vorgestellten Experimentes vorstellen. Das Kapitel wird mit einer Übersicht über die Beziehungen zwischen den verschiedenen Scripts und Klassen des Projektes, in Form eines UML-Klassendiagramm beginnen. Danach stelle ich jede einzelne Klasse einmal vor. Anschließend beschreibe ich einzelne Bereiche der Implementierung genauer. Zuerst die Hierarchie der Unity-Szene, dann das Design der Level, der virtuelle Avatar, die Umsetzungen von Rotation-Gains und Impossible Space, die Umsetzung der für die später vorgestellte Pilotierungsstudie relevanten unterschiedlichen Fortbewegungsarten, der Mechanismus, der die Nutzer:in dazu anreizt, sich für die Rotation-Gains an der richtigen Stelle zu drehen, wie die Türen funktioneren, wie die unterschiedlichen Versuchsbedingungen ausgewählt und geladen werden können und wie die für die Studie gemessenen Daten gemessen und gespeichert werden.

Die gesamte Programmierung für dieses Projekt ist in der Entwicklungsumgebung Unity (Version: 2020.1.17f1) [30] und folglich mit der Programmiersprache "C#" erfolgt. Um die Software während der Entwicklung testen zu können und um damit das Experiment durchführen zu können wurde mir freundlicherweise eine "Oculus Quest 2" [15] Datenbrille, vom Arbeitsbereich Mensch-Computer-Interaktion der Universität Hamburg zur Verfügung gestellt. Um mit der Schnittstelle davon zu interagieren nutzt das Projekt das, von Oculus frei zur Nutzung gestellte, "Oculus Integration SDK" für Unity [16]. Um die Rotation-Gains zu implementieren wurde mir freundlicherweise die Library "Space-Extender" von dem Unternehmen "Curvature-Games" zur Verfügung gestellt, welche diesen Prozess stark vereinfacht. Diese findet sich mittlerweile auch bei GitHub [23].

3.1 Klassendiagramm

Um das Diagramm übersichtlich zu halten beschränkt es sich ausschließlich auf die Relationen zwischen den Klassen, es wird also nicht wie sonst in UML-Klassendiagrammen üblich die gesamte Schnittstelle aller Klassen inklusive ihren Attributen und ihren Methoden aufgelistet. Aus dem selben Grund wurden auch Grundklassen/-typen mit denen Standardmäßig in Unity gearbeitet wird (zum Beispiel "Vector3", "MonoBehaviour" oder "GameObject") weggelassen.

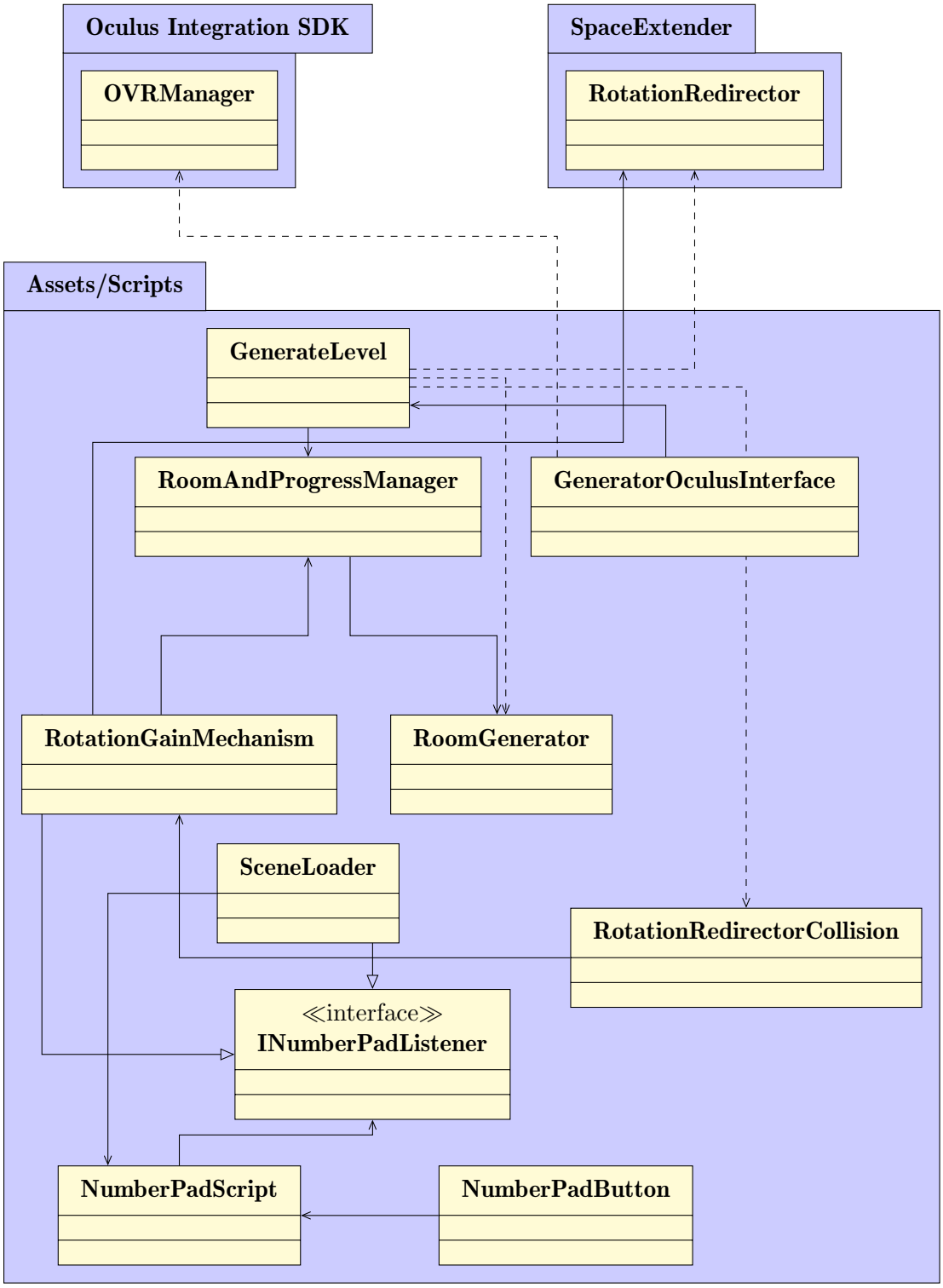

Abbildung 3.1: Ein Überblick über die Beziehungen der Klassen untereinander in Form eines UML-Klassendiagramms. Für die Übersichtlichkeit wurden Unity-eigene Klassen, wie zum Beispiel *Vector*3 oder *MonoBehaviour*, herausgelassen.

3.2 Vorstellung der Klassen

GeneratorOculusInterface Kommuniziert mit der Schnittstelle des "Oculus Integration SDK" um Informationen, die im Projekt benötigt werden bereitzustellen. Die wichtigsten Informationen, die hier geliefert werden, sind die Koordinaten der Begrenzungsecken des Trackingspaces. Die Nutzer:in der Oculus Quest 2 stellt im Setup der Brille den sogenannten "Guardian" ein. Dies ist dann eine Begrenzung, die sie um den begehbaren Raum in dem sie sich befindet zieht, und der immer dann in der virtuellen Realität eingeblendet wird, wenn sie dieser Begrenzung mit den Hand-Controllern oder der Datenbrille zu nahe kommt. Diese Begrenzung ist allerdings nicht rechteckig, was für den hier vorgestellten Levelgenerierungsalgorithmus aber notwendig wäre. Doch das Oculus SDK bietet in seiner Schnittstelle an, das größte, in diese Begrenzung passende, Rechteck auszugeben. Dies ist dann die Grundlage für die Raumgenerierung. Zudem ist diese Klasse dafür zuständig, je nach ausgewählter Testbedingung entsprechende Einstellungen an das Oculus SDK zu übergeben.

RoomGenerator Implementiert die Generierung der einzelnen Räume. Dafür liegt in dieser Klasse die zentrale "Generate"-Methode. Anhand der vier Eckkoordinaten des Raumes, der Richtung, in die der Raum zeigen soll und verschiedener Zusatzeigenschaften wie zum Beispiel, der Tiefe des zu generierenden Korridors, der Höhe der zu generierenden Türen, der Dicke der Wände und der Höhe des zu generierenden Korridors, erzeugt diese Methode das GameObject für den zu generierenden Raum. Außerdem erzeugt er das entsprechende Mesh für den Korridor und platziert die ansonsten nötigen Objekte (zum Beispiel Türen, Tafel, Eingabefeld) an den richtigen Positionen. Um die Tür nicht immer mittig zu platzieren bekommt die "Generate"-Methode außerdem einen Zufallswert zwischen 0 und 1 übergeben. Dieser bestimmt dann die Position der Tür, die in den Korridor hineinführt. (0 bedeutet ganz links, 1 hingegen ganz rechts). Des Weiteren bietet diese Klasse mehrere Methoden an, die Berechnungen über den Raum, den sie generiert haben, anstellen. So kann man sich beispielsweise berechnen lassen ob der Korridor des Raumes auf der längeren, oder der kürzeren Kante des Grundrechtecks liegt, wie die Koordinaten des Mittelpunkts eines Raumes inklusive des Korridors sind und wie die Koordinaten des Mittelpunktes des Bereichs vor dem Korridor sind. (An diese Stelle im ersten Raum wird die Proband:in bei den Testbedingungen, in denen kein Real-Walking stattfindet, teleportiert.) Die Klasse besteht aus circa 1000 Zeilen Programmierung und ist somit die längste selbst geschriebene Klasse des Projektes.

GenerateLevel Diese Klasse implementiert den Levelgenerierungsalgorithmus der genauer in Kapitel 4 beschrieben wird. Sie ist die zentrale Klasse des Projektes. In ihren öffentlichen Attributen können die Einstellungen für die zu generierende Testbedingung verändert werden. In ihrer "Generate"-Methode implementiert sie den Levelgenerierungsalgorithmus und verbindet darin die restlichen Teilmodule des Projekts. Sie ist also nicht nur dafür verantwortlich die "Generate"-Methode der "RoomGenerator"-Klasse mit den richtigen Eingabewerten aufzurufen, sondern plat-

ziert auch die Rotation-Gain Funktionalität an die richtige Stelle, und übergibt die generierten Räume an den "RoomAndProgressManager", in dem sie in einer Liste gespeichert werden.

RoomAndProgressManager Ist dafür zuständig den Fortschritt der Proband:in zu verfolgen und entsprechend zu reagieren wenn es notwendig ist. Beispielsweise öffnet er die entsprechenden Türen, wenn ein Raum abgeschlossen ist. Des Weiteren hält diese Klasse Instanzen der GameObjects für die Räume, in einer Liste, sodass sie dazu in der Lage ist, an den zugehörigen RoomGenerator-Instanzen Methoden aufzurufen. Diese Klasse ist außerdem dafür verantwortlich, dass nur der Raum in dem die Nutzer:in sich gerade befindet, der Raum davor und 2 Räume danach eingeblendet und sichtbar sind, um sicher zu gehen, dass die Räume nicht von Elementen aus anderen Räumen überlappt werden. Auf ähnliche Weise verwaltet sie auch die Sichtbarkeit der Wände der einzelnen Räume, die für die Real-Walking Bedingung nur angezeigt werden, wenn die Nutzer:in in dem Raum ist. Um später vergleichen zu können, wie gut die Einschätzung der Proband:innen war misst, diese Klasse außerdem die von der Proband:in zurückgelegte Strecke. Wenn der letzte Raum des aktuellen Versuchs erreicht wurde, übergibt sie die entsprechenden Daten an den FirebaseHandler, der diese dann an eine Cloud-Datenbank verschickt, sodass der Versuchsdurchgang im Nachhinein ausgewertet werden kann.

RotationGainMechanism Diese Klasse implementiert den Mechanismus, der in der Real-Walking Bedingung bei den jeweiligen Räumen angewandt wird um den genauen Ablauf des Rotation-Gains zu definieren. Er zeigt die Zahlen auf der Tafel im Korridor an, verwertet die Eingabe des Eingabefelds, startet den Rotation-Gain und beendet ihn auch wieder. Er signalisiert dem RoomAndProgressManager wenn der Rotation-Gain vollständig ist. Dies ist der Fall wenn der Trackingspace um 90° gedreht hat, und die Proband:in dann eine richtige Eingabe tätigt. Dies verursacht, dass sich die Tür in den nächsten Raum öffnet. In den anderen Bedingungen ist er dafür verantwortlich dies nach einer zufälligen, geringen, Anzahl richtiger Eingaben in das Nummerneingabefeld zu tun, sodass die Proband:in auch in diesen Bedingungen den selben Ablauf erlebt. Dieser Mechanismus wird in Abschnitt 3.9 genauer ausgeführt.

SceneLoader Beim Betreten des Levels sieht die Proband:in ein virtuelles Eingabefeld, in dass sie die verschiedenen Testbedingungen eingeben kann, um den Versuch zu starten. Der SceneLoader ist dafür zuständig, auf diese Eingabe entsprechend zu reagieren und die entsprechende Bedingung zu laden. Dazu verändert dieser die Attribute der "GenerateLevel"-Instanz und ruft dann ihre "Generate"-Methode auf. Dieser Prozess wird nochmal in Abschnitt 3.11 vertieft.

RotationRedirectorCollision Kleine Hilfsklasse, die erkennt ob der Nutzer sich gerade in dem entsprechenden Bereich befindet, indem der Rotation-Gain aktiv sein soll.

Door Die Türen nutzen dieses Script. Auf ihm kann die Methode "OpenDoor" aufge-

rufen werden, um sie zu öffnen.

Dissolvable Wird von der "Door"-Klasse genutzt, um der Tür die Funktionalität zu geben, fließend zu verschwinden. Dafür nutzt sie einen entsprechend programmierten Shader, der mithilfe einer Wolkentextur die Transparenz des Türmaterials anpasst.

INumberPadListener Die Klassen, die über Eingaben in einem NumberPad informiert werden wollen, können dieses Interface implementieren um zu einem NumberPadListener zu werden. Dies geschieht also nach dem "Observer Pattern" (siehe [9])

NumberPadScript Dieses Script ist eine Komponente eines Nummerneingabefeldes und ist dafür zuständig die Eingabe zu verarbeiten und dann alle angemeldeten Observer über Änderungen zu informieren.

NumberPadButton Verbessert die Knöpfe eines Nummerneingabefeldes, indem es ihnen eine "CoolDown"-Periode gibt, sodass nicht unbeabsichtigt mehrere Eingaben entstehen und indem es einen Soundeffekt abspielt um der Nutzer:in als Feedback zu bestätigen, dass der Knopf gedrückt wurde.

FireBaseHandler Bietet die Methode "PostResult" an, die die Daten des aktuellen Versuchs an eine Cloud-Datenbank verschickt. So werden dann die Anzahl an Räume des Levels, die gelaufene Strecke in Metern, die Fortbewegungsart, und das Datum, inklusive der genauen Uhrzeit des Abschickens gespeichert. Außerdem wird gespeichert ob das Verschicken im Unity-Editor, also zum Testen, geschehen ist, oder ob es wirklich von der Datenbrille aus versendet wurde.

3.3 Hierarchy der Unity-Scene

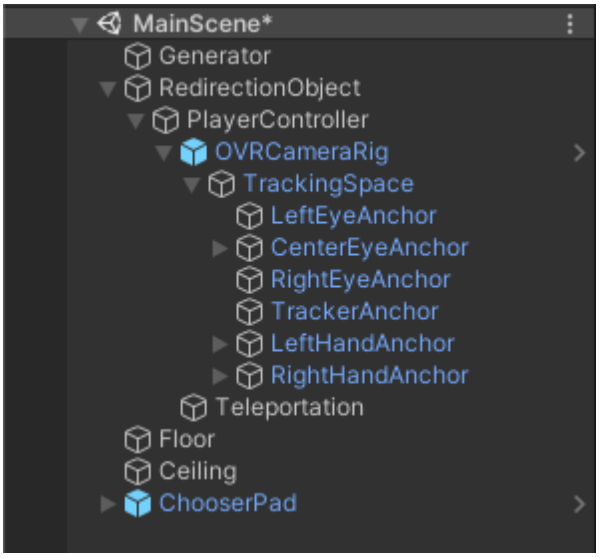

Abbildung 3.2: Die Hierachie der Unity-Szene vor der Laufzeit

Der folgende Abschnitt beschreibt die Objekte in der Unity-Szene und ihre Beziehung untereinander.

Das “Generator”-Objekt existiert um die Scripts, die ihm als Komponenten angehängt sind, in die Szene zu integrieren, sodass diese ausgeführt werden. Darauf folgt das “RedirectionObject”, welches, als effektiver Parent des “OVRCameraRig” später dafür genutzt wird, die Rotation-Gains auszuführen. Zwischen diesen beiden Objekten liegt aber noch der PlayerController. Dieser wird in den beiden Kontrollbedingungen für die Fortbewegungsarten genutzt.

Das “OVRCameraRig” und die hierarchisch untergeordneten Elemente sind ein vom Oculus Integration SDK bereitgestelltes Prefab, es integriert die Funktionalität der virtuellen Umgebung in die Unity-Szene.

“Floor” und “Ceiling” repräsentieren Boden und Decke der Szene. Das zuletzt in Abbildung 3.2 dargestellte Objekt ist das “ChooserPad”. Dies ist ein, den später beschriebenen Nummerneingabefeldern nachempfundener Mechanismus mit dem die Nutzer:in eingeben kann, welche der Testbedingungen sie ausführen möchte. Dies wird genauer in Abschnitt 3.11 erklärt.

Abbildung 3.3 zeigt die Hierarchie der erst nach Levelgenerierung entstandenen “RotGainCorridor”-Objekte, die nicht nur die, später in Abschnitt 4.2 beschriebenen, Korridore, und alle Objekte darin repräsentieren, sondern auch die Wände um die zugehörigen Räume, also alle Objekte, die zu dem jeweiligen Raum gehören.

Abbildung 3.3: Die Unity-Hierarchie der Objekte, die der Szene erst bei der Levelgenerierung während der Laufzeit hinzugefügt wurden

3.4 Design der Szene

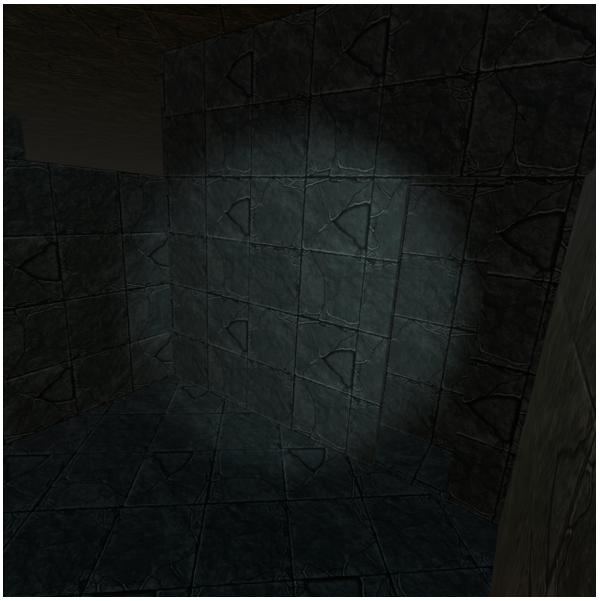

Abbildung 3.4: Ein Raum von innen

Im Folgenden werde ich die Designentscheidungen, die die Optik der Szene bestimmen erläutern. Ein generierter Raum ist in Abbildung 3.4 zu sehen.

Die virtuelle Umgebung, die die Nutzer:in durchschreitet, ist der eines Dungeons nachempfunden. Dies ist in Computerspielen ein häufig eingesetztes Setting, weltbekannte Spiele wie “The Elder Scrolls V: Skyrim”, die “Final Fantasy” Spielreihe, die “The Legend of Zelda” Spielreihe, und besonders sogenannte “Roguelike”-Spiele, wie die “Diablo” Spielreihe, “The Binding of Isaac”, oder “Darkest Dungeon”, beinhalten alle zumindest zum Teil Dungeonartige Level, die den Spieler herausfordern, eine dunkle Umgebung zu erkunden und gegen fiese Gegner zu kämpfen. Obgleich letztere in den hier erzeugten Leveln nicht vorkommen, ist das Ambiente bewusst ein wenig düster gehalten und erinnert optisch an einen Dungeon. Dafür die aus dem Unity-Asset-Store heruntergeladene Textur [29], die die Wände schmückt und ihnen ein rustikales, dungeonartiges Gefühl verleihen, sehr hilfrich.

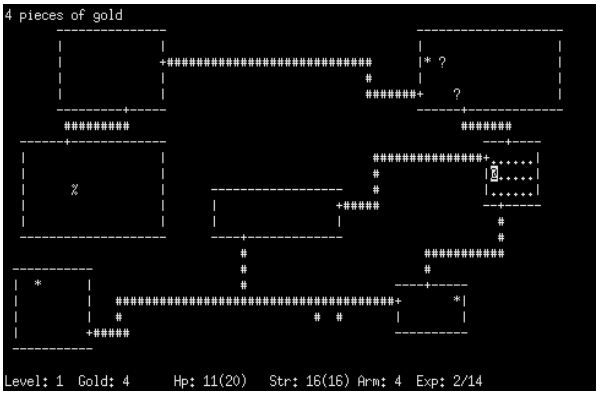

Abbildung 3.5: Screenshot des beliebten Computerspiels “Rogue” (1980) Bildquelle: https://commons.wikimedia.org/wiki/File:Rogue_Screenshot.png

Das Spiel “Rogue”, von 1980, nachdem auch das Genre “Roguelike” benannt wurde, ist ein Meilenstein in der Prozeduralen Levelgeneration gewesen und hat bis heute großen Einfluss auf die Welt der Computerspiele. Auch dabei geht es darum einen Dungeon zu erkunden, auch wenn dieser verständlicherweise graphisch, noch nicht besonders aufbereitet gewesen ist (siehe Abbildung 3.5).

Abgesehen davon bietet die Umgebung eines Dungeons viele Freiheiten, was den Realismus der Welt angeht. Es wird nicht hinterfragt, wieso ein Gebäude so strukturiert sein sollte, dass man die ganze Zeit um die Ecke geht oder wieso die Türen sich öffnen und schließen, in dem sie magisch erscheinen oder sich auflösen.

Um die Erkundungsstimmung noch ein wenig zu verstärken und eine realistische Lichtquelle in die Welt einzubauen trägt die Nutzer:in eine Kopflampe. Dies ist ein einfacher Spotlight mit leicht bläulichem Licht, dessen Parent die Kamera des Spieler:innenavatars ist.

3.5 Virtueller Avatar

Abbildung 3.6: Die Hände des virtuellen Avatars vor einem Nummerneingabefeld

Um die Nutzer:in in der Szene zu repräsentieren und ihr die Möglichkeit zu geben mit dieser zu interagieren kann sie über die Tranformation von Datenbrille und Controller einen virtuellen Avatar steuern. Auf diese Weise ist es ihr möglich, mithilfe der Hände dieses virtuellen Avatars die Knöpfe von Nummerneingabefeldern zu bestätigen und in der später beschriebenen Teleportationsbedingung den Ort auszuwählen, zu dem sie sich teleportieren möchte (siehe Abbildung 3.8).

Die virtuellen Hände folgen der Bewegung und Drehung der echten Hände sogar der Stellung der Finger, von der Nutzer:in solange sie die Oculus Controller auf die richtige Weise hält. Die Hände sind in Abbildung 3.6 zu sehen. Auch mit ihrem Kopf kann die Nutzer:in mit der Szene interagieren und Knöpfe betätigen, wenngleich dies auch eine weniger hilfreiche Funktionalität ist.

Die Funktionalität der virtuellen Avatare wurde vom Oculus SDK bereit gestellt und wurde von mir lediglich in die Szene integriert. Damit virtuelle Objekte innerhalb der Szene dem realen Körper der Nutzer:in folgen wurden diese hierarchisch den dafür bereit gestellten “Anchor”-Objekten unterstellt (siehe Abbildung 3.2). Für die Hände wurden dafür die vom Oculus SDK bereiten gestellten Prefabs genutzt. An den “CenterEyeAnchor” wurde die in Abschnitt 3.4 erwähnte Kopflampe angehangen, sodass diese der Blickrichtung der Nutzer:in folgt.

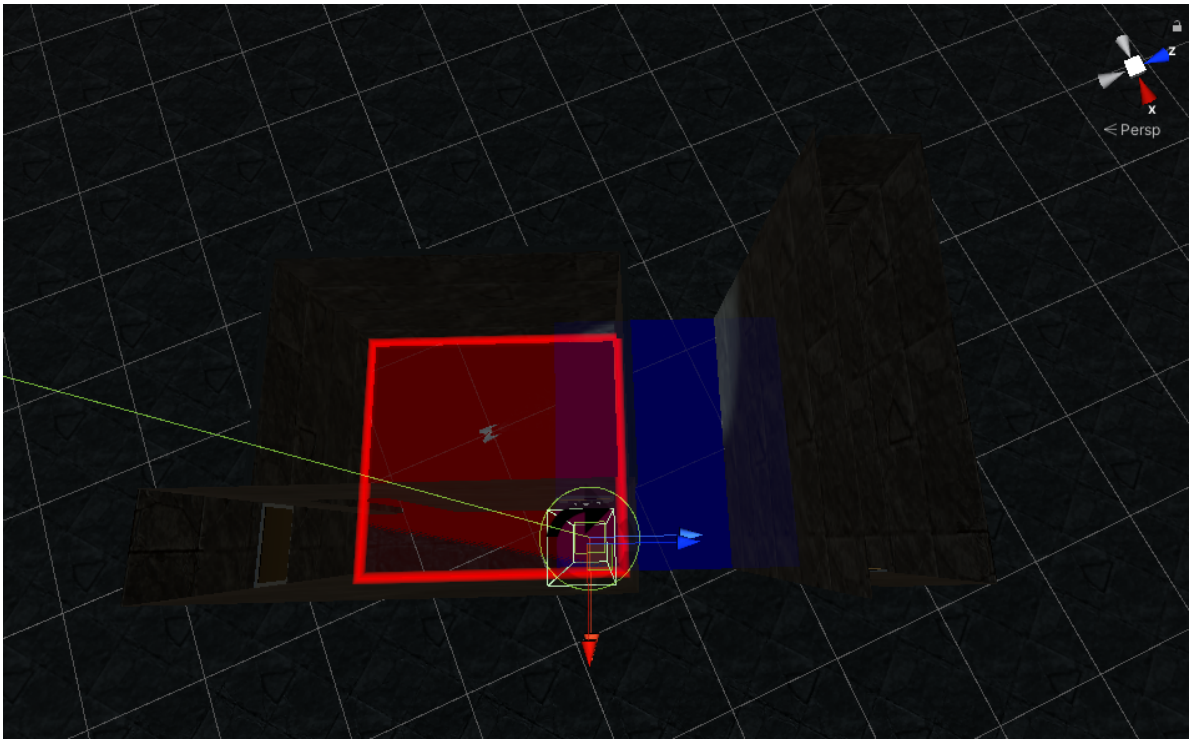

Abbildung 3.7: Screenshot eines instaziierten “RotatinRedirector”-Objekts von oben. Der rote Bereich repräsentiert den Trackingspace bevor der Rotation-Gain ausgeführt wurde, der Blaue den danach. Der grün umrandete Quader stellt den Collider dar, in dem die Nutzer:in sich befinden muss, damit der Rotation-Gain angewendet wird. Die grün nach oben steigende Linie repräsentiert den Punkt $\vec{P}$

3.6 Implementierung der Rotation-Gains

Die Umsetzung der Rotaiongains erfolgt mithilfe des mir zur Verfügung gestellten Unity-Package “Space-Extender” [23]. Dieses stellt das Prefab “RotationRedirector” zur Verfügung. Dieses kann im Code instanziiert werden und ist, nachdem es richtig positioniert wurde und einige Parameter übergeben bekommen hat, dazu in der Lage einen Rotation-Gain auf die Nutzer:in anzuwenden. Dafür benötigt es die Dimensionen des Trackingspaces, den virtuellen Avatar der Nutzer:in und die Koordinaten, des Punktes, um den der Trackingspace rotiert werden soll. Wie das generierte und konfigurierte RotationRedirector-Objekt im Unity-Editor aussieht ist in Abbildung 3.7 zu sehen. Der rote Bereich repräsentiert den innerhalb Trackingspaces begehbaren Bereich vor der Anwendung des Rotation-Gains und der Blaue den innerhalb des Trackingspace begehbaren Bereich nach der vollständigen Ausführung des Rotaiongains. Der RotationRedirector wird, nachdem das alles geschehen ist, für jeden Raum, an die Instanz des RotationGain-Mechanism, die für eben jenen Raum zuständig ist, übergeben. Sobald das Signal, dass die Nutzer:in sich über dem Rotationspunkt ($\vec{P}$) befindet, vom entsprechenden Collider kommt, wird der Prozess des Rotation-Gains am RotationRedirector aktiviert. (Auch

der Collider ist in Abbildung 3.7 abgebildet.) Der Rotation-Gain verstärkt oder verringert die Drehung in jeweilige Richtung, um 10%. Damit liegt der Rotation-Gain noch innerhalb der in [24] gefundenen Grenzen der Wahrnehmbarkeit (Detection-Thresholds), und sollte somit nicht von Nutzer:innen erkennbar sein. Des Weiteren sind die Rotation-Gains bei dieser Implementierung abhängig von der Kopf-Drehgeschwindigkeit, bei langsamen Kopfdrehungen wird ein schwächerer Effekt angewandt.

3.7 Implementierung der Impossible-Spaces

Die Wände der Räume werden für die Real-Walking Studienbedingung, wie später in Unterabschnitt 4.3.1 ausgeführt, auf eine Weise generiert, die in der realen Welt nicht möglich wäre, die Räume überlagern sich an den Grenzbereichen. Sie werden also zu Impossible-Spaces. Das Ein- und Ausblenden der entsprechenden Wände wird von der "RoomAndProgressManager"-Klasse übernommen.

3.8 Fortbewegungsarten

Im folgenden werde ich beschreiben wie die unterschiedlichen Fortbewegungsarten implementiert wurden.

3.8.1 Joystick-Steuerung

Der Joystick des rechten Controllers bestimmt bei dieser Fortbewegungsbedingung die Yaw-Rotation des Avatars, während der des linken Controllers die Bewegung ermöglicht. Den linken Joystick nach oben zu schieben beschleunigt den Avatar in die aktuelle Blickrichtung.

Die Joystick-Fortbewegung wurde mithilfe einiger, vom Oculus Integration SDK gelieferter Scripts, hauptsächlich dem "OVRPlayerController" umgesetzt. Dieser befindet sich dann auf dem in Abschnitt 3.3 beschriebenen PlayerController und steuert so diesen und dessen Kindelemente, so auch den virtuellen Avatar der Nutzer:in.

Um das Aufkommen der Simulatorkrankheit möglichst gering zu halten werden Drehungen nicht wie bei Joystick-Steuerung, beispielsweise in Konsolenspielen üblich, fließend auf die Kopfdrehungen übersetzt, sondern werden in festen 45°-Intervallen angewandt (siehe [8]).

Die Beschleunigung des Nutzer:innen-Avatars liegt bei $0,1$. Da die Fortbewegung über den OVRPlayerController geregelt wird, wird hier keine feste Geschwindigkeit, sondern eben nur die Beschleunigung festgelegt.

3.8.2 Teleportierung

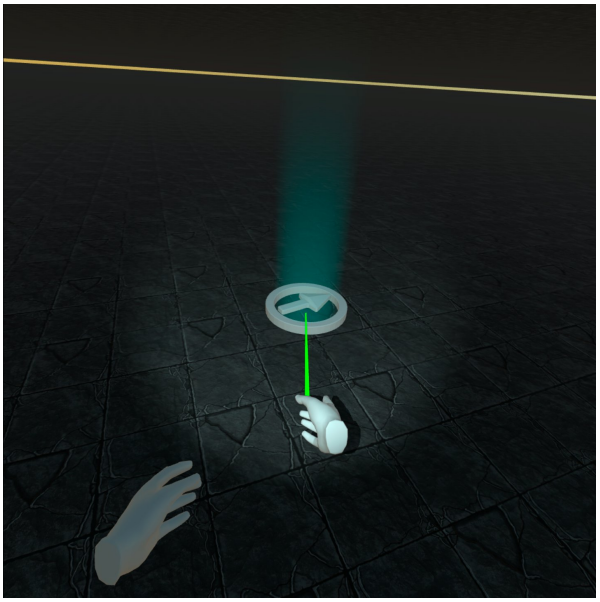

Abbildung 3.8: Der Teleportationscursor, mit dem die Nutzer:in kurz vor der Teleportation ihr Ziel markieren kann. Der Pfeil innerhalb des Rings repräsentiert die Blickrichtung nach der Teleportation

Die Teleportierung funktioniert indem die Nutzer:innen mit der rechten Hand ihres Avatars während sie die *A*-Taste auf dem rechten Controller gedrückt halten, auf die Stelle am Boden zielen, zu der sie sich bewegen möchten. Währenddessen werden ein grüner Laserstrahl, der aus der rechten Hand zielt und der Teleportationscursor eingeblendet (siehe Abbildung 3.8). Mit dem Joystick des linken Controllers kann die Blickrichtung nach der Teleportation eingestellt werden, diese wird auch durch den Pfeil im Teleportationscursor repäsentiert. Wenn die Nutzer:in nun den Zeigefinger-Trigger ihres rechten Controller bedient, wird die Teleportation ausgeführt.

Auch diese Fortbewegungsart wird mithilfe einiger, mit dem Oculus Integration SDK gelieferter Scripts, die auf dem extra für diesen Zweck platzierten Kind-Objekt des PlayerControllers “Teleportation” und auf dem PlayerController-Objekt als Komponenten liegen, implementiert. Diese sind “PlayerController (Simple Capsule with Stick Movement)”, “CharacterController”, “Locomotion Teleport”, “Teleport Input Handler Touch”, “Teleport Target Handler Physical”, “Teleport Aim Handler Laser”, “Teleport Aim Visual Laser”, “Teleport Orientation Handler Thumbstick”, und “Teleport Transition Instant”. Des Weiteren werden zwei Prefabs, nämlich: “TeleportDestination” und“,”TeleportLaser benötigt.

3.8.3 Real-Walking

Für die Real-Walking Fortbewegungsart bedarf es keiner besonderen Implementierung, es ist lediglich wichtig, dass die Einstellung “Tracking Origin Type” im, auf dem “OVR-CameraRig”-Objekt liegenden, Script “OVR Manager”, auf “Stage” gesetzt wird. Dies unterdrückt den Zentrierungsmechanismus, der standardmäßig ausgeführt wird wenn der “OculusButton” gehalten wird. Der Zentrierungsmechanismus ist für die anderen beiden Fortbewegungsarten wichtig, um die Position des Nutzer:innen-Avatars wieder mit der wahren Position der Nutzer:in abzugleichen.

3.9 Der Rotation-Gain-Anreiz-Mechanismus

Damit der Rotation-Gain zur Wirkung kommt muss sich die Nutzer:in ungefähr auf den Punkt $\vec{P}$ stellen und ihren Kopf um die Yaw-Achse drehen. Für das intuitivere Gefühl und um die Aufmerksamkeit nicht so konkret auf den Rotation-Gain zu lenken, habe ich mich dagegen entschieden den Punkt $\vec{P}$ zu markieren und die Nutzer:innen zu bitten dort ihren Kopf zu drehen, sondern habe mir einen Vorwand ausgedacht, wieso es zum einen notwendig ist an der Stelle zu stehen und zum anderen, wieso sie ihren Kopf drehen müssen.

Abbildung 3.9: Ein Nummerneingabefeld. Mit der # Taste wird eine Eingabe bestätigt, die c-Taste löscht die aktuelle Eingabe

Um diesen Effekt zu erzielen wird ein Nummerneingabefeld (siehe Abbildung 3.9), an der Korridorinnenwand in Richtung des Raumes aus dem die Nutzer:in gerade kommt,

genau vor die Position $\vec{P}$ plaziert. In dieses Eingabefeld sollen dann mehrere zweistellige Nummern, die am anderen Ende des Korridors, auf einer digitalen Tafel angezeigt werden, eingegeben werden, bis die Tür sich öffnet. Auf der Tafel wird nur eine zweistellige Nummer zur Zeit angezeigt. Dies sorgt dann sowohl dafür, dass die Nutzer:in auf den Punkt $\vec{P}$ stellen, um möglichst einfach die Nummern eingeben zu können, als auch dafür, dass die Nutzer:in ihren Kopf wiederholt um die Yaw-Achse dreht, weil sie abwechselnd ablesen muss, welche Zahlen auf der Tafel hinter ihr angezeigt werden und die Zahlen in das Eingabefeld schräg vor ihr eingeben muss. Dies sorgt für einen intuitiven Grund den Kopf zu drehen und somit dafür den Rotation-Gain zu nutzen. Wenn der Fortschritt des Rotation-Gains abgeschlossen ist und ein weiteres Mal die korrekte Nummer in das Eingabefeld eingegeben wurde, öffnet sich die Tür vor der Nutzer:in (also die "Seitentür") sodass die Nutzer:in in den nächsten Raum gehen kann. Da nun der Trackingspace genau den nächsten Raum umgibt, ist es wichtig, dass die Nutzer:in nicht versucht, zurück in den vorherigen Raum zu gehen. Aus diesem Grund schließt sich hinter der Nutzer:in eine vorher noch nicht zu erkennenende Tür.

Im Programm-Code findet sich dieser ganze Vorgang in der Klasse "RotationGainMechanism". Dieser wird für jeden Raum erstellt und auch dem Objekt des Raumes als Komponente hinzugefügt. Eine Instanz des RotationGainMechanisms ist also immer für den Raum zuständig, auf dem er sich befindet. Mit der "Init"-Funktion wird das Objekt direkt nach der Erstellung initialisiert, hier meldet sich die Instanz dann auch in der zugehörigen Eingabefeld-Klasse ("NumPadScript") als Observer an. Sobald sich die Nutzer:in dann in den um die Position $\vec{P}$ herum platzierten Collider begibt, wird mit der Funktion "StartMechanism" die Prozedur gestartet. Zunächst wird eine neue zweistellige Zahl mithilfe des (von Unity bereitgestellten) Pseudozufallgenerators erstellt, dann wird diese Zahl auf der dem aktuellen Raum zugehörigen Tafel angezeigt. Sobald die Bestätigungstaste des Eingabefelds gedrückt wurde ("#"), wird überprüft ob der Fortschritt des Rotation-Gains schon bei 100% ist. Falls dem nicht so ist, beginnt der Vorgang von vorne und es wird eine neue Zahl erstellt. Wenn der Rotation-Gain vollständig abgeschlossen ist (also eine 90° Drehung stattgefunden hat) wird dem "RoomAndProgressManager" mitgeteilt dass der aktuelle Raum abgeschlossen ist und die Tür öffnet sich.

In den Kontrollbedingungen wird kein Rotation-Gain angewandt. Dennoch müssen die Proband:innen auch hier Zahlen in das Eingabefeld eingeben. In diesen Szenarien wird die Türöffnung nicht von der Vervollständigung des Rotation-Gains abhängig gemacht, sondern wird eine Zufallszahl zwischen 4 und 8 bestimmt und mitgezählt, wie oft Zahlen bestätigt wurden. Wenn die Zufallszahl erreicht wird, öffnet sich die Tür. Durch Erfahrungen beim Testen hat sich gezeigt dass es etwa zwischen 4 und 8 Eingaben bedarf, bevor der Rotation-Gain abgeschlossen ist.

Abbildung 3.10: Links: Eine sich gerade mit einem Auflösungseffekt öffnende Seitentür. Rechts: Die nun verschlossene Tür, die verhindert, dass die Nutzer:in nach der Ausführung des Rotation-Gains in den vorherigen Raum zurück geht

3.10 Seitentür Mechanismen

Um die Nutzer:in daran zu hindern in Bereiche zu gehen, die ihr noch nicht zugänglich sind, weil beispielsweise der Rotation-Gain noch nicht abgeschlossen ist, wird ihr dies durch eine verschlossene Tür suggeriert. Nach abgeschlossenem Rotation-Gain öffnet die Tür sich mit einem auflösenden optischen Effekt (siehe Abbildung 3.10, links) und einem Erfolg andeutenden Geräusch. Damit sie nun aber nicht auf die Idee kommt, in den vorherigen Raum zurück zu gehen (dieser läge nun außerhalb des begehbaren Bereichs) wird hinter ihr eine weitere Tür mit dem selben Effekt verschlossen (siehe Abbildung 3.10 rechts).

Der visuelle Effekt entsteht durch einen Fragmentshader, der auf der Tür liegt. Dieser kontrolliert die Durchsichtigkeit der Bereiche der Tür mithilfe einer animierbaren Variable und einer Wolkentextur. Die dunklen Bereiche der Wolkentextur werden bei ansteigen der Variable zuerst ausgeblendet, bis irgendwann die ganze Tür unsichtbar geworden ist. Bei der schließenden Tür geschieht dies genau umgekehrt.

Um zu verhindern, dass in andere Räume hinragende Korridore betreten werden können, wurden auch die Haupttüren dieser Korridore mit dem selben Effekt verschließbar gemacht. Diese öffnen sich jedoch immer direkt beim Betreten des Raumes, in dem der Korridor steht.

Abbildung 3.11: Das Eingabefeld, mit dem die verschiedenen Versuchsbedingungen ausgewählt werden können. Die c-Taste löscht die aktuelle Eingabe, mit der #-Taste wird das aktuell ausgewählte Level gestartet

3.11 Bedingungsauwahl und Laden der Szene

Damit die Versuchspersonen der später vorgestellten Studie die Level mit den unterschiedlichen Bedingungen ausführen können, gibt es ein Menü, in dem diese ausgewählt werden können. Darin befindet sich nur der Boden, die Decke und das Bedingungsauswahlfeld (siehe Abbildung 3.11). Auf diesem kann die Nutzer:in auf einem 3×3 Eingabefeld auswählen, welche Versuchsbedingung sie starten möchte. In der späteren Studie wird vorgegeben, welche die Proband:in auswählen soll. Die einzelnen Bedingungen sind folgendermaßen kodiert. A steht für die Real-Walking Fortbewegungsart-Bedingungen, B für Joystick-Steuerung, C für Teleportation, 1 für eine Raumanzahl von 3 Räumen, 2 für 6 und 3 für 9 Räume. Die Auswahl erfolgt per Tastendruck, allerdings wird das Level erst generiert und gestartet, sobald anschließend die Bestätigungstaste # gedrückt wird. Diese Kodierung ist den Versuchspersonen nicht bekannt.

Das ganze funktioniert dann so, dass die Instanz der “SceneLoader”-Klasse, die auf dem “Generator”-Objekt liegt, eine Instanz auf das Auswahlfeld und auf die “GenerateLevel”-Instanz hält. Sobald die Bestätigungstaste gedrückt wurde, werden die entsprechenden Einstellungen daran vorgenommen, einige VR-Einstellungen getätigt (beispielsweise das Deaktivieren der Reset-Möglichkeit, die bei der Real-Walking Bedingung zu Problemen führen könnte), die Fortbewegungsart festgelegt, indem die entsprechenden Unity-Komponenten, die die Joysticksteuerung oder die Teleportation ermöglichen aktiviert, oder deaktiviert werden und zu guter letzt die Teleportierung der Nutzer:in in den ersten Raum des Levels getätigt.

3.12 Messmechanismen und Datenspeicherung

Um an späterer Stelle einschätzen zu können, wie gut die Proband:innen die zurückgelegte Strecke geschätzt haben, ist es erforderlich diese zu messen. Zusätzlich dazu ist es auch notwendig diese Daten zu speichern. Im Folgenden werde ich erläutern, wie dies implementiert ist.

Da es nicht darum geht, die Luftlinie zwischen Start und Ende des Levels zu messen, sondern ihre konkret zurückgelegte Distanz zu messen, ist diese Aufgabe nicht gänzlich trivial. Würde man einfach in jedem Tick die Kopfposition des virtuellen Avatars der Proband:in speichern, die Distanz zum vorherigen Punkt berechnen und aufsummieren, wäre das Ergebnis unerwartet groß, und insoweit inkorrekt, als dass damit dann nicht mehr die zurückgelegte Distanz, wie sie wahrscheinlich intuitiv interpretiert werden würde gemeint wäre. Da nun jede Millimeterbewegung des Kopfes aufsummiert werden würde, würde eine einfache Kopfdrehung die zurückgelegte Strecke schon um einen großen Teil erhöhen. Dies würde dazu führen, dass die Messung der zurückgelegten Distanz nicht mehr intuitiv einzuschätzen wäre.

Um dem entgegenzuwirken habe ich die Auflösung dieser Messung deutlich veringert, indem nicht mehr jeder Tick, diesen Prozess durchläuft, sondern nur ein Tick etwa alle 2 Sekunden. So werden deutlich grobere Bewegungen gemessen und die gemessenen Ergebnisse passen deutlich besser zu der intuitiven Einschätzung der Entfernung. Es ist aber dennoch zu erwähnen, dass durch die verlorenen Details auch eine gewisse Strecke verloren geht, beispielsweise wenn die Nutzer:in zwischen den zwei Ticks eine Kurve gegangen ist. Es ist dennoch unumgänglich eine solche, oder ähnliche Messauflösungsveringerung vorzunehmen. Andernfalls wäre zusätzlich davon auszugehen, dass die Real-Walking Bedingung aufgrund von natürlichen Schwankungen beim Gehen deutlich längere gemessene Strecken produzieren würde, als die virtuellen Fortbewegungsarten.

All dies ist im “RoomAndProgressManager” implementiert. Hier gibt es eine “StartTimeMeasure”-Methode, die zum Start der Bedingung vom “SceneLoader” aufgerufen wird und eine “AddDistance”-Methode, die in der von der “Update”-Methode alle 150 Frames einmal aufgerufen wird. Diese bestimmt dann euklidisch die Distanz zwischen der aktuellen Position der Nutzer:in und der zuletzt gemessenen und addiert diese auf eine Kummulationsvariable.

Sobald das Level der Versuchsbedingung abgeschlossen ist, wird diese Kummulationsvariable an den “FirebaseHandler” weitergeleitet, der nun die Daten der aktuellen Versuchsbedingung an die private Datenbank sendet um die Daten für die spätere Auswertung zu speichern.

Kapitel 4

Levelgenerierung

Dieses Kapitel beschreibt, wie genau die Generierung eines mit der Methode, die diese Arbeit vorstellt, generierten Levels funktioniert. Nacheinander werde ich die grundlegenden Ideen hinter der Levelgenerierung erörtern und dabei jeweils genauer auf ihre Implementierung eingehen. Anschließend werde ich den Algorithmus, der die unterschiedlichen Codeblöcke nutzt um die Levelstruktur zu generieren beschreiben und vorstellen. Zuerst bespreche ich, wie die Räume angeordnet werden müssen damit, sie durch den Rotation-Gain verbunden werden können. Dann stell ich vor, wie die Korridore, die in den Räumen generiert werden um der Nutzer:in einen klaren Weg voran zu präsentieren und den Übergang zwischen den Räumen intuitiver zu gestalten, erstellt werden. Als nächstes beschreibe ich die Vorgänge, die für die Generierung der Wände der Räume verantwortlich sind und erkläre wie die entsprechenden Berechnungen stattfinden. Anschließend werde ich beschreiben, an welchen Stellschrauben manipuliert werden kann um heterogene Ergebnisse bei der Levelgenerierung zu bekommen, erst hier können (Pseudo-)Zufallsfaktoren eine Rolle spielen. Mit der Erklärung der Zusammensetzung des Algorithmus schließe ich dieses Kapitel ab.

4.1 Raumplatzierung

Um zu verstehen, wie die Ecken eines Raumes berrechnet werden, ist es zunächst essentiell, die Grundidee hinter der Raumanordnung zu verstehen.

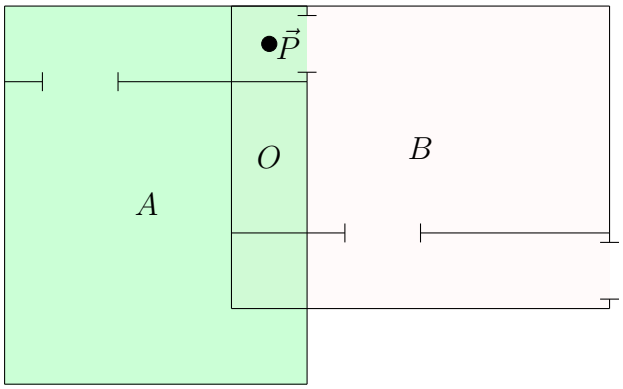

Abbildung 4.1: Darstellung davon, wie die Räume für den antizipierten Effekt angeordnet sein müssen. Wenn an Punkt $\vec{P}$ ein Rotation-Gain über 90° ausgeführt wird liegt der Trackingspace vor der Ausführung des Rotation-Gains über dem Bereich A und danach über dem Bereich B. O bezeichnet den Bereich, bei dem sich die beiden Räume überlagern

4.1.1 Grundidee

Wenn ein Rotation-Gain angewandt wird, verdrehen sich virtuelle und reale Realität umeinander, um einen konkreten Drehpunkt. In der Regel stellt dieser die Position der Nutzer:in dar, damit der Effekt allerdings genau plan- und vorhersagbar ist, findet die Drehung der Umgebung nicht um den Kopf der Nutzer:in, sondern um einen bestimmten vordefinierten Punkt $\vec{P}$ statt.

Wenn diese Position $\vec{P}$ sich in einer Ecke des (rechteckigen) Trackingspace-Raumes befindet, zudem genau gleich weit von beiden Wänden, die diese Ecke bilden, entfernt ist und dann ein gerichteter Rotation-Gain angewandt wird, bis genau 90° Verdrehung erreicht sind, dann steht der aktuelle begehbare Bereich in der virtuellen Umgebung B, dem vorherigen begehbaren virtuellen Bereich, der ebenfalls durch den Trackingspace definiert war A, genau orthogonal gegenüber.

Vorher hat der Bereich A genau den virtuellen Bereich innerhalb des Trackingspaces dargestellt. Die beiden Areale A und B überlappen sich zwar in einem gewissen Bereich, in dem die Nutzer:in nach Beendigung des Rotation-Gains dann auch gerade steht O, aber ein Großteil von A ist nun Teil des Bereiches der virtuellen Umgebung geworden, der, begrenzt durch den Trackingspace, nicht zugänglich ist. Auf diese Weise ist also für die Nutzer:in ein neuer Bereich der virtuellen Umgebung begehbar geworden. Wenn der Rotation-Gain subtil genug angewandt wurde hat die Nutzer:in (idealerweise) nicht bewusst wahrgenommen, dass die Welten sich verdreht haben. Wenn sie anschließend geradeaus geht, entsteht der Eindruck, dass nun ein weiterer Teil der Welt zugänglich geworden ist.

Ziel des Algorithmus ist es eine Reihe von aufeinander folgenden Räumen zu generieren,

sodass die Nutzer:in sich durch ein Dungeon-artiges Level bewegen kann um vom ersten zum letzten Raum zu gelangen. Die Grundidee hinter der Raumgenerierung ist es also, dass genau die beiden Areale A und B von Wänden umgeben werden. In Abbildung 4.1 wird ersichtlich, wie die beiden Räume zueinander stehen müssen, um diesen Effekt zu ermöglichen.

Um dann also von Raum A zu Raum B zu gelangen, muss die Nutzer:in sich an der Position $\vec{P}$ oft genug drehen und dann muss ein Rotation-Gain bis zu 90° in die jeweilige Richtung angewandt werden. Bei dem Beispiel in Abbildung 4.1 muss der Rotation-Gain bei Yaw-Kopfdrehungen nach links positiv und nach rechts negativ verlaufen um den gewünschten Effekt möglichst effizient zu erzielen.

Die Nutzer:in kann nun geradeaus gehen, da sie sich innerhalb des realen Trackingspaces um 90° gedreht hat, also nicht mehr so steht, dass sie aus dem Trackingspace hinaus gehen würde, würde sie geradeaus gehen. Wenn sie diese Drehung aber nicht wahrgenommen hat, entsteht die Illusion sie könnte geradeaus, über die Grenzen des Trackingspaces hinaus gehen. Der reale Trackingspace umschließt bei der hier vorgestellten Methode also immer genau den Raum, in dem sich die Nutzer:in gerade befindet.

4.1.2 Berechnung der Ecken

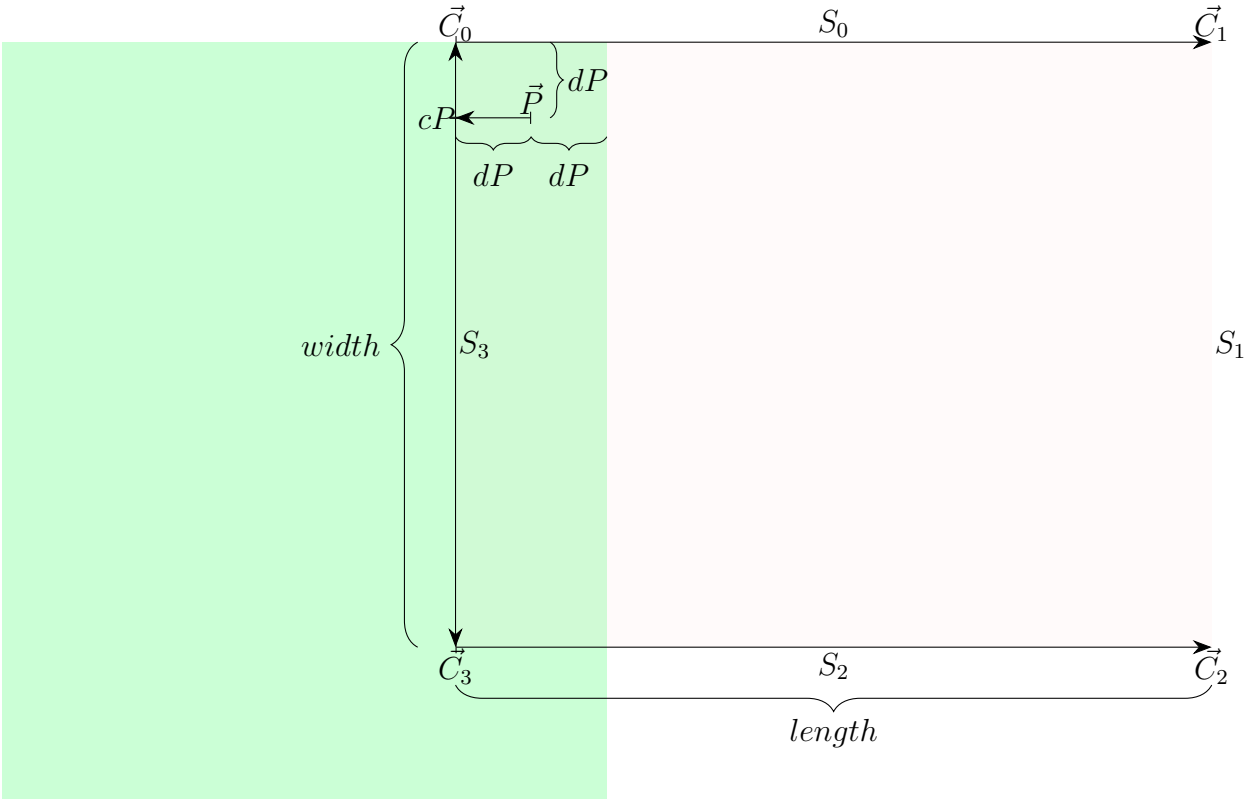

Abbildung 4.2: Die Pfeile repräsentieren die zur Eckenberechnung genutzten Vektoren. Mit $\vec{C_0}, ..., \vec{C_3}$ werden die Ecken bezeichnet, mit $S_0, ..., S_3$ die Seiten des Raumes

Der Effekt, dass die Räume in denen sich die Nutzer:innen gerade befinden, immer genau den Trackingspace darstellen, wird zu großen Teilen durch die besondere Art der Raumgenerierung ermöglicht. Diese möchte ich im Folgenden vorstellen.

Zunächst müssen die Koordinaten der Ecken innerhalb der virtuellen Umgebung bestimmt werden. Für den ersten Raum ist es nicht schwierig, die virtuellen Koordinaten der Ecken der Trackingspace-Boundaries werden vom Oculus Integration SDK geliefert. In der “GeneratorOculusInterface” Klasse werden diese zu Beginn der Levelgenerierung abgefragt und der “GenerateLevel”-Klasse zur Verfügung gestellt. Um allerdings die Raumecken der folgenden Räume zu bestimmen, müssen einige Punkte, Längen und Vektoren bereits definiert oder errechnet worden sein. Wie diese zustande kommen, wird im darauf folgenden Absatz erläutert. Für alle Räume, außer dem ersten berechnen, sich die Eckkoordinaten dann folgendermaßen.

$$c\vec{P} = \vec{P} + (-1) * s\hat{D}D * dP \tag{4.1}$$

$$\tag{4.2}$$

$$\vec{C_0} = c\vec{P} + (-1) * b\hat{t}F * dP \tag{4.3}$$

$$\vec{C_1} = \vec{C_0} + s\hat{D}D * length \tag{4.4}$$

$$\vec{C_2} = \vec{C_3} + s\hat{D}D * length \tag{4.5}$$

$$\vec{C_3} = c\vec{P} + b\hat{t}F * (width - dP) \tag{4.6}$$

Auch zu sehen ist diese Berechnung in dem in der Abbildung 4.3 gezeigen Quelltext. Wie die hier genutzen Variablen $c\vec{P}$, *width* und *length* definiert sind wird aus Abbildung 4.2 ersichtlich. $c\vec{P}$ steht dabei für den Punkt hinter $\vec{P}$, der genau eine dP-Länge in Richtung des alten Raumes liegt. $s\hat{D}D$ (sideDoorDirection) ist ein normalisierter Richtungsvektor, der die Richtung des Korridors des letzen Raumes erweitert. Also aus dem vorherigen in den aktuellen Raum hineinzeigt. $b\hat{t}F$ (backToFront) ist auch ein normalisierter Richtungsvektor, der orthogonal zu $s\hat{D}D$ steht. Er zeigt von der hinteren Wand des Korridors des vorherigen Raumes zur vorderen Wand. Diese Richtungsvektoren werden vor der Eckenberechnung errechnet indem die Vektoren, die die entsprechenden Ecken des schon bestehenden Raumes darstellen, voneinander subtrahiert werden um die gemeinten Richtungen in Form von Vektoren zu errechnen, bevor sie dann normalisiert werden.

width beschreibt die Breite der Räume, während *length* die Länge beschreibt. Diese beiden Variablen hängen von den Maßen des ersten Raumes ab, bei dem die Eckkoordinaten ja den Trackingspace der Nutzer:in abbilden. Es ist noch wichtig zu erwähnen, dass die in der eben erwähnten Abbildung gezeigte Folge von Ecken und Seiten nicht genau spiegelverkehrt ist, wenn der Raum in die andere Richtung generiert würde (siehe Abschnitt 4.4), sondern die Reihenfolge der Ecken genau invertiert läuft. Das Ganze ist also nicht nur vertikal, sondern auch horizontal gespiegelt zu der Darstellung in Abbildung 4.2.

Listing 4.1: calculateNewCorners.cs

```
private Vector3[] CalculateNewCorners(Vector3 outOfDoor,
   Vector3 backToFront, Vector3 rotationPoint, bool
   sideDoorIsRight, bool corridorOnWidthSide)
{
    //depending on whether the corridor is on the width
       side or on the depth side this changes
    float xLength = corridorOnWidthSide ? roomWidth :
       roomDepth;
    float yLength = corridorOnWidthSide ? roomDepth :
       roomWidth;

    float remainingX = xLength - corridorDepth;

    Vector3 wallBehindRotationPoint = rotationPoint +
       Vector3.Normalize(outOfDoor) * (corridorDepth / 2) *
       -1;

    Vector3 backFirstCorner = wallBehindRotationPoint +
       Vector3.Normalize(backToFront) * corridorDepth / 2 *
       -1; // toBack (back as in: the direction of the old
       corridor)

    Vector3 frontFirstCorner = wallBehindRotationPoint +
       Vector3.Normalize(backToFront) * corridorDepth / 2 +
       Vector3.Normalize(backToFront) * remainingX; //first
       as in: nearer to the outdoor

    Vector3 backSecondCorner = backFirstCorner +
       Vector3.Normalize(outOfDoor) * yLength;

    Vector3 frontSecondCorner = frontFirstCorner +
       Vector3.Normalize(outOfDoor) * yLength;

    Vector3[] output = new Vector3[]
    {
        backFirstCorner,
        backSecondCorner,
        frontSecondCorner,
        frontFirstCorner,
    };

    if (!sideDoorIsRight)
    {
        Array.Reverse(output);
    }

    return output;
}
```


Abbildung 4.3: Funktion zur Berechnung der virtuellen Eckkoordinaten des nächsten Raumes

4.2 Korridore

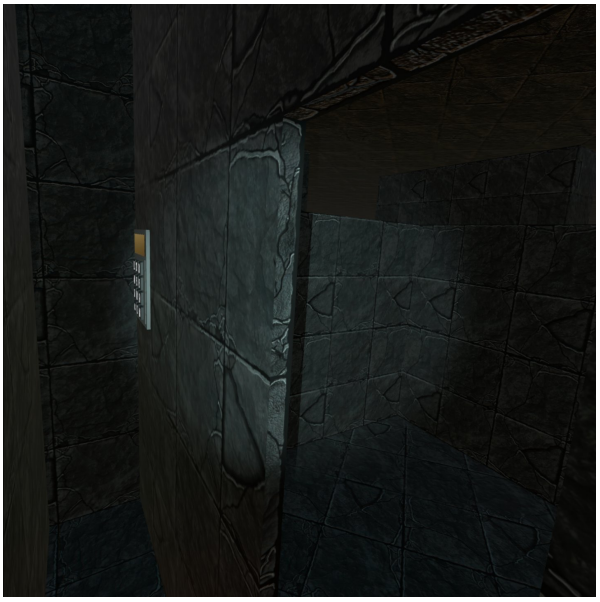

Abbildung 4.4: Ein Korridor von innen. Links im Bild, am Ende des Ganges befindet sich die Seitentür, an der Wand daneben das Nummerneingabefeld. Rechts im Bild lässt sich aus der Tür hinaus in den Raum blicken

Damit die Nutzer:in versteht, wie sie in den nächsten Raum kommt, sind die Räume so gestaltet, dass es einen offensichtlichen Weg voran gibt. Um dies zu erreichen, wird in die Räume jeweils ein Korridor, also ein kleiner Flur, der auf die Tür zum nächsten Raum zuläuft, generiert. Wie ein solcher Korridor von innen aussieht ist in Abbildung 4.4 zu sehen. In der hier vorgestellten Implementierung der Levelgenerierungmethode wird der Korridor für jeden Raum generiert, bevor mit der Eckenberechnung für den nächsten Raum begonnen wird, und wie in Abschnitt 4.4 beschrieben wird, hängt diese sogar von dem Korridor ab. Sie sind also wichtiger Bestandteil dieser Implementierung, die Levelgenerierungmethode ließe sich aber auch ohne sie realisieren. In diesem Abschnitt wird die Generierung dieser Korridore für ihre jeweiligen Räume erläutert.

Zunächst wird eine Kante des Grundecks des Raumes ausgewählt, an der ein Korridor erstellt wird (diese wird pseudozufällig ausgewählt, siehe Abschnitt 4.4). Er kann sowohl an einer kürzeren Kante, als auch an einer längeren Kante des Grundecks des Raumes erstellt werden, nur nicht an der Seite aus der das Ende des Korridors des vorherigen Raums herausragt, da sich die beiden Korridore sonst kreuzen würden.

Der Korridor hat zwei Türen: Eine “Haupttür”, durch die die Nutzer:in in den Korridor hineinkommt und eine “Seitentür”, die anfangs noch verschlossen ist und in den nächsten Raum führt.

In diesem Korridor werden dann Elemente platziert, die gemeinsam für einen Mechanismus sorgen, der es der Nutzer:in nahelegt, sich genug um die Yaw-Achse zu drehen, dass

die wirkliche und die virtuelle Realität sich dank des Rotation-Gains genug umeinander drehen, dass der nächste Raum nun mit dem realen Tracking-Space übereinstimmt. Dieser Mechanismus wurde in Abschnitt 3.9 genauer beschrieben.

4.2.1 Korridor-Meshgenerierung

Um einen solchen Korridor zu generieren, müssen zunächst die Koordinaten der für das Korridor-Mesh genutzten Knotenpunkte (Vertices) bestimmt werden. Zudem müssen dann entsprechende Flächen (Faces) gespannt und für jeden Knotenpunkt dann auch die UV-Koordinaten errechnet werden, sodass die Korridore texturiert werden können. All dies geschieht in der "RoomGenerator"-Klasse.

Ein Korridor besteht aus einem Boden, einer Frontalwand, zwei Seitenwänden und einer Rückwand. Zudem sind in der Frontalwand und einer der Seitenwände Türen eingelassen. Bevor die Meshgenerierung stattfinden kann, muss zunächst die Türposition auf der Frontalachse, D, bestimmt werden. Dabei handelt es sich um einen Wert zwischen 0 und 1, der für jeden Raum pseudo-zufällig generiert wird (siehe Abschnitt 4.4). Mit diesem Wert wird die Frontal-Türposition bestimmt. Je geringer er ist desto weiter links liegt die Haupttür in der Frontalwand. Ist der Wert geringer als 0,5, befindet sich die Seitentür in der rechten Seitenwand, sonst in der linken, sodass diese immer auf der gegenüberliegenden Seite der Haupttür liegt.

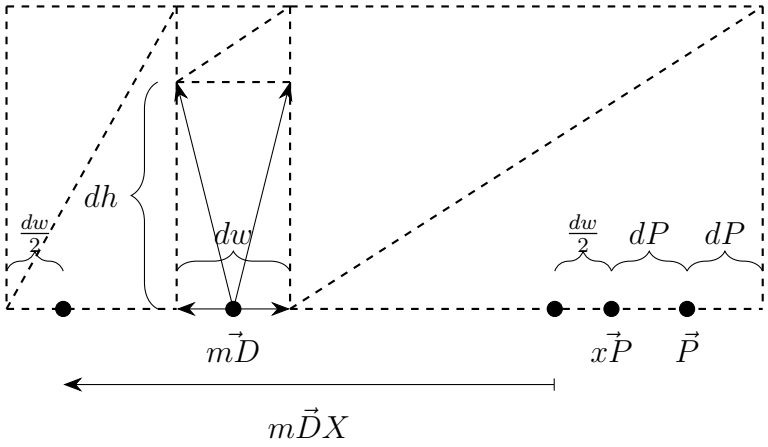

Abbildung 4.5: Frontalansicht von der Struktur der Vorderseite eines Korridors

Es gibt zwei Möglichkeiten der Struktur einer Wand, abhängig davon, ob in ihr eine Tür eingelassen ist oder nicht. Der simplere Fall trifft zu, wenn in der Wand keine Tür liegt, dann besteht die Wand aus einem Quader, also aus 8 Vertices. Die Wandstruktur der Tür-beinhaltenden Alternative wird in Abbildung 4.5 veranschaulicht. Dabei ist allerdings zu beachten, dass nur die Frontalansicht auf eine solche Wand gezeigt ist. Die Innenseite derselben Wand hat die selbe Struktur. Zudem sind die frontalen Vertices der

Tür jeweils mit ihren inneren Gegenspielern durch Faces verbunden, sodass man beim d Durchschreiten der Tür nicht ins Innere der Wand blicken kann.

Vertices Das Generieren eines solchen Korridors beschreibe ich im Folgenden. Gegeben sind die Eckkoordinaten des Raumes, in den der Korridor platziert werden soll ($\vec{C_0}, ..., \vec{C_3}$, diese werden wie in Unterabschnitt 4.1.2 berechnet), die Seite des Raumes, an der der Korridor liegen soll (S_s) und die Türposition D, welche mithilfe eines Pseudo-Zufallsalgorithmus ermittelt werden (siehe Abschnitt 4.4). Dazu noch einige Werte, die im vorhinein von der Gestalter:in der Level vordefiniert werden: Die Tiefe des Korridors d, durch die sich auch der in Unterabschnitt 4.1.2 besprochene Wert dP, also der Abstand des Punktes $\vec{P}$ zu den Wänden der Ecke des Raumes ergibt: $dP = \frac{d}{2}$, die Höhe h und Dicke der Wände w sowie die Höhe (dh) und Breite (dw) der Tür.

Als erstes werden die unteren, äußeren Punkte berechnet. ($\vec{F_0}, ..., \vec{F_3}$) Dabei wird mit den beiden hinteren, äußeren Punkten des Korridors begonnen. Da die Ecken eines Raumes immer im Uhrzeigersinn gespeichert werden, lassen sich diese Punkte heraussuchen, indem die s'te Ecke des Raumes und die im Uhrzeigersinn darauf Folgendel, auswählt werden.

$$\vec{F_0} = \vec{C_s}$$
$$\vec{F_1} = \vec{C_{((s+1) \pmod 4)}}$$

Damit wird der Richtungsvektor $\hat{rTL}$ (rightToLeft) errechnet, indem die rechte Ecke von der linken Ecke subtrahiert wird und das Ergebnis dann normalisiert wird.

$$\vec{rTL} = \vec{F_0} - \vec{F_1}$$

$$\hat{rTL} = \frac{\vec{rTL}}{\left|\vec{rTL}\right|}$$

Um die beiden vorderen äußeren Ecken zu bestimmen, muss nun ein Richtungsvektor $\hat{tF}$ (toFrontWall) berechnet werden, der von der Seite S_s in Richtung der gegenüber liegenden Seite des Raumes zeigt. Hierfür wird das Kreuzprodukt von $\hat{rTL}$ und dem nach oben gerichteten Einheitsvektor berechnet und das Ergebnis normalisiert.

$$\hat{up} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$$

$$\vec{tF} = \hat{rTL} \times \hat{up}$$

$$\hat{tF} = \frac{\vec{tF}}{\left|\vec{tF}\right|}$$

Um nun die vorderen Punkte zu bestimmen, wird einfach der neue Richtungsvektor $\hat{tF}$ mit $2 * dP$ multipliziert und auf die hinteren Punkte addiert.

$$\vec{F_2} = \vec{F_0} + \hat{tF} * dP$$
$$\vec{F_3} = \vec{F_1} + \hat{tF} * dP$$

Nach dem nun klar gewordenen Prinzip, die Knotenpunkte zu berechnen, indem schon errechnete Knotenpunkte mit den entprechenden Richtungsvektoren multipliziert werden, lassen sich nun die äußeren oberen Ecken des Korridors bestimmen. Hierfür werden die unteren Ecken mit dem Vektor $\vec{T}$ addiert, der sich wie folgt berechnet:

$$\vec{T} = \hat{up} * h$$

$$\vec{F_4}, ..., \vec{F_7} = \vec{F_0} * \vec{T}, ..., \vec{F_3} * \vec{T}$$

Um die inneren Punkte zu berechnen, werden die äußeren Ecken nach dem selben Prinzip, jeweils mit den entsprechenden Richtungsvektoren, die dann mit w multipliziert werden, addiert. Die inneren Punkte liegen mit den äußeren Punkten auf einer Höhe. Die linken Ecken werden also jeweils nach rechts addiert, die rechten nach links, die hinteren nach vorne, die vorderen nach hinten, sodass jede innere Ecke mit zwei w-langen Vektoren von den äußeren Punkten in Richtung der Mitte verschoben wurde. Für die Richtung nach links wird $\hat{rTL}$ verwendet, nach rechts $\hat{rTL} * (-1)$, nach vorne $\hat{tF}$ und nach hinten $\hat{tF} * (-1)$.

In der folgenden Erklärung wird davon ausgegangen, dass $D <= 0.5$ ist und somit die Haupttür links und die Seitentür rechts erzeugt wird, andernfalls wären natürlich $\hat{rTL}$ und $\hat{rTL} * (-1)$ vertauscht. Genauso verhält es sich mit den beiden äußeren, unteren, vorderen Ecken F_2 (links vorne) und F_3 (rechts vorne), die Nutzung in den folgenden Formeln wäre vertauscht, wenn $D > 0.5$.

Für die Tür-Vertices wird zunächst der untere Mittelpunkt der Tür ($\vec{mD}$) berechnet.

Hierfür ist es zunächst wichtig die entsprechende Achse $\vec{mDX}$ zu berechnen, auf der die Tür dann erzeugt wird (siehe Abbildung 4.5)

Dafür wird erstmal der Punkt $\vec{dP}$, an der Frontalseite des Korridors, diagonal hinter $\vec{P}$ berechnet.

$$\vec{xP} = \vec{P} + \hat{tF} * dP + \hat{rTL} * dP$$

Von da aus spannt sich die Achse $\vec{mDX}$ dann bis zur anderen Seite des Korridors. Da $\vec{mD}$ ja den Mittelpunkt der Haupttür darstellen soll, muss an beiden Seiten der Achse

$\frac{dw}{2}$ als Padding eingesetzt werden, damit die Tür nicht weiter links oder rechts übersteht, falls D einen Extremwert annimmt.

$$m\vec{D}X = (F_2 + r\hat{T}L * (-1) * \frac{dw}{2}) - (x\vec{P} + r\hat{T}L * \frac{dw}{2})$$

$m\vec{D}$ berechnet sich dann folgendermaßen:

$$m\vec{D} = x\vec{P} + m\vec{D}X * D$$

Die Berechnung und $m\vec{D}X$ wird auch aus dem in Abbildung 4.6 dargestellten Quelltext ersichtlich.

Die Berechnung der Vertices, in der Haupttür ist nun trivial, erfolgt mit der bekannten Methode und wird auch in Abbildung 4.5 ersichtlich. Türhöhe (dh), Türbreite (dw) und Wandbreite (w) werden mit den entsprechenden Richtungsvektoren multipliziert und auf mD addiert.

Listing 4.2: calcMainDoor.cs

```
bool mainDoorIsRight = mainDoorPosition > 0.5f; // main door is more right than left
    this.sideDoorIsRight = !mainDoorIsRight;

    Vector3 innerLeftToRight = leftToRight + Vector3.Normalize(rightToLeft) * wallThickness * 2;

    Vector3 innerDoorGenSpanLeftToRight = innerLeftToRight + Vector3.Normalize(rightToLeft) * rotationPointDistance * 2;

    //leftToRight but less long: whole length - inner - rotpointdistance - doorwith
    Vector3 innerDoorGenSpanLeftToRightDoorPadding = innerDoorGenSpanLeftToRight + Vector3.Normalize(rightToLeft) * doorWidth;

    Vector3 toBehindRotPointLeftRight = Vector3.Normalize(leftToRight) * rotationPointDistance * 2;

    Vector3 sideDoorCornerBehindRotPoint = sideDoorIsRight ? boRiFrCorner + -toBehindRotPointLeftRight : boLeFrCorner + toBehindRotPointLeftRight;

    Vector3 innerCrossDirection = sideDoorIsRight ? -innerDoorGenSpanLeftToRightDoorPadding : innerDoorGenSpanLeftToRightDoorPadding;

    Vector3 sideDoorCornerBehindRotPointDoorPadding = sideDoorCornerBehindRotPoint + Vector3.Normalize(innerCrossDirection) * doorWidth / 2;
    //main door origin is generated between sideDoorCornerBehindRotPoint and the other corner - inner - half of the door width

    if (sideDoorIsRight)
    {
        mainDoorPosition = 1 - mainDoorPosition;
    }

    Vector3 mainDoorOrigin = sideDoorCornerBehindRotPointDoorPadding
                             + innerCrossDirection * mainDoorPosition;
```


Abbildung 4.6: Der Quelltext, der für die Errechnung des Mittelpunktes der Haupttür eines Korridors verantwortlich ist

Faces Der nächste Schritt, der für die Meshgenerierung erforderlich ist, ist das Spannen der verschiedenen Faces (Triangles). Die meisten für den Korridor erforderlichen Flächen bestehen nur aus zwei aneinanderliegenden Triangles, die somit gemeinsam die viereckige Fläche bilden. Die Ausnahmen sind die Frontalfläche und die Seitenfläche, in die die Seitentür generiert wurde (jeweils mit der zugehörigen Innenseite). Die Struktur dieser Ausnahmen wird auch in Abbildung 4.5 ersichtlich. Auch hier werden die benötigten Vertices jeweils durch Multiplizieren mit den entsprechenden Richtungsvektoren und der anschließenden Addierung auf bestehende Vertices errechnet. [1]

Um die späteren UV Berechnungen nicht zu verunreinigen, wird jeder Vertex dem Buffer pro Face einmal hinzugefügt. Die Faces werden dann gespannt, indem die jeweiligen Indizes der Vertices im Buffer aufgelistet werden. Dies beides wird dann in Listenform dem neu erzeugten $MeshFilter$ Objekt übergeben.

UV Die Berechnung der UV-Koordinaten erfolgt für jeden Knotenpunkt einzeln und wird nach Triangles geordnet iterativ durchgeführt. Der Quelltext dazu findet sich in der GenerateUV Methode, die im folgenden Quelltext abgebildet ist. Die Berechnung besteht daraus die Normale des Triangles zu berechnen, diese mit den Richtungsvektoren abzugleichen und die UV-Vektoren der Vertices dann entsprechend zu Rotieren.

Listing 4.3: uvCode.cs

```
Vector2[] GenerateUV(Vector3[] vertices, int[] triangles)
    {
        Vector2[] uv = new Vector2[vertices.Length];

        for (int i = 0; i < triangles.Length; i += 3)
        {
            int vi0 = triangles[i + 0];
            int vi1 = triangles[i + 1];
            int vi2 = triangles[i + 2];

            Vector3 normal = CalculateNormal(vertices[vi0],
               vertices[vi1], vertices[vi2]);

            if (VectorsAreParallel(normal, toFrontWall))
            {
                Vector3 v0 = vertices[vi0];
                Vector3 v1 = vertices[vi1];
                Vector3 v2 = vertices[vi2];

                v0 = gameObject.transform.TransformPoint(v0);
                v1 = gameObject.transform.TransformPoint(v1);
                v2 = gameObject.transform.TransformPoint(v2);
```

[1] Bei der Seitentür verläuft die Berechnung des Türmittelpunkts deutlich simpler, es handelt sich einfach um die Mitte des Korridors.

```
                Quaternion rotation =
                    Quaternion.FromToRotation(leftToRight,
                    Vector3.forward);

                v0 = rotation * v0;
                v1 = rotation * v1;
                v2 = rotation * v2;

                uv[vi0] = new Vector2(v0.z, v0.y);
                uv[vi1] = new Vector2(v1.z, v1.y);
                uv[vi2] = new Vector2(v2.z, v2.y);
            }
            else if (VectorsAreParallel(normal, leftToRight))
            {
                Vector3 v0 = vertices[vi0];
                Vector3 v1 = vertices[vi1];
                Vector3 v2 = vertices[vi2];

                Quaternion rotation =
                    Quaternion.FromToRotation(leftToRight,
                    Vector3.forward);

                v0 = rotation * v0;
                v1 = rotation * v1;
                v2 = rotation * v2;

                uv[vi0] = new Vector2(v0.x, v0.y);
                uv[vi1] = new Vector2(v1.x, v1.y);
                uv[vi2] = new Vector2(v2.x, v2.y);
            }
            else if (VectorsAreParallel(normal, Vector3.up))
            {
                Vector3 v0 = vertices[vi0];
                Vector3 v1 = vertices[vi1];
                Vector3 v2 = vertices[vi2];

                Quaternion rotation =
                    Quaternion.FromToRotation(leftToRight,
                    Vector3.forward);

                v0 = rotation * v0;
                v1 = rotation * v1;
                v2 = rotation * v2;

                uv[vi0] = new Vector2(v0.z, v0.x);
```

```
                uv[vi1] = new Vector2(v1.z, v1.x);
                uv[vi2] = new Vector2(v2.z, v2.x);
            }
        }

        return uv;
    }
```

4.3 Wandgenerierung

Natürlich reichen die Meshs der Korridore noch nicht aus, um das Gefühl eines Raumes zu vermitteln, zumal diese ja auch nur an einer Seite des Raumes statuiert sind. Wie in Unterabschnitt 4.1.1 besprochen, sollen die Räume von Wänden umgeben sein.

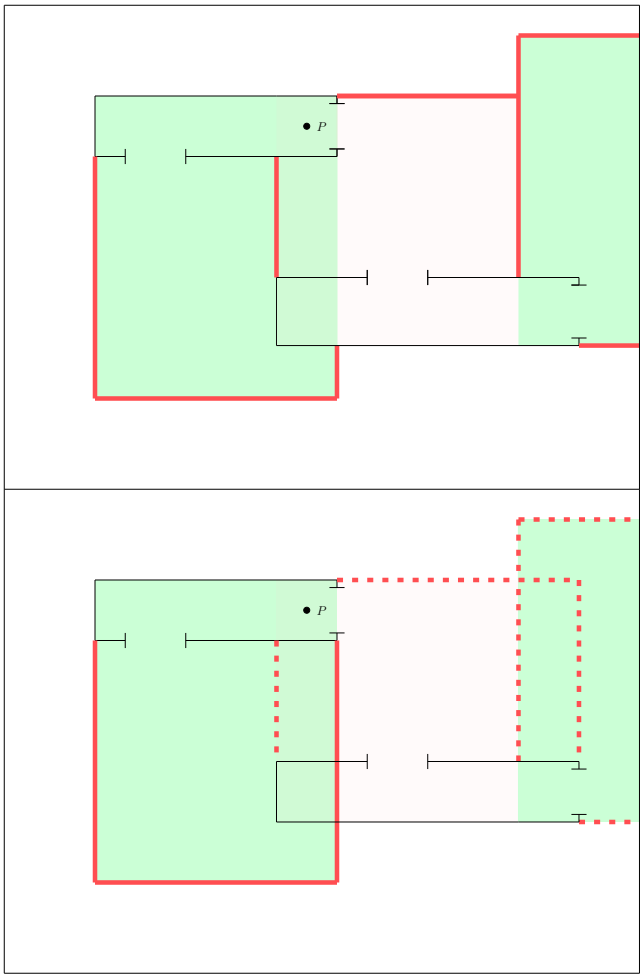

Abbildung 4.7: Vergleich der zwei Wandgenerierungsvarianten. Oben: Die “realistische” Variante, die in Joystick- und Teleportationsbedingung eingesetzt wird. Unten: “unmögliche” Variante, die in der Real-Walking Bedingung die Wände generiert. Gestrichelte Linien repräsentieren Wände, die erst unter bestimmten Umständen sichtbar werden

Die Wandgenerierung erfolgt über ein Prefab, dass instanziiert und dann gestreckt, positioniert und rotiert wird. Es besteht aus einem Würfel mit der entsprechenden Textur, der Höhe der Wandhöhe h, und der Breite und Tiefe der Wände w. Dies geschieht in der Methode “GenerateWall”. Diese bekommt als Eingabe zwei Punkte, und instanziiert dann eine Wand, die von dem einen dieser Punkte zum anderen reicht. Außerdem müssen für jeden Raum, für jede Wand, jeweils die zwei Endpunkte der Wand berechnet werden. In dem hier vorgestellten Projekt werden die Wände auf zwei verschiedene Arten generiert. In der Abbildung 4.7 werden die beiden Varianten miteinander verglichen.

4.3.1 Realistische Wandgenerierung

Die erste Art, fortlaufend "realistische" Variante genannt, generiert die Wände zwischen den Räumen so, wie sie auch real existieren könnten. Dazu gibt es einen ausgefeilten Algorithmus, der alle Eventualitäten der Raumgenerierung kennt und die Wände so zwischen den Räumen platziert, dass die Räume noch möglichst viel Platz bieten, aber auch rundherum von Wänden umgeben sind. Der Code für diese Generierung wird in der "RoomGenerator"-Klasse in den beiden "GenerateWallsLegacy"-Methoden ausgeführt. Diese Variante wird in den beiden Kontrollbedingungen "Teleport" und "Joystick" verwendet, weil Versuchsbedingungen mit ihnen keine Impossible-Spaces beinhalten.

Bei der realistischen Wandgenerierung ist die Idee, dass die Wände genau so zwischen die Korridore platziert werden, dass der nachfolgende Raum die Priotität über den Platz hat. Der Bereich O, der in der Abbildung 4.1 zu sehen war, gehört bei dieser Variante also immer zu dem Raum B und nicht zu dem Raum A.

Davon, ob der Korridor des Raumes an der längeren, oder der kürzeren Seite des Rechtecks generiert wurde, hängt ab, wieviele Wände generiert werden. Ist der Korridor an der längeren Seite des Raumes, werden ausgehend vom Korridor des letzten Raumes zwei Wände generiert, da die breite Seite des nächsten Raumes als Wand genutzt werden kann. Ist jedoch der Korridor des aktuellen Raumes an der kürzeren Seite des Rechtecks, so muss eine dritte Wand generiert werden, die verhindert, dass eine offene Stelle zwischen dem aktuellen und dem nächsten Raum entsteht.

Um die Wände generieren zu können, ist es wichtig, die Punkte diagnonal vor und hinter dem Punkt $\vec{P}$ des vorherigen Raumes zu kennen. Dazu werden diese nach der Generierung als Iterationsübergreifende Variablen gespeichert und somit an den jeweils nächsten Raum übergeben, siehe Unterabschnitt 4.5.2.

4.3.2 Unmögliche Wandgenerierung

Die andere Variante der Levelgenerierung, hier "unmögliche" Variante genannt, umschließt alle Räume mit drei Wänden, zeigt aber immer nur die Wände des Raums, in dem sich die Nutzer:in gerade befindet an und hat dementsprechend die Freiheit, die gesamte Raumgröße zu umschließen, weil keine Kompromisse für den sich überlappenden Bereich O gemacht werden müssen (siehe Abbildung 4.1). Der Code für diese Generierung wird in der "RoomGenerator"-Klasse in den beiden "GenerateWalls"-Methoden ausgeführt. Im Experiment wird diese Variante ausschließlich in der Real-Walking Bedingung eingesetzt. Durch sie werden die Räume erst zu Impossible-Spaces, da die Räume sich nun überlappen.

Die Umsetzung der unmöglichen Wandgenerierung erfolgt recht ähnlich, wenn auch um einiges minimalistischer als die der realistischen. Anstatt die Wandgenerierung auf die verschiedenen Räume aufzuteilen, werden die Wände so generiert, dass sie abgesehen

von den Korridoren, des aktuellen, und des vorherigen Raumes den ganzen Raum umschließen. Das Ein- und Ausblenden, der Wände wird dann vom "RoomAndProgress-Manager", der ja Überblick darüber hat, wo die Nutzer:in sich gerade befindet, geregelt. Es werden immer nur die Wände des aktuellen Raumes, angezeigt.

4.4 Zufallsfaktoren in der Levelgenerierung

Der hier beschriebene Levelgenerierungsalgorithmus nutzt das Element des Pseudozufalls um durch Verändern verschiedener Variablen das Ergebnis zu beeinflussen. So ist es möglich viele unterschiedliche Level zu generieren. Welche Variablen sich dafür unterscheiden können, werde ich an dieser Stelle vorstellen.

4.4.1 Korridorrichtung

Zu Beginn der Korridorgenerierung wird eine Seite des Raumes als Korridorseite S_s bestimmt. Diese wird pseudozufällig ausgewählt. Im ersten Raum ist es noch vollkommen beliebig, welche Seite ausgewählt wird[2]. Bei den weiteren Räumen hingegen können bestimmte Seiten nicht ausgewählt werden, weil der Korridor nach der Generierung sonst mit dem Ausgang des Korridors des vorherigen Raumes überlappt. Durch die Art und Weise, wie die Eckenberechnung der Räume funktioniert, lässt sich feststellen, dass S_3 in allen Umständen immer die Seite ist, aus der der Korridor des letzten Raumes herauszeigt (siehe Unterabschnitt 4.1.2). S_3 kann also zur Korridorgenerierung nicht ausgewählt werden. Falls die Seitentür des vorherigen Raumes auf der rechten Seite liegt (wie in der Abbildung 4.2), muss außerdem S_0 blockiert werden, da die Nutzer:in sonst von einem Korridor direkt in den nächsten läuft. Ist die Seitentür aber auf der linken Seite ist die Ecken- und Seitenbelegung, wie in Unterabschnitt 4.1.2 beschrieben, nicht nur horizontal, sondern auch vertikal spiegelverkehrt. Deshalb muss in dem Fall die Seite S_2 blockiert werden. Damit diese Seiten bei der zufälligen Auswahl ignoriert werden, funktioniert die Methode dafür ("RandomDirectionBlocked", siehe Abbildung 4.8) folgendermaßen: Als Eingabe wird eine Liste LB mit den blockierten Richtungen übergeben. Alle Richtungen (repräsentiert von Zahlen von 0-3), die darin vorkommen, werden aus der Liste aller möglichen Richtungen entfernt. Mit dem Unity-Objekt *Random* lässt sich durch Abrufen der Eigenschaft *value* eine zufällige Zahl zwischen 0 und 1 generieren[3]. Um nun die Zahl aus der Liste verbleibender Richtungen LD auszuwählen, wird das Ergebnis nun mit $4 - length(LB)$ multipliziert, um den Index zu berechnen.

[2]Im Versuchsaufbau wird dabei aber immer die selbe Seite ausgesucht um das Erscheinen im Raum einheitlicher zu gestalten. (Die Nutzer:in guckt nach dem Start des Versuchs in Richtung des Korridors).

[3]Falls tatsächlich eine volle 1 ausgewählt wird, würde dies dazu führen, dass eine zu hohe Zahl ausgewählt werden würde. Deshalb wird im Code mithilfe der Min-Funktion dafür gesorgt, dass dieser Fall nicht eintreten kann, siehe Abbildung 4.8

Listing 4.4: RandomDirectionBlocked.cs

```
private int RandomDirectionBlocked(List<int> block)
    {
        if (block.Count == 0)
        {
            return RandomDirection();
        }

        List<int> directions = new List<int>
        {
            0, 1, 2, 3
        };

        foreach (int i in block)
        {
            if (i == -1)
            {
                return RandomDirection();
            }
            else
            {
                directions.Remove(i);
            }
        }

        float rand = Random.value;
        rand = Math.Min(rand, 0.99999f);

        return directions[(int) (rand * ( 4 - block.Count))];
    }
```

Abbildung 4.8:

4.4.2 Türposition

Die Frage, in welche Richtung der jeweils nächste Raum generiert wird, hängt gänzlich davon ab, wie die Haupttür des Korridors positioniert ist. In Unterabschnitt 4.2.1 wird erklärt, wie die Haupttür auf der Frontalwand des Korridors generiert wird und abhängig von dem Wert D auf der links-rechts Achse positioniert wird. Falls D einen Wert von 0 hat, wird die Tür ganz links positioniert, bei einem Wert von 1 ganz rechts. Davon ist dann auch die Positionierung der Seitentür abhängig. Ist $D <= 0.5$, wird die Seitentür in die rechte Seitenwand des Korridors generiert, sonst in die Linke. Davon ist dann natürlich auch wieder abhängig, in welche Richtung der folgende Raum generiert wird, denn die Nutzer:in soll ja aus der Seitentür heraus-, in den Raum hineintreten. Man

kann den Wert D also sowohl als einen verstehen, der feinere Unterschiede der Position auf der Frontalwand bestimmt, damit nicht jeder Raum genau gleich wirkt, als auch als einen wichigen Wert, der die Richtung des folgenden Raumes vorherbestimmt.

Auch hier müssen wieder einige Varianten ausgeschlossen werden, um die Begehbarkeit der erzeugten Level zu garantieren.

S_1 stellt - unabhängig der Richtung - immer die Seite gegenüber der Seite dar, aus der man kommt, wenn man aus dem vorherigen Raum in den aktuellen Raum tritt. Wenn der Korridor an diese Seite generiert wurde, kann die Türposition nicht mehr so sein, dass der als nächstes generierte Raum Überschneidungen mit der Seite des aktuellen Raumes hat, aus der der Korridor des vorherigen Raumes kommt, da sonst der Weg aus dem Korridor des vorherigen Raumes nicht frei ist, weil der nachfolgende Raum darüber generiert wurde. Je nach Raumdimensionen würden sich dann sogar potentiell 3 Räume überlappen.

Genauso wichtig ist, dass das Ende des Korridors des aktuellen Raumes, nicht in Richtung des vorherigen Raumes zeigt.

Diese Varianten werden bei der Levelgenerierung durch Fallunterscheidungen ausgeschlossen. Dazu hat die Methode "GenerateRandomDoorPosition" in der "GenerateLevel"-Klasse, die für die Berechnung von D verantwortlich ist zwei Parameter, $forceLeftSideDoor$ und $forceRightSideDoor$, die zwar beide $false$ sein dürfen, (dann wird nichts verhindert), aber nicht beide $true$ sein dürfen.

Die Berechnung von D erfolgt nun wieder über das Aufrufen von Unitys $Random.value$ Attribut, welches einen pseudozufälligen Wert der, zurückgibt.

Soll die rechte Seite forciert werden, wird der Wert durch 2 geteilt. Falls die linke Seite forciert werden soll, wird nach der Division durch 2, nochmal $0,5$ addiert. Falls nichts blockiert werden soll, wird der Wert direkt zurückgegeben.

4.5 Umsetzung der Levelgenerierung

Nachdem ich nun nacheinander die einzelnen Komponenten der Levelgenerierung erklärt habe, werde ich nun erklären, wie diese zusammenspielen um den Levelgenerierungsalgorithmus zu bilden. Diesen werde ich zunächst in seiner Vorgehensweise erläutern und anschließend die Grundstruktur des Quelltextes darstellen.

4.5.1 Funktionsweise des Levelgenerierungsalgorithmus

Im Rahmen des in dieser Arbeit vorgestellten Experiments war es notwendig, Level zu generieren, bei denen im vorhinein die Anzahl der Räume L definiert werden konnte, und die dementsprechend nicht unendlich lang waren. Jedoch ist es durchaus trivial den

folgend vorgestellten Algorithmus leicht zu verändern, sodass er die Räume fortlaufend generiert, während die Nutzer:in bereits dabei ist, das Level zu erkunden. Nachdem L ausgewählt wurde (in diesem Fall indem die Proband:in eine der Versuchsbedingungen auswählt, siehe Abschnitt 3.11) kann das Level generiert werden. Grundlegend besteht der Algorithmus aus der Deklarierung einiger Variablen, die den Zustand über die Iterationen hinweg speichern, und einer Schleife, die pro Iteration einen neuen Raum generiert. In der hier vorgestellten Form des Algorithmus wird diese Schleife zu Beginn der Versuchsbedingung ganz ausgeführt, sodass das Level fertig generiert wurde, bevor die Proband:in angefangen hat, es zu erkunden. Um den Prozess so umzubauen, dass das Level fortlaufend weiter generiert wird, ist es lediglich notwendig, den Quellcode, der innerhalb der Schleife steht, in eine Funktion auszuverlagern, die immer dann, wenn die Nutzer:in einen Raum betritt, einen weiteren erzeugt. Damit dies nicht bemerkt wird, ist es an dieser Stelle ratsam, einen kleinen Puffer von etwa 2 oder 3 Räumen einzubauen, sodass die neu erzeugten Räume nicht direkt vor den Augen der Nutzer:in auftauchen. Dies lässt sich einfach erzielen, indem man zu Beginn des Levels einige wenige Räume erzeugt und die neuen Räume erst hinter diesen generiert werden.

4.5.2 Inhalt einer Iteration

Eine Iteration besteht aus zwei Bereichen. Im ersten wird zunächst der Raum anhand des aktuellen Zustands der Iterationsübergreifenden Variablen generiert, indem die "Generate" Methode der "RoomGenerator" Klasse aufgerufen wird. Der zweite Teil ist dann dafür verantwortlich, die nächste Iteration vorzubereiten. Dies hat den Zweck, dass der erste Teil der ersten Iteration, den Ursprungszustand der Variablen nutzen kann. Ein Beispiel dafür sind die vom Oculus Integration SDK übergebenen Eckkoordinaten des realen Trackingspaces oder die, vom Designer ausgewählte, erste Korridorrichtung, die das Erscheinen im ersten Raum vereinheitlichen soll.

Generierungs-Teil Darin werden dann zunächst die Meshs des Korridors erzeugt (siehe Unterabschnitt 4.2.1) und als GameObjects instanziiert. Zudem werden diese mit den entprechenden Materialien für die Optik und den entsprechenden Scripts versehen. Dann werden die für den Rotation-Gain erforderlichen Objekte instanziiert, positioniert und die entsprechenden Mechanismen in die Wege geleitet, indem die Eigenschaften des zuständigen "RotationRedirector" gesetzt werden. Zudem wird die Innenausstattung des Korridors instanziiert, positioniert, mit den entsprechenden Scripts ausgestattet und mit den eben erzeugten Mechanismen, die den Rotation-Gain überwachen, verknüpft.

Die Generierung der Wände wird als nächstes in die Wege geleitet, je nach Variante wird zwischen den verschiedenen Methoden unterschieden, die danach aufgerufen werden, um die Wandpunkte zu berechnen und dann Wandobjekte zu initialisieren. Hierfür ist zudem erforderlich, das in Abschnitt 4.3 besprochene Prefab zur Verfügung zu stellen, das vorher von der Designer:in erstellt werden muss.

Berechnungsteil Der zweite Teil einer Iteration ist dann für die Berechnung der Zustandsvariablen zuständig. Diese müssen berechnet werden, weil der erste Teil der nächsten Iteration keinen Zugriff mehr auf die Objekte der aktuellen Iteration hat. Hier findet die in Unterabschnitt 4.1.2 erklärte Eckenberechnung statt. So können die Richtungsvektoren und andere Informationen von dem Raum, der in der aktuellen Iteration generiert wurde, genutzt werden, um die Generation des nächsten Raumes vorzubereiten. Auch die in Abschnitt 4.4 besprochenene Pseudozufallsvariablen und die in Unterabschnitt 4.3.1 besprochenen Punkte diagonal vor und hinter dem Punkt $\vec{P}$ des vorherigen Raumes werden hier berechnet, sodass sie in der nächsten Iteration schon feststehen und genutzt werden können.

Kapitel 5

Konzipierung der Pilotierungsstudie

Im folgenden Kapitel möchte ich das für diese Arbeit konzipierte Experiment vorstellen. Dabei handelt es sich um eine informelle Pilotierungsstudie zum Vergleich von Fortbewegungsarten für das Raumverständnis und das Präsenzgefühl in der generierten virtuellen Umgebung. Dabei soll konkreter untersucht werden, ob die Fortbewegungsart "Real-Walking" in diesen Punkten bessere Effekte erzielt als vergleichbare alternative Fortbewegungsarten. Um das Experiment vorzustellen, werde ich zunächst die Hypothesen, denen ich mit dieser Arbeit auf den Grund gehen will, aufstellen um deutlich zu machen, was das Ziel des Experiments ist. Im darauf folgenden Teil werde ich dann beschreiben wie die statistische Versuchsplanung der konzipierten Pilotierungsstudie aussieht. Dabei werde ich genauer auf die drei verschiedenen Fortbewegungsbedingungen eingehen, die untersucht werden sollten und beschreiben, wie diese für die Studie implementiert wurden.

5.1 Hypothesen und Messvariablen

An dieser Stelle möchste ich die in dem Experiment zu überprüfenden Hypothesen vorstellen.

1. Nachdem ein mit der hier vorgestellten Levelgenerierungmethode erstelltes Level mit der Fortbewegungsart "Real-Walking" durchquert wurde, ist die Fähigkeit, die darin zurückgelegte Strecke einzuschätzen, besser als wenn es mit den alternativen Fortbewegungsarten "Joystick" und "Teleportation" durchquert wurde.

2. Beim Durchqueren der mit der hier vorgestellten Levelgenerierungmethode erstellten Level mit der "Real-Walking" Fortbewegungsart haben Nutzer:innen ein größeres Präsenzgefühl als mit den beiden Alternativen "Joystick" und "Teleportation".

5.1.1 Distanzschätzung

Mehrere Forschungsergebnisse deuten darauf hin, dass natürliches Gehen das Raumverständnis im Vergleich zu anderen Fortbewegungsarten verbessert. [10, 17, 21]

Die hier vorgestellte Hypothese basiert zum Teil auf der Annahme, dass ein besseres Raumverständnis dazu führt, dass Nutzer:innen bereits gegangene oder anderweitig zurückgelegte Distanzen besser schätzen können. Dafür spricht die von Peck et al. [17] gefundene Untersuchung, dass Proband:innen in Real-Walking Fortbewegungsbedingungen die größe der virtuellen Umgebung nach dem Versuch siginifikant besser einschätzen konnten, als mit anderen virtuellen Fortbewegungsarten.

Wie gut die Nutzer:innen die zurückgelegte Distanz schätzen, wird hier gemessen, indem die Proband:innen nach jedem Durchlauf einer Versuchsbedingung schätzen sollen, wieviele Meter sie sich gerade fortbewegt haben.[1] Neben der Schätzung wurde auch die vom virtuellen Avatar zurückgelegte Strecke gemessen (siehe Abschnitt 3.12), sodass die absolute Differenz von Schätzung und Messung widerspiegelt, wie gut die Proband:innen die zurückgelegte Strecke einschätzen konnten.

Diese Art von Messung bietet sich für Level an, die mit der hier vorgestellten Methode generiert werden, weil diese sehr linear sind. Ein Raum folgt auf den nächsten, ohne dass die Nutzer:in Richtungsentscheidungen tätigt oder Abzweigungen geht.

Zwar gibt es in der Literatur viele Hinweise darauf, dass Menschen in virtuellen Umgebungen die Entfernung zwischen ihnen und Objekten in der Umgebung, verglichen mit der realen Umgebung, nicht besonders gut einschätzen können [1, 19], allerdings werden in der hier vorgestellten Studie alle drei Fortbewegungsarten in einer (sich auch nicht grundlegend zwischen den Versuchsbedingungen unterscheidenden) virtuellen Umgebung durchgeführt, sodass dieser Effekt ausgeglichen wird.

5.1.2 Präsenzgefühl

In dem Paper [32] geben Usoh et al. eine Zusammenfassung über die bis dahin diskutierten Ideen darüber, wie die Stärke der Präsenz in virtuellen Umgebungen stark erhöht wird. Diese wird in 5 Punkte aufgeteilt, die übersetzt und zusammengefasst so lauten:

1. Informationen werden den Proband:innen in hoher Auflösung angezeigt, in einer Weise, die die Existenz des Displays nicht erkennen lässt.
2. Die dargestellte Realität ist über alle Sinnesmodalitäten hinweg konsistent.
3. Die Möglichkeit der Nutzer:in durch die Umgebung zu navigieren und mit Objekten oder anderen Agenten zu interagieren.
4. Der virtuelle Avatar der Nutzer:in sollte sich äußerlich oder funktionell dem echten Körper dieser ähneln und angemessen auf Kopf-, Augen- und Gliedmaßenbewegungen reagieren.

[1] Dabei ist zu spezifizieren, dass es nicht um die euklidische Entfernung von Start und Ziel ging (die "Luftlinie"), sondern um die gesamte zurückgelegte Strecke.

5. Die Verbindung zwischen den Aktionen und ihren Effekten sollte unkompliziert genug sein, sodass sie leicht zu erlernen ist.

Besonders interessant für dieses Projekt sind hier die Punkte 2 und 3, da diese sich zwischen den verschiedenen Fortbewegungsarten unterscheiden. So ist zum Beispiel davon auszugehen, dass die Real-Walking-Fortbewegungsart in mehr Sinnesmodalitäten Konsistenz bietet, da die Nutzer:in ihre Stellung im Raum auch durch ihre Propriozeption und ihren Gleichgewichtssinn wahrnehmen kann. Nicht außer Acht zu lassen ist natürlich auch, dass natürliches Gehen durch dessen Alltagsnähe eine deutlich intuitivere Fortbewegungsmethode darstellt und dementsprechend anzunehmen ist, dass es als solche einem Präsenzgefühl weniger im Weg steht als die ungewohntere Navigation mit den Alternativen.

Um diesen Effekt zu erfassen, wurde mit den Teilnehmer:innen nach jeder Bedingung ein Slater-Usoh-Steed-Präsenz-Questionaire [32] durchgeführt. (Fortlaufend SUS-Questionaire genannt.) Dieser ist im Durchlauffragebogen enthalten (P1-P7) und ist im Anhang abgebildet. Der SUS-Questionaire besteht aus 6 Fragen, die auf einer Skala von 1-7 beantwortet werden, wobei 7 immer für die höchste und 1 für die geringste wahrgenommene Präsenz in der Umgebung steht. Die letze Frage des SUS-Questionaires ermöglicht den Proband:innen noch etwas dazu hinzuzufügen.

Usoh et al. überprüfen in ihrer Arbeit [31], inwieweit sich das Präsenzgefühl zwischen natürlichem Gehen, virtuellem Gehen und Fortbewegung durch Fliegen unterscheidet. Die Ergebnisse deuten darauf hin, dass die beiden Geh-Fortbewegungsarten zu einem signifikant höhreren Präsenzgefühl führen und sie konkludieren, dass dies mit der stärkeren Assoziation mit dem virtuellen Avatar zu tun haben könnte. Dies ist die Grundlage für die zweite, in dieser Thesis überprüfte, Hypothese.

5.2 Versuchsplanung

Im Folgenden werde ich den Aufbau des Experiments beschreiben. Um aus der geringen Teilnehmerzahl der Studie (siehe Abschnitt 6.1) dennoch möglichst viele Daten zu sammeln und weil nicht davon auszugehen ist, dass die Bedingungen der Fortbewegungsarten sich gegenseitig beeinflussen, habe ich mich für ein "Within-Subjects Design" entschieden. Folglich haben alle Teilnehmenden alle 3 Konditionen durchlaufen. Um dabei zu vermeiden, dass die Reihenfolge der Konditionen die Ergebnisse beeinflusst (beispielsweise indem alle zuerst die Real-Walking Bedingung durchlaufen und zu diesem Zeitpunkt noch am besten einschätzen können, wie weit sie gegangen sind, weil die Schätzungsfähigkeit mit der Zeit abnimmt), wurden die Teilnehmenden in 3 Gruppen eingeteilt, die die 3 möglichen unterschiedlichen Reihenfolgen der 3 Bedingungen abbilden.

Um die Schätzung der zurückgelegten Strecke im Generellen, sowohl bei mehr als auch bei weniger vielen Räumen zwischen Start und Ziel (also bei längeren und kürzeren

Leveln) in die Daten einfließen zu lassen, wurden die Bedingungen in unterschiedlichen Raumlängen durchgeführt. Um diesen Effekt gleichmäßig auf alle Messergebnisse zu verteilen, wurden die Raumlängen alternierend den unterschiedlichen Gruppen und den unterschiedlichen Fortbewegungsbedingungen zugewiesen, sodass jede Teilnehmer:in einmal jede der 3 Fortbewegungsarten und einmal jede der drei Raumlängen 3, 6 und 9 durchlief. Es wurde auch darauf geachtet, dass sich für jede Gruppe die Kombination aus Fortbewegungsart und Raumlänge bei allen drei durchgeführten Bedingungen unterschied.

	G1	G2	G3
Schritt 1	$B3$	$C2$	$A1$
Schritt 2	$C1$	$A3$	$B2$
Schritt 3	$A2$	$B1$	$C3$

Abbildung 5.1: Das Griechisch-Lateinische Quadrat, das die Reihenfolge der Versuchsbedingungen (Schritt 1-3) der verschiedenen Gruppen (G1-3) darstellt. A steht dabei für Real-Walking, B für Joystick Steuerung und C für Teleportation. $1-3$ repräsentieren die verschiedenen Raumlängen 3, 6 und 9

Dementsprechend bot sich als Versuchsplan ein Griechisch-Lateinisches Quadrat in der Größe 3×3 an. Abbildung 5.1 stellt das für die Studie genutzte Griechisch-Lateinische Quadrat dar, wobei A für die Versuchsbedingung "Real-Walking", B für die Versuchsbedingung "Joystick" und C für die Bedingung "Teleportation" steht, während 1 für eine Raumlänge von 3 Räumen, 2 von 6, und 3 für eine Raumlänge von 9 Räumen steht.

Die abhängige Variable ist die jeweilige Messvariable der Bedingung, also die absolute Differenz von Messung und Schätzung des zurückgelegten Wegs, oder das mit dem SUS-Präsenz Questionaire gemessene Präsenzgefühl. Die unabhängige Variable ist die Fortbewegungsart $(A - C)$.

5.3 Fortbewegungsbedingungen

Im Folgenden werde ich die drei verschiedenen Fortbewegungsbedingungen vorstellen, diese stellen die unterschiedlichen Fortbewegungsarten dar. Zunächst beschreibe ich die beiden virtuellen Fortbewegungsarten: Joystick und Teleportation. Diese repräsentieren klassische Fortbewegungsarten in VR. Im Kontrast dazu steht dann die Fortbewegungsart des natürlichen Gehens.

5.3.1 Kontrollbedingung I Joystick

In der Joystick-Kontrollbedingung (B) bewegen sich die Proband:innen virtuell mithilfe der sich an ihren Oclus Quest 2 Controllern befindenden Joysticks fort. Der sich auf dem Controller für die linke Hand befindende Joystick (von nun an "linker Joystick" genannt) ist dabei für die Fortbewegung zuständig. Der sich auf dem rechten Controller befindende Joystick ("rechter Joystick") gibt der Nutzer:in die Möglichkeit sich umzuschauen. Diese unterscheidet sich leicht von der Steuerung der meisten Joystick basierten Videospiele, indem ausschließlich die Yaw-Achsen Kopfdrehung von dem Joystick beeinflusst wird, nicht zusätzlich auch die Pitch-Achsen Kopfdrehung und indem die Kopfdrehung in gleichmäßigen Intervallen von 45° verändert wird. Wie in Unterabschnitt 4.3.1 erklärt, nutzt diese Bedingung die realistische Wandgenerierung.

Die Implementierung der Joysticksteuerung wurde bereits in Unterabschnitt 3.8.1 beschrieben. Dort sind auch die technischen Details beschrieben.

5.3.2 Kontrollbedingung II Teleportation

Auch in dieser Bedingung C haben die Proband:innen die Möglichkeit, sich ähnlich wie in Bedingung B mithilfe des rechten Joysticks in Intervallen von 45°% umzusehen. Die eigentliche Fortbewegung funktioniert hier allerdings auf anderem Wege. Wenn die Proband:innen die sich auf dem rechten Controller befindliche A-Taste gedrückt halten, sehen sie einen grünen Laserstrahl, der aus ihrer rechten Hand gestrahlt wird. Wenn dieser Strahl auf den Boden trifft, wird an dieser Stelle ein Teleportationscursor angezeigt. Dieser ist in Abbildung 3.8 abgebildet und dient sowohl dazu, der Proband:in anzuzeigen, wohin sie sich aktuell teleportieren würde, als auch dazu, die Blickrichtung nach der Teleportierung mithilfe eines Pfeils in der Mitte anzuzeigen. Währenddessen lässt sich mit dem linken Joystick diese Blickrichtung verändern, sodass schon vor dem Teleportieren entschieden werden kann, wohin danach geguckt wird. Um die Teleportierung zu dem ausgewählten Punkt mit der ausgewählten Blickrichtung durchzuführen, muss die Proband:in den Zeigefingertrigger des rechten Controllers bestätigen. Anschließend wird sie augenblicklich an die Stelle teleportiert, auf der kurz zuvor noch der Teleportationscursor zu sehen war. Dabei blickt sie dann auch in die Richtung, in die eben noch der Pfeil des Cursors gezeigt hat. Auch bei dieser Bedingung wird die realistische Wandgenerierung genutzt.

5.3.3 Versuchsbedingung Real-Walking

In der Versuchsbedingung Real-Walking können die Proband:innen sich durch natürliches Gehen im Raum bewegen. Nur in dieser Bedingung wird ein Rotation-Gain angewandt. Des Weiteren wird in dieser Bedingung die in Unterabschnitt 4.3.1 besprochene unmögliche Wandgenerierung verwendet, sodass die Proband:innen mehr Platz innerhalb der Räume haben. Dafür eine Raumstruktur genutzt wird, die darauf basiert, dass

Wände von Räumen, in denen sich die Nutzer:in gerade nicht befindet, ausgeblendet werden. Die Räume sich also überlagern sich folglich werden und somit zu Impossible-Spaces.

Kapitel 6

Durchführung und Ergebnisse

Nachdem das vorherige Kapitel die Versuchsplanung zur Pilotierungsstudie beschrieben hat, wird dieses davon handeln, wie die Studie durchgeführt und die Datenevaluation mitsamt der Berechnung der statistischen Tests verlaufen ist. Dementsprechend werden zunächst Umstände und Teilnehmer:innen des Experiments vorgestellt, dann der genaue Ablauf der Studie beschrieben, um danach die Methoden der Datenevaluation vorzustellen und diese, zu guter letzt, durchzuführen. Die Fragebögen wurden mit "Google-Forms" implementiert und sind im "Anhang - Fragebögen und Daten" mitsamt allen Daten und Antworten zu finden.

6.1 Umstände

Aufgrund der anhaltenden Covid-19 Situation wurde die hier vorgestellte Studie nicht, wie sonst bei vergleichbaren Arbeiten üblich, im offiziellen Rahmen der Universität durchgeführt, sondern unter meiner privaten Verantwortung. Es wurden ausreichend Sicherheitsmaßnahmen durchgeführt, um die Gefahr einer Infektion durch die Umstände des Experiments so gering wie möglich zu halten. Abgesehen von einer teilnehmenden Person, hatten alle Proband:innen zum Zeitpunkt der Durchführung bereits vollen Impfschutz. Alle Teilnehmenden wurden vor der Durchführung des Experiments mit einem Schnelltest auf Covid-19 getestet und ihre Kontaktdaten wurden festgehalten, um sie zu informieren, falls dennoch erforderlich. Außerdem wurden die genutzen Materialen nach jedem Versuchsdurchlauf desinfiziert. Alle Teilnehmenden wurden gebeten, während der Durchführung des Experiments einen Mund- und Nasenschutz zu tragen sowie sich vorher die Hände mit Handdesinfektionsmittel zu desinfizieren.

Aufgrund dieser Umstände war es leider nicht möglich, dem Standard einer solchen Arbeit entsprechend viele Proband:innen zu finden. Deshalb musste ich mich dafür entscheiden, eine Pilotierungsstudie statt einer vollwertigen Studie durchzuführen. Deshalb möchte ich an dieser Stelle betonen, dass die statistische Aussagekraft der hier erhobenen Ergebnisse nicht so groß wie gewünscht ist. Dies werde ich natürlich in der Auswertung und Einordnung der Ergebnisse berücksichtigen. Ich hoffe dennoch, dass zukünftige Studien mit mehr Teilnehmer:innen von meiner hier geleisteten Vorarbeit profitieren können.

6.2 Teilnehmende

An der Studie nahmen 10 Proband:innen im Alter zwischen 20 und 69 (Median: 23), davon 8 männlich und 2 weiblich, teil. Die Teilnehmenden waren ausnahmlos private Bekannte von mir und haben keinerlei Entschädigung für die freundliche Teilnahme erhalten. Alle Teilnehmer:innen gaben ihre mündliche Zustimmung zur Teilnahme an der Studie und der Verwertung und anonymen Veröffentlichung der darin erhobenen Daten ab. Da alle Teilnehmer:innen aus meinem engen persöhnlichen Umfeld kommen, wurde mir dieses Vorgehen von dem Zweitgutachter dieser Arbeit als ausreichend bestätigt. Die Teilnehmer:innen waren zwischen 1,62m und 1,89m groß (durschnittlich 1,80m). 5 der 10 Teilnehmer:innen gaben an, nicht an einer diagnostizierten Sehschwäche zu leiden, 3 trugen eine Brille oder Kontaktlinsen und 2 der Proband:innen gaben an, zwar unter einer Sehschwäche zu leiden, aber keine Korrigierung dafür zu nutzen. Ansonsten gaben die Teilnehmer:innen keine Vorerkrankungen an, mit Ausnahme einer Person, die äußerte unter einer Rot-Grün-Schwäche zu leiden. 70% der Proband:innen hatten schon einmal eine virtuelle Realitätserfahrung mithilfe einer Datenbrille erlebt. Die angegebene Erfahrung mit stereoskopischem 3D und Computerspielen ist aus Abbildung 6.1 ersichtlich.

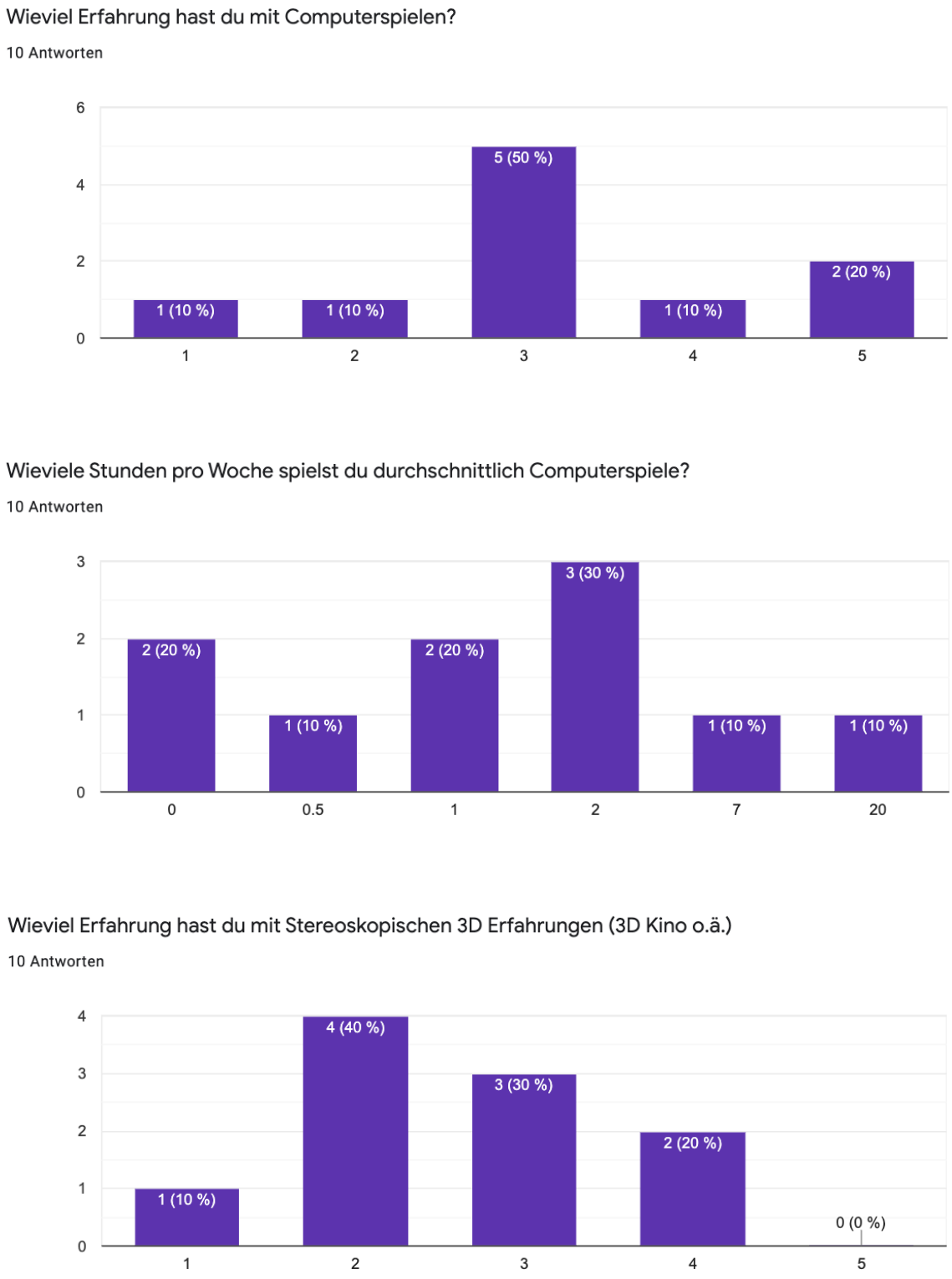

Abbildung 6.1: Angaben der Teilnehmenden zur Erfahrung mit Computerspielen und stereoskopischem 3D. Bei allen drei Diagrammen gilt: X-Achse: Antworten der Teilnehmenden. Y-Achse: Häufigkeit der Antworten.

6.3 Ablauf der Studie

An dieser Stelle werde ich den genauen Ablauf der Studie beschreiben. Die Studie wurde zwischen dem 12.08.2021 und dem 20.08.2021 in einer geräumigen Wohnung in Hamburg/Stellingen ausgeführt. Nach Ankunft der Teilnehmer:innen wurden zunächst die notwendigen Covid-Sicherheitsmaßnahmen thematisiert. Anschließend haben diese sich ihre Hände desinfiziert, danach führten sie einen Antigen-Schnelltest an sich aus. Bis zur Feststellung des Testergebnisses wurden den Proband:innen in die Steuerung von der Joystick- und Teleportationsbedingung eingewiesen. Zudem wurde ihnen erklärt, dass ihre Aufgabe während des Experiments darin bestünde, die zurückgelegte Entfernung in jeder Bedingung einzuschätzen. Dabei wurde betont, dass es nicht um die Luftlinie zwischen Start und Ziel ging, sondern um die Strecke, die sie insgesamt zurücklegten. Zudem sollten sie von Raum zu Raum gelangen, indem sie an der Tür zum nächsten Raum die erforderlichen Zahlencodes eingeben, bis die Tür sich öffnet.

Nachdem das negative Testergebnis abzulesen war (bei keiner Teilnehmer:in gab es ein positives Testergebnis), wurden die Proband:innen einer der Gruppen zugeteilt. Ihnen wurde der erste Abschnitt des im Anhang abgebildeten Demographiefragebogens auf einem Laptop bereit gestellt. Nachdem dieser ausgefüllt war, durften die Proband:innen das erste Mal die für die Durchführung der Studie genutzte Oculus Quest 2 Datenbrille aufsetzen, um sich kurz damit vertraut zu machen. Nebenbei wurde das Bild der Brille über einen Google-Chromecast angezeigt, sodass es von außen möglich war zu sehen, was in der Datenbrille dargestellt wurde. Den Teilnehmer:innen wurde erklärt, wie sie innerhalb des Oculus Startbildschirm navigieren können, sodass diese selbst die APK für die Studie öffnen konnten (siehe Abbildung 6.2).

Nachdem diese geöffnet wurde (und der Unity-Ladebildschirm abgeklungen war), befanden die Teilnehmer:innen sich im virtuellen Menü der Studie (siehe Abbildung 3.11). Im Menü der Studie war die Teleportier-Fortbewegungsart aktiviert, damit die Teilnehmer:innen unabhängig der Reihenfolge, in der sie später die drei Bedingungen durchliefen, schon einmal mit dieser vertraut waren. Dort befand sich auch das Versuchsbedingungseingabefeld, mit dem die Teilnehmenden sowohl ausprobieren konnten, wie die späteren Nummerneingabefelder funktionieren, als auch die Studienbedingung auswählen konnten. Dafür musste zuerst die entsprechende Taste ($A1 - C3$, je nach Bedingung wie in Abschnitt 5.2 definiert) und danach die Bestätigungstaste # gedrückt werden, mit der die Bedingung augenblicklich gestartet wurde. Den Proband:innen wurde, abhängig von ihrer Gruppe, mitgeteilt, welche Bedingung sie ausführen sollen. Sie starteten sie diese und führten sie aus. Nach Beendigung der Versuchsbedingung wurde für die Proband:innen auf dem Laptop der, auch im Anhang abgebildete, Durchlauffragebogen geöffnet, den diese dann ausfüllten. Bei diesem wurde zuerst nach der Distanzschätzung gefragt und dann mithilfe des SUS-Questionaires nach dem Präsenzgefühl.

Das Öffnen, das Durchführen der Bedingung und das nachfolgende Ausfüllen des Fragebogens wurde für jede der drei Bedingungen in der entsprechenden Reihenfolge durchgeführt. Danach wurde von mir überprüft, ob die Daten, die am Ende einer Bedingung

Abbildung 6.2: Das “Home”-Menü des Betriebssystems der Oculus Quest 2 Datenbrille, auf der der Versuch ausgeführt wurde. Nutzer:innen starteten hier eigenständig die Anwendung, in der die Studie durchgeführt wurde.

an die Firebase-Datenbank versendet werden, angekommen sind, um im Falle eines Fehlers gegebenenfalls einzelne Bedingungen zu wiederholen. Dies ist erfreulicherweise nicht notwendig gewesen. Zusätzlich dazu wurde die aktuelle VP-ID händisch von mir in die neu hinzugekommenen Daten eingetragen, um die Ergebnisse später eindeutig zuordnen zu können.

Zuletzt wurden die Proband:innen gebeten, die letzten noch fehlenden Daten in den Abschnitt 2 des Demographiefragebogens zu füllen.

Bei der Versuchsperson mit der ID 2, gab es leider einen kleinen Fehler mit der Wandgenerierung, sodass die zuerst von ihr durchgeführte Versuchsbedinung $C2$ abgebrochen und wiederholt wurde. Im Anschluss wurde der Fehler, der nur in seltenen Fällen auftrat, von mir aus dem Quelltext beseitigt.

Bei der Versuchsperson mit der ID 10 stellte sich kurz nach Beendigung des Experiments heraus, dass diese aufgrund eines Missverständnisses die Frage nach der Distanzschätzung falsch interpretiert hatte. Bei den statistischen Untersuchungen wurden die gesammelten Daten die der ID 10 zugehörig waren folglich nicht mitgenutzt.

6.4 Auswertung der Ergebnisse

Die Daten aus den Google-Forms Formularen und die gemessenen Daten aus der Firebase Datenbank wurden jeweils anonym hinterlegt. Die Datensätze lassen sich einander

lediglich über eine Versuchspersonen-ID, die Bedingung, und die Levellänge zuordnen. Um die Ergebnisse der beiden Quellen in einer gemeinsame Tabelle zu vereinen, habe ich die Antworten auf das "Google-Forms"-Formular in einer "Google-Tabellen"-Tabelle gesammelt und dann die Messergebnisse aus der Datenbank dazugetragen. Die Zuordnung erfolgte über die Versuchspersonen-ID, die Fortbewegungsbedingung und die Levellänge, die sich in beiden Datenquellen finden ließen und in der Kombination eindeutig einander zuordbar sind. Mit der Rechnung 6.1 berechnete ich eine neue Spalte für die Differenz. Die Daten habe ich dann als *.csv* Datei exportiert, um die weitere Datenverarbeitung und statistische Auswertung in R durchzuführen. Dafür habe ich die R Version "stable 4.1.1" genutzt. Die statistische Auswertung dieser Daten, zunächst für die Schätzung der zurückgelegten Strecke, dann für das Präsenzgefühl werde ich in den folgenden Abschnitten vorstellen.

Alle Antworten aus den Fragebögen und die in der Datenbank gespeicherten Mess-Daten sind im "Anhang - Fragebögen und Daten" abgebildet. Letztere wurden von mir der Übersicht wegen vom *.json* Format in ein Tabellenformat umformatiert.

6.4.1 Schätzung der zurückgelegten Distanz

An dieser Stelle werde ich die statistische Auswertung zur Schätzung der zurückgelegten Strecke vorstellen. Dabei untersuche ich die Unterschiede zwischen geschätzter und gemessener Distanz mit einer einfaktoriellen Varianzanalyse auf den Einfluss der Fortbewegungsart.

Die Qualität der Schätzung der Proband:innen wird über die absolute Differenz zwischen gemessener Strecke und geschätzter Strecke folgendermaßen, berechnet:

$$\mid D_g - D_m \mid \tag{6.1}$$

D_g steht für die geschätzte Distanz und D_m für die gemessene Distanz.

Um die Unterschiede der errechneten Differenzen zwischen geschätzten und gemessenen zurückgelegten Distanzen zu vergleichen und auf ihre Signifikanz zu testen, bietet sich eine Varianzanalyse an. Aufgrund des "Within-Subjects Design" wird eine einfaktorielle Varianzanalyse mit Messwiederholung durchgeführt.

Arithmetische Mittel und Standardabweichungen der absoluten Differenz von Messung und Schätzung der zurückgelegten Distanz sind in Tabelle 6.1 abgebildet.

Ein Boxplot der Messergebnisse findet sich in Abbildung 6.3

Eine Varianzanalye mit Messwiederholung setzt voraus, dass die einzelnen Stufen des "Within-Subject Factor", in diesem Fall die Fortbewegungsart, normalverteilt sind. Diese sind in in Form eines Quantil-Quantil-Diagramms in Abbildung 6.4 abgebildet.

Zudem wurde ein Shapiro-Wilk Test für die einzelnen Faktorstufen durchgeführt (Ergebnisse in Tabelle 6.2). Dieser deutete nicht darauf hin, dass die Annahme der Normalverteilung verletzt worden sei.

Fortbewegungsart	n	M	SD
Redirection	9	16,9	12,0
Joystick	9	16,0	16,8
Teleport	9	13,8	9,62

Tabelle 6.1: Arithmetische Mittel und Standardabweichungen der absoluten Differenz zwischen gemessener und geschätzter zurückgelegter Distanz

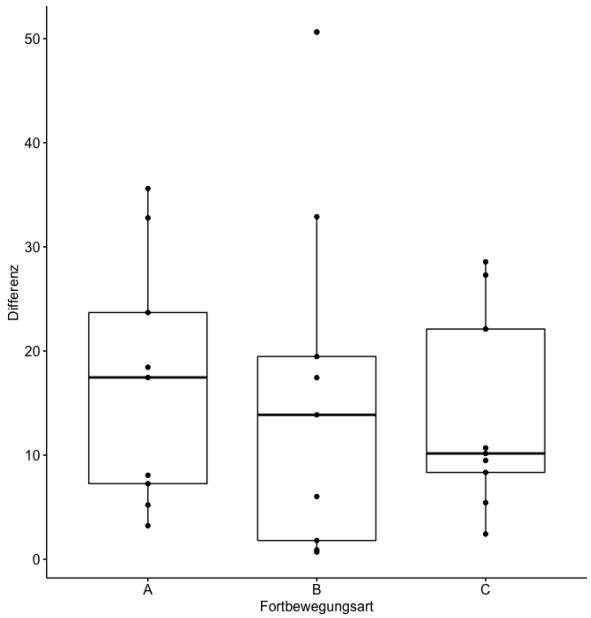

Abbildung 6.3: Box-Plot der absoluten Differenzen zwischen gemessener und geschätzter zurückgelegter Distanz, aufgeteilt nach Fortbewegungsart. A steht für Real-Walking, B für Joystick Steuerung und C für Teleportation

Bedingung	Teststatistik	p
Real-Walking	0,908	0,300
Joystick	0,868	0,117
Teleport	0,867	0,113

Tabelle 6.2: Die Ergebnisse eines Shapiro-Wilk Tests der absoluten Differenz von Schätzung und Messung der zurückgelegten Distanz für die unterschiedlichen Fortbewegungsarten

Im Datensatz wurden keine extremen Ausreißer festgestellt.

Mit den erhobenen Daten wurde eine Varianzanalyse mit Messwiederholung zur Feststellung signifikanter Unterschiede zwischen den Fortbewegungsarten zum $\alpha = 5\%$ Level

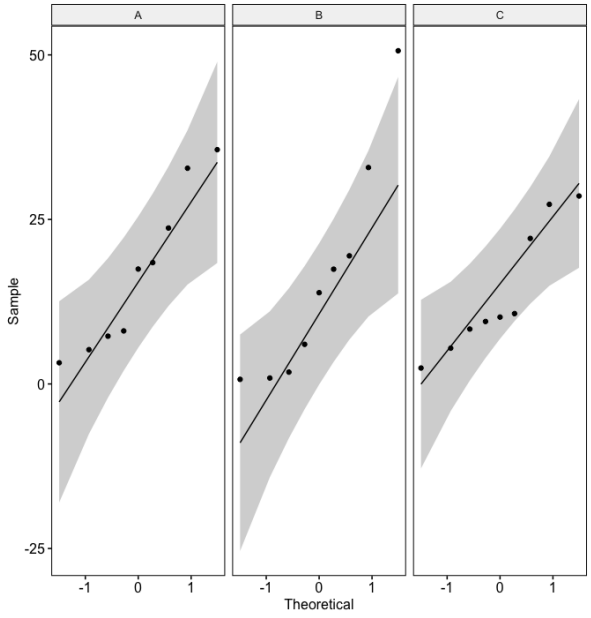

Abbildung 6.4: Korrelation zwischen Messergebnissen und einer Normalverteilung in einem Quantil-Quantil-Diagramm, nach Fortbewegungsart getrennt. A steht für Real-Walking, B für Joystick Steuerung und C für Teleportation

durchgeführt.

Ein Mauchly's Test auf Sphärizität zeigte keine Verletzung dieser Bedingung auf.

Die Varianzanalyse erfasste keine signifikanten Unterschiede zwischen den Fortbewegungsarten: $F(2, 16) = 0,207$ $p = 0,815$ $\eta^2 = 0,0104$

Post-Hoc Tests zur genaueren Bestimmung der Unterschiede waren folglich nicht erforderlich.

6.4.2 Präsenzgefühl

Ob die Unterschiede der Ergebnisse des SUS-Präsenz-Questionaire signfikant sind wird im Folgenden ermittelt werden. Dabei werden die "SUS-Scores" der Durchläufe verglichen. Der SUS-Score ermittelt sich über die Anzahl der Fragen, die mit einem der Werte 6 oder 7, also den beiden Werten, die am meisten erlebte Präsenz beschreiben, beantwortet wurden.

Für jeden Versuchsdurchlauf wurde der SUS-Score berechnet. Die arithmetischen Mittel für die einzelnen Fortbewegungsarten sind in Tabelle 6.3 ersichtlich.

Abbildung 6.5 Stellt die SUS-Scores als Boxplot dar.

Auch für die Präsenz-Messdaten wurden keine extremen Ausreißer gefunden.

Bedingung	n	M	SD
Real-Walking	9	2,67	2
Joystick	9	1,56	2,13
Teleport	9	1	1,12

Tabelle 6.3: Arithmetische Mittel und Standardabweichungen der SUS-Scores der Teilnehmenden bei den verschiedenen Fortbewegungsbedingungen

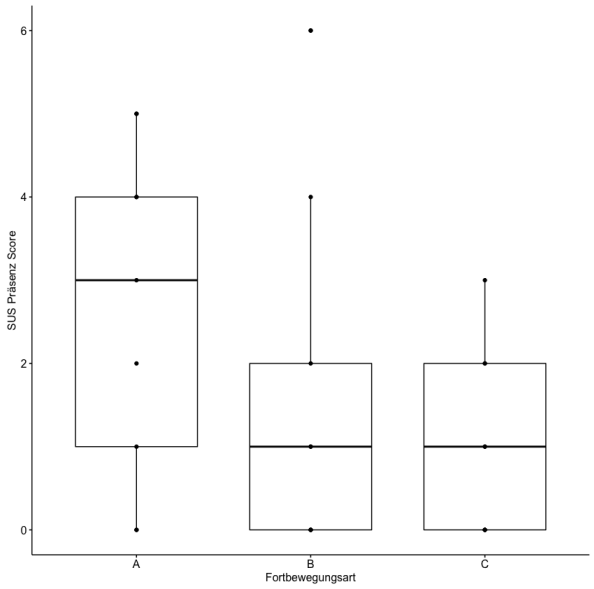

Abbildung 6.5: Box-Plot der SUS-Scores nach Fortbewegungsart. A steht für Real-Walking, B für Joystick Steuerung und C für Teleportation

Die Abbildung 6.6 stellt die Korrelation zwischen den Daten und einer Normalverteilung für jede der Fortbewegungsarten dar in einem Quantil-Quantil-Diagramm dar. Die Messpunkte in den Kategorien "Joystick" und "Teleportation" folgen nicht der Referenzlinie.

Um Klarheit zu schaffen, wurde ein Shapiro-Wilk-Test zum $\alpha = 5\%$ Signifikanzniveau durchgeführt. Dieser weist darauf hin, dass eine Normalverteilung bei der Bedingung Joystick ($p = 0,0125$) nicht angenommen werden kann. Die Ergebnisse des Shapiro-Wilk Tests sind in Tabelle 6.4 zu sehen.

Da die Bedingungen einer Varianzanalyse für die Daten zur Präsenzerhebung nicht eingehalten werden können, wurde stattdessen ein Friedmann Test zur Feststellung signifikanter Unterschiede zwischen den Fortbewegungsarten zum $\alpha = 5\%$ Niveau durchgeführt, da dieser keine normalverteilte Daten voraussetzt. Dieser zeigte signifikante Unterschiede zwischen den Fortbewegungsarten: $\chi^2(2) = 6,95$ $p = 0,0309$

Die Effektstärke wurde mithilfe einer Kendall'schen Konkordanzanalyse errechnet. Da-

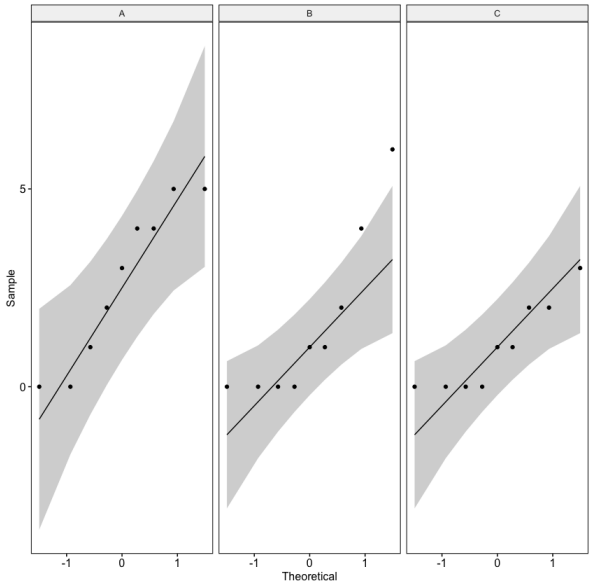

Abbildung 6.6: Quantil-Quantil-Diagram der Korrelationen von SUS-Scores und Normalverteilung, nach Fortbewegungsart getrennt. A steht für Real-Walking, B für Joystick Steuerung und C für Teleportation

Bedingung	Teststatistik	p
Real-Walking	0,894	0,221
Joystick	0,782	0,0125
Teleport	0,844	0,0648

Tabelle 6.4: Ergebnisse eines Shapiro-Wilk Tests der verschiedenen Faktorstufen

bei ergab sich eine Effektstärke von 0,386, dies gilt nach J.Cohen als moderater Effekt [7].

Dennoch wurden bei einer paarweise durchgeführten Post-Hoc Analyse der Präsenzdaten mit einem Wilcoxon-Vorzeichen-Rang-Test mit Bonferroni-Korrektur der p-Werte keine signifikanten Unterschiede zwischen den jeweiligen Paaren von Fortbewegungsarten festgestellt. (Real-Walking mit Joystick: $p = 0,402$, Real-Walking mit Teleport: $p = 0,105$, Joystick mit Teleport: $p = 0,807$) Die Ergebnisse sind in Tabelle 6.5 dargestellt.

Bedingung 1	Bedingung 2	Teststatistik	p	p korrigiert
Real-Walking	Joystick	13,5	0,134	0,402
Real-Walking	Teleport	21	0,035	0,105
Joystick	Teleport	8,5	0,269	0,807

Tabelle 6.5: Ergebnisse der Post-Hoc Analyse: Paarweise-Untersuchung mit Wilcoxon-Vorzeichen-Rang-Test für gepaarte Datensätze. Die korrigierten p-Werte sind mithilfe der Bonferroni Korrektur korrigiert

Kapitel 7

Diskussion, Ausblick und Konklusion

In diesem letzten Kapitel werde ich die im vorherigen Kapitel errechneten Ergebnisse diskutieren und einordnen. Damit verbunden werde ich einen Ausblick formulieren, wie zukünftige Forschungsarbeiten auf diese Arbeit aufbauen sollten. Anschließend werde ich eine abschließende Zusammenfassung der in dieser Arbeit vorgestellten Ideen, Umsetzungen und Erkenntnissen geben.

7.1 Diskussion und Ausblick

Die Ergebnisse liefern leider keine Unterstützung der vorgestellten Hypothesen. Dies kann verschiedene Gründe haben. An dieser Stelle werde ich die Ergebnisse einordnen und die Gründe herauszuarbeiten. Vorweg möchte ich betonen, dass die für diese Pilotierungsstudie erfasste Datenlage, nicht hinreichend gewesen wäre um konfidente Ergebnisse zu präsentieren. Es ist zudem durchaus möglich, dass mit einer größeren Teilnehmerzahl Effekte feststellbar gewesen wären, die die Hypothesen bestätigt hätten.

7.1.1 Schätzung der zurückgelegten Distanz

Die oben formulierte Hypothese zur Schätzung der zurückgelegten Distanz lautete wie folgt:

Nachdem ein mit der hier vorgestellten Levelgenerierungmethode erstelltes Level mit der Fortbewegungsart “Real-Walking” durchquert wurde, ist die Fähigkeit, die darin zurückgelegte Strecke einzuschätzen, besser als wenn es mit den alternativen Fortbewegungsarten “Joystick” und “Teleportation” durchquert wurde.

Die in der hier vorgestellten Pilotierungsstudie erhobenen Daten konnten diese Hypothese leider nicht stützen, die Gegenhypothese konnte nicht verworfen werden. Die absolute Differenz zwischen geschätzter und gemessener zurückgelegter Distanz ist bei der Fortbewegungsart “Real-Walking” nicht geringer als bei den Alternativen. Es konnten keine signifikanten Unterschiede bei den Fortbewegungsbedingungen erfasst werden.

Im Folgenden arbeite ich einige mögliche Gründe dafür heraus.

In jedem der Räume ist es notwendig, mithilfe eines Rotation-Gains die virtuelle Umgebung 90° um die reale Umgebung zu rotieren. Auch wenn dieser Effekt aufgrund der Subtilität des Rotation-Gains nicht bemerkt wird, könnte dieser Effekt dennoch für eine starke Beeinträchtigung der Orientierungsfähigkeit der Nutzer:innen sorgen. Mehrere Proband:innen gaben während der Real-Walking Versuchsbedingung mit der Fortbewegungsmethode "Real-Walking" verbal an, schon nach einem Raum die Orientierung verloren zu haben, wo im realen Trackingspace sie sich gerade befinden. Dies sollte nicht überraschen, da der virtuell zurückgelegte Pfad ohne Redirection-Maßnahmen aufgrund der Trackingspace-Grenzen nicht in der realen Umgebung möglich gewesen wäre. Ob dieser Effekt tatsächlich das kognitive Raumverständnis der virtuellen Umgebung erschwert, sollte in zukünftigen Arbeiten konkreter untersucht werden. Dabei sollte die zusätzliche Redirection Komponente der durch die Wandgenerierung entstehenden Impossible-Spaces herausgenommen werden, da diese für zusätzliche Verwirrung sorgen könnten.

Ein weiterer möglicher Grund dafür, wieso diese Hypothesen nicht unterstützt werden konnten, ist die Monotonie der generierten Räume. Die Räume unterscheiden sich voneinander nur unwesentlich, nämlich in Form der Seite des Raumes, an die der Korridor zum nächsten Raum generiert wird. Es liegt nahe, dass Nutzer:innen eine bessere kognitive Karte der Umgebung erstellen könnten, wenn sie gewisse Anhaltspunkte hätten, an die sie sich erinnern können. Dies lässt sich begrenzt verbessern, indem bei der Levelgenerierung in die Räume vordefinierte Objekte platziert werden. Dies ist ein Grundproblem von generierten (generischen) Leveln.

In diesem Fall könnte es also so sein, dass sich die Fähigkeit von Nutzer:innen, die zurückgelegte Distanz einzuschätzen, zwischen von Menschen gestalteten und von Computern generierten Leveln signifikant unterscheidet. Vor dem Hintergrund dieser Überlegung könnte ein Versuchsaufbau gestaltet werden, bei dem die kognitive Kartenbildung in menschlich gestalteten Leveln mit der in prozedural generierten Leveln verglichen wird.

Ein weiterer möglicher Grund dafür, wieso diese Hypothesen nicht unterstützt wird, kann sein, dass die vorher getroffene Annahme, die Schätzung der zurückgelegten Distanz nähme mit besserem Raumverständnis zu, nicht zutrifft. Zwar weist eine Post-Hoc Untersuchung von Peck et al. [17] darauf hin, dass Versuchspersonen bei einer Real-Walking Fortbewegungsbedingung die Größe der virtuellen Umgebung besser einschätzen können als bei alternativen virtuellen Fortbewegungsarten, allerdings bedeutet dies auch nicht zwingend, dass die Versuchspersonen ihre eigene zurückgelegte Distanz über mehrere Räume hinweg besser einschätzen können, als in anderen Bedingungen.

In diesem Fall gäbe es mit der Real-Walking Fortbewegungsart zwar ein besseres Raumverständnis der Nutzer:innen, allerdings wäre diese von der hier angewandten Methode unentdeckt geblieben, da sie keine Auswirkungen darauf hat, wie gut die Schätzungen sind. Aus diesem Grund sollte in zukünftigen Studien mit unterschiedlichen Methoden gemessen werden, wie gut das Raumverständnis der Proband:innen ist.

Alternative Möglichkeiten, das Raumverständnis zu messen, sind beispielsweise, die Nut-

zer:innen zu bitten, nach dem Durchschreiten des Levels eine Karte von diesem zu zeichnen und die Genauigkeit der Karte dann von einer Jury bewerten zu lassen. Eine andere Möglichkeit wäre es, während des Levels Objekte zu platzieren, auf die, die Proband:innen später zeigen müssen und dann die Differenz der gezeigten Winkel mit den gemessenen Winkeln zu vergleichen. (siehe [10, 17]). Es wäre dann sinnvoll, die prozedural generierten Level zu speichern und zu exportieren, sodass die Beschreibungen der Proband:innen mit den wirklich generierten Leveln verglichen werden können. Bei Leveln, die mit der hier vorgestellten Levelgenerierungsmethode erstellt wurden, wäre es simpel, dies zu lösen, indem der Algorithmus so erweitert würde, dass die in Abschnitt 4.4 beschriebenen Variablen ex- und importierbar gemacht würden.

7.1.2 Präsenzgefühl in der virtuellen Umgebung

Die oben formulierte Hypothese zum Präsenzgefühl lautete:

Beim Durchqueren der mit der hier vorgestellten Levelgenerierungmethode erstellten Level mit der "Real-Walking" Fortbewegungsart haben Nutzer:innen ein größeres Präsenzgefühl als mit den beiden Alternativen "Joystick" und "Teleportation".

Der Durchschnittswert der SUS-Präsenz-Scores ist bei der Real-Walking Testbedingung $(2,67)$ ist deutlich größer als bei der Joystick-$(1,56)$ oder Teleportationsbedingung (1). Leider kann für diese Hypothese in den hier erhobenen Daten dennoch keine Unterstützung gefunden werden. Zwar weist ein statistischer Test der Präsenz-Scores der drei Fortbewegungsarten auf signifikante Unterschiede hin, allerdings lassen sich in den darauf folgenden Tests, in denen die 3 Fortbewegungsarten paarweise miteinander verglichen werden keine signifikanten Unterschiede feststellen.

Ein möglicher Grund für diese ambivalenten Ergebnisse ist die Datenlage, aus der diese Ergebnisse entstanden, die um einiges weniger gut ist, als sie in einer offiziellen Studie gewesen wäre. Da eine Normalverteilung der Daten nicht angenommen werden konnte, konnten nur nicht-parametrische statistische Tests verwendet werden. Es ist davon auszugehen, dass die Ergebnisse im Falle einer größeren Datenmenge deutlich klarer gewesen wären.

Folglich ist es wünschenswert, diese Frage in einem größerem Kontext genauer zu untersuchen.

7.2 Konklusion

In dieser Arbeit wurde eine Methode zur prozeduralen Generierung von Raumbasierten Leveln für virtuelle Umgebungen vorgestellt, bei der Nutzer:innen mithilfe von den Redirection-Techniken "Rotation-Gains" und "Impossible-Spaces" ermöglicht wird, sich mit natürlichem Gehen auf eine Weise von Raum zu Raum fortzubewegen, bei der der

nicht begehbare Bereich der virtuellen Umgebung immer wieder transformiert wird, ohne dass einem dies bewusst wird. Es entsteht die Illusion, die Nutzer:in könnte unabhängig von ihrem realen Trackingspace die virtuelle Umgebung erkunden. Diese Methodik ermöglicht potentiell endlos lange Level, bei denen die Generierung während der Erkundung des Levels fortgeführt wird.

Anschließend wurde eine informelle Pilotierungsstudie vorgestellt und durchgeführt, bei der drei verschiedene Fortbewegungsarten (Joysticksteuerung, Teleportierung und Real-Walking mit den eben genannten Redirection-Techniken) in, mit dieser Methode generierten, Leveln verglichen wurden. Dabei wurde versucht Vorteile zu zeigen, die die Ermöglichung natürlichen Gehens in virtuellen Umgebungen bieten könnte, nämlich eine vereinfachte Schätzung zurückgelegter Distanzen und ein erhöhtes Präsenzgefühl für Nutzer:innen. Dies könnte sowohl an der - den Umständen geschuldeten - geringen Datenlage von 10 Versuchspersonen liegen, als auch daran, dass diese Vorteile nicht in mit dieser Methode prozedural generierten Leveln gegeben sind.

Weiterführende Studien auf Basis einer größeren Datenmenge sind notwendig, um die Vorteile der Ermöglichung natürlichen Gehens in zufallsgenerierten virtuellen Umgebungen mithilfe von Redirection-Techniken realistisch einzuschätzen und um in dieser Arbeit eröffneten Fragen weiter auf den Grund zu gehen.

Danksagungen

An dieser Stelle möchte ich mich bei Allen bedanken, die mich während diese Projekts unterstützt und motiviert haben. Herzlicher Dank gebürht bei all meinen Freunden und Verwandten, die freundlicherweise an der Studie teilgenommen haben! Ganz besonderes dankbar bin ich Jutta Dalladas-Djemai, für das ausgiebige Korrekturlesen dieser Arbeit. Außerdem möchte ich mich bei meinem Betreuer Dr. Eike Langbehn bedanken, dessen Expertise und Gedult mir die Arbeit an diesem Projekt erleichtert hat.

Bibliographie

[1] Gerd Bruder, Fernando Argelaguet Sanz, Anne-Helene Olivier und Anatole Lecuyer. “Distance estimation in large immersive projection systems, revisited”. In: (2015), Seiten 27–32. DOI: 10.1109/VR.2015.7223320.

[2] Sarah Chance, Florence Gaunet und Andrew Beall. “Locomotion Mode Affects the Updating of Objects Encountered During Travel: The Contribution of Vestibular and Proprioceptive Inputs to Path Integration”. In: *Presence* 7 (Apr. 1998), Seiten 168–178. DOI: 10.1162/105474698565659.

[3] Lung-Pan Cheng, Eyal Ofek, Christian Holz und Andrew D. Wilson. “VRoamer: Generating On-The-Fly VR Experiences While Walking inside Large, Unknown Real-World Building Environments”. In: *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*. 2019, Seiten 359–366. DOI: 10.1109/VR.2019.8798074.

[7] Jacob Cohen. *Statistical power analysis for the behavioral sciences*. Routledge, 1988. ISBN: 9780805802832.

[8] Yasin Farmani und Robert J. Teather. “Viewpoint Snapping to Reduce Cybersickness in Virtual Reality”. In: *Proceedings of the 44th Graphics Interface Conference*. GI ’18. Toronto, Canada: Canadian Human-Computer Communications Society, 2018, Seiten 168–175. ISBN: 9780994786838. DOI: 10.20380/GI2018.23. URL: https://doi.org/10.20380/GI2018.23.

[9] Erich Gamma, Richard Helm, Ralph Johnson und John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. USA: Addison-Wesley Longman Publishing Co., Inc., 1995. ISBN: 0201633612.

[10] Eike Langbehn, Paul Lubos und Frank Steinicke. “Evaluation of Locomotion Techniques for Room-Scale VR: Joystick, Teleportation, and Redirected Walking”. In: *Proceedings of the Virtual Reality International Conference - Laval Virtual*. VRIC ’18. Laval, France: Association for Computing Machinery, 2018. ISBN: 9781450353816. DOI: 10.1145/3234253.3234291. URL: https://doi.org/10.1145/3234253.3234291.

[11] Eike Langbehn, Paul Lubos und Frank Steinicke. “Redirected Spaces: Going Beyond Borders”. In: *2018 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*. 2018, Seiten 767–768. DOI: 10.1109/VR.2018.8446167.

[12] Eike Langbehn und Frank Steinicke. “Redirected Walking in Virtual Reality”. In: März 2018.

[13] Jr. Joseph LaViola. “A discussion of cybersickness in virtual environments”. In: *ACM SIGCHI Bulletin* 32 (Jan. 2000), Seiten 47–56. DOI: 10.1145/333329.333344.

[14] Sebastian Marwecki und Patrick Baudisch. “Scenograph: Fitting Real-Walking VR Experiences into Various Tracking Volumes”. In: *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*. UIST ’18. Berlin, Germany: Association for Computing Machinery, 2018, Seiten 511–520. ISBN: 9781450359481. DOI: 10.1145/3242587.3242648. URL: https://doi.org/10.1145/3242587.3242648.

[17] Tabitha C. Peck, Henry Fuchs und Mary C. Whitton. “An evaluation of navigational ability comparing Redirected Free Exploration with Distractors to Walking-in-Place and joystick locomotio interfaces”. In: *2011 IEEE Virtual Reality Conference*. 2011, Seiten 55–62. DOI: 10.1109/VR.2011.5759437.

[18] Sharif Razzaque, Zachariah Kohn und Mary C. Whitton. “Redirected Walking”. In: *Eurographics 2001 - Short Presentations*. Eurographics Association, 2001. DOI: 10.2312/egs.20011036.

[19] Rebekka S. Renner, Boris M. Velichkovsky und Jens R. Helmert. “The Perception of Egocentric Distances in Virtual Environments - A Review”. In: *ACM Comput. Surv.* 46.2 (Dez. 2013). ISSN: 0360-0300. DOI: 10.1145/2543581.2543590. URL: https://doi.org/10.1145/2543581.2543590.

[20] Roy A. Ruddle und Simon Lessels. “The Benefits of Using a Walking Interface to Navigate Virtual Environments”. In: *ACM Trans. Comput.-Hum. Interact.* 16.1 (Apr. 2009). ISSN: 1073-0516. DOI: 10.1145/1502800.1502805. URL: https://doi.org/10.1145/1502800.1502805.

[21] Roy A. Ruddle, Ekaterina Volkova und Heinrich H. Bülthoff. “Walking Improves Your Cognitive Map in Environments That Are Large-Scale and Large in Extent”. In: *ACM Trans. Comput.-Hum. Interact.* 18.2 (Juli 2011). ISSN: 1073-0516. DOI: 10.1145/1970378.1970384. URL: https://doi.org/10.1145/1970378.1970384.

[22] Mel Slater, Martin Usoh und Anthony Steed. “Taking steps: The influence of a walking technique on presence in virtual reality”. In: *ACM Transactions on Computer-Human Interaction (TOCHI)* 2 (Dez. 1995), Seiten 201–219. DOI: 10.1145/210079.210084.

[24] Frank Steinicke, Gerd Bruder, Jason Jerald, Harald Frenz und Markus Lappe. “Estimation of Detection Thresholds for Redirected Walking Techniques”. In: *IEEE Transactions on Visualization and Computer Graphics* 16.1 (2010), Seiten 17–27. DOI: 10.1109/TVCG.2009.62.

[25] Evan A. Suma, Zachary Lipps, Samantha Finkelstein, David M. Krum und Mark Bolas. “Impossible Spaces: Maximizing Natural Walking in Virtual Environments with Self-Overlapping Architecture”. In: *IEEE Transactions on Visualization and Computer Graphics* 18.4 (2012), Seiten 555–564. DOI: 10.1109/TVCG.2012.47.

[26] Evan Suma Rosenberg, Gerd Bruder, Frank Steinicke, David Krum und Mark Bolas. “A taxonomy for deploying redirection techniques in immersive virtual environments”. In: *Proceedings - IEEE Virtual Reality* (März 2012), Seiten 43–46. DOI: 10.1109/VR.2012.6180877.

[27] Julian Togelius, Emil Kastbjerg, David Schedl und Georgios Yannakakis. “What is Procedural Content Generation? Mario on the borderline”. In: (Jan. 2011). DOI: 10.1145/2000919.2000922.

[28] Julian Togelius, Georgios Yannakakis, Kenneth Stanley und Cameron Browne. “Search-Based Procedural Content Generation”. In: Apr. 2010, Seiten 141–150. ISBN: 978-3-642-12238-5. DOI: 10.1007/978-3-642-12239-2_15.

[31] Martin Usoh, Kevin Arthur, Mary Whitton, Rui Bastos, Anthony Steed, Mel Slater und Frederick Brooks Jr. “Walking > Walking-in-Place > Flying, in Virtual Environments”. In: *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques ACM* (Juni 1999). DOI: 10.1145/311535.311589.

[32] Martin Usoh, Ernest Catena, Sima Arman und Mel Slater. “Using Presence Questionnaires in Reality”. In: *Presence: Teleoperators and Virtual Environments* 9 (Apr. 2000). DOI: 10.1162/105474600566989.

[33] Khrystyna Vasylevska, Hannes Kaufmann, Mark Bolas und Evan A. Suma. “Flexible spaces: Dynamic layout generation for infinite walking in virtual environments”. In: *2013 IEEE Symposium on 3D User Interfaces (3DUI)*. 2013, Seiten 39–42. DOI: 10.1109/3DUI.2013.6550194.

Electronic

[4] Jarosław (Void Room) Ciupiński. *Open world with impossible spaces*. Apr. 2019. URL: https://void-room.itch.io/tea-for-god/devlog/76304/procedural-level-generation-using-impossible-spaces (besucht am 01. 07. 2021).

[5] Jarosław (Void Room) Ciupiński. *Open world with impossible spaces*. März 2021. URL: https://void-room.itch.io/tea-for-god/devlog/235914/open-world-with-impossible-spaces (besucht am 01. 07. 2021).

[6] Jarosław (Void Room) Ciupiński. *Tea for God*. URL: https://void-room.itch.io/tea-for-god (besucht am 01. 07. 2021).

[15] *Oculus Quest 2 Website*. URL: https://www.oculus.com/quest-2/ (besucht am 07. 07. 2021).

[16] *Oculus Quest 2 Website*. URL: https://developer.oculus.com/downloads/package/unity-integration/ (besucht am 07. 07. 2021).

[23] *Space Extender*. URL: https://github.com/curvaturegames/space-extender (besucht am 11. 01. 2021).

[29] *Unity Asset Store: Dungeon Ground Texture*. URL: https://assetstore.unity.com/packages/2d/textures-materials/floors/dungeon-ground-texture-33296 (besucht am 14. 09. 2021).

[30] *Unity Website*. URL: https://unity.com/ (besucht am 07. 07. 2021).

Anhang - Fragebögen und Daten

Demographiefragebogen

Abschnitt 1 - Zu Beginn

F0: Wie ist deine VP-Nummer? *

F1: Wieviel Erfahrung hast du mit Computerspielen? *

F2: Wieviele Stunden pro Woche spielst du durchschnittlich Computerspiele? *

F3: Wieviel Erfahrung hast du mit Stereoskopischen 3D Erfahrungen (3D Kino o.ä.) *

F4: Hast du schonmal eine VR-Brille genutzt? *

F5 Hast du eine Sehschwäche, wenn ja trägst du eine Brille/Kontaktlinsen * single choice - 0: Kaum/Keine Sehschwäche - 1: Sehschwäche, aber Brille/Kontaktlinsen - 2: Nicht-Korrigierte Sehschwäche

F6 Hast du eine Gleichgewichtsstörung? *

F7 Hast du sonstige Krankheiten oder gesundheitliche Probleme die deine Sicht beschränken? *

Abschnitt 2 - Zuletzt

F8 Wie groß bist du? *

F9 Wie ist dein Geschlecht? Männlich, Weiblich, sonstiges

F10 Wie alt bist du? *

Antworten Demographiefragebogen:

Zeitstempel	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
18.08.2021 17:06:19	5	1	0	1	Nein	1	Nein	Nein	162	Weiblich	63
20.08.2021 00:27:02	10	2	0	3	Ja	2	Nein	Nein	171	Weiblich	23
17.08.2021 02:26:03	4	3	0,5	2	Nein	0	Nein	Nein	180	Männlich	23
19.08.2021 17:03:41	8	3	1	2	Ja	0	Nein	Nein	184	Männlich	20
19.08.2021 23:07:13	9	3	1	4	Ja	1	Nein	Rot-Grün-Schwäche	185	Männlich	23
13.08.2021 15:48:28	3	3	2	4	Ja	0	Nein	Nein	189	Männlich	24
18.08.2021 18:16:54	6	3	2	3	Nein	1	Nein	Nein	172	Männlich	69
18.08.2021 20:19:23	7	4	2	3	Ja	2	Nein	Nein	189	Männlich	20
13.08.2021 14:54:26	2	5	7	2	Ja	0	Nein	Nein	181	Männlich	24
12.08.2021 14:32:30	1	5	20	2	Ja	0	Nein	Nein	185	Männlich	23

Durchlauffragebogen

Wie ist deine VP-Nummer? *

Welche Bedingung hast du soeben getätigt? *

Bitte schätzen: Wie weit hast du dich im letzen Durchlauf innerhalb der virtuellen Umgebung fortbewegt? Dabei zählt nur der Bereich innerhalb der Versuchsbedingung(nicht das Menü). *

Präsenzfragen: P1: Please rate your sense of being in the virtual environment, on a scale of 1 to 7, where 7 represents your normal experience of being in a place. *

P2: To what extent were there times during the experience when the virtual environment was the reality for you? *

P3: When you think back to the experience, do you think of the virtual environment more as images that you saw or more as somewhere that you visited? *

P4: During the time of the experience, which was the strongest on the whole, your sense of being in the virtual environment or of being elsewhere? *

P5: Consider your memory of being in the virtual environment. How similar in terms of the structure of the memory is this to the structure of the memory of other places you have been today? By 'structure of the memory' consider things like the extent to which you have a visual memory of the virtual environment, whether that memory is in colour, the extent to which the memory seems vivid or realistic, its size, location in your imagination, the extent to which it is panoramic in your imagination, and other such structural elements. *

P6: During the time of your experience, did you often think to yourself that you were actually in the virtual environment? *

P7: Anything you would like to add?

Antworten Durchlauffragebogen:

Zeitstempel	VP-Nummer	Bedingung	Schätzung	P1	P2	P3	P4	P5	P6	P7
12.08.2021 14:11:09	1	A1	25	6	6	7	5	3	6	
12.08.2021 14:19:52	1	B2	50	4	2	2	5	2	5	
12.08.2021 14:31:56	1	C3	40	3	1	2	5	1	2	
13.08.2021 14:32:37	2	C2	23	6	5	5	5	3	6	Gab leichte Bugs
13.08.2021 14:46:05	2	A3	32	7	7	5	2	6	7	Drehen des Bildschirms wä gabe
13.08.2021 14:53:23	2	B1	12	3	3	2	6	2	3	zoom effect while walking
13.08.2021 15:34:37	3	B3	10m	5	2	2	6	1	6	
13.08.2021 15:40:17	3	C1	5m	5	2	1	7	2	2	
13.08.2021 15:47:42	3	A2	10m	7	3	5	4	4	6	
16.08.2021 19:19:04	4	A1	11	6	7	6	3	3	3	Having a ceiling makes a rence on the immersion
17.08.2021 02:11:29	4	B2	29	5	3	2	5	1	3	
17.08.2021 02:25:40	4	C3	44	5	5	5	5	3	5	
18.08.2021 16:28:58	5	C2	16m	5	6	6	6	4	5	
18.08.2021 16:42:59	5	A3	18 m	6	6	6	7	4	6	
18.08.2021 17:05:45	5	b1	10 m	7	7	6	6	6	6	
18.08.2021 17:36:54	6	b3	10	7	5	6	7	2	7	richtig neue erfahrung
18.08.2021 18:00:58	6	c1	5	5	6	5	5	4	6	war schwierig mit der orient
18.08.2021 18:15:28	6	a2	5	6	7	7	5	6	7	mit der orientierung war vie
18.08.2021 20:03:09	7	A1	13m	3	2	4	4	2	1	
18.08.2021 20:10:39	7	b2	32m	2	1	2	2	1	1	
18.08.2021 20:19:02	7	c3	40m	2	1	1	2	1	1	
19.08.2021 16:44:51	8	c2	14	5	3	5	6	2	2	
19.08.2021 16:56:51	8	a3	35	2	2	5	6	4	2	
19.08.2021 17:03:11	8	b1	20	1	1	1	7	1	1	
19.08.2021 22:49:21	9	B3	43	2	1	4	4	1	3	No
19.08.2021 22:57:26	9	C1	13	3	1	2	4	1	2	Hell no
19.08.2021 23:06:54	9	a2	32	5	3	4	5	2	5	Fucking hell no
20.08.2021 00:00:57	10	a1	3	5	4	7	4	6	5	
20.08.2021 00:12:27	10	b2	1,5	2	2	1	2	2	1	
20.08.2021 00:26:34	10	c3	10	4	2	2	5	4	3	

Distanzmessungen:

dateTime	vp	rooms	locomotion	environment	distanceWalked
12.08.21 14:06	1	3	Redirection	Android	21.78
12.08.21 14:16	1	6	Joystick	Android	30.53
12.08.21 14:26	1	9	Teleport	Android	50.69
13.08.21 14:26	2	6	Teleport	Android	33.16
13.08.21 14:41	2	9	Redirection	Android	50.44
13.08.21 14:48	2	3	Joystick	Android	13.8
13.08.21 15:29	3	9	Joystick	Android	42.89
13.08.21 15:37	3	3	Teleport	Android	14.49
13.08.21 15:45	3	6	Redirection	Android	33.69
16.08.21 19:14	4	3	Redirection	Android	19.06
17.08.21 02:08	4	6	Joystick	Android	35.02
17.08.21 02:23	4	9	Teleport	Android	49.44
18.08.21 16:22	5	6	Teleport	Android	44.56
18.08.21 16:39	5	9	Redirection	Android	53.6
18.08.21 17:01	5	3	Joystick	Android	23.87
18.08.21 17:31	6	9	Joystick	Android	60.63
18.08.21 17:55	6	3	Teleport	Android	27.11
18.08.21 18:11	6	6	Redirection	Android	37.78
18.08.21 19:59	7	3	Redirection	Android	18.21
18.08.21 20:08	7	6	Joystick	Android	31.1
18.08.21 20:17	7	9	Teleport	Android	48.34
19.08.21 16:40	8	6	Teleport	Android	41.29
19.08.21 16:53	8	9	Redirection	Android	52.46
19.08.21 17:01	8	3	Joystick	Android	19.3
19.08.21 22:44	9	9	Joystick	Android	60.44
19.08.21 22:54	9	3	Teleport	Android	15.42
19.08.21 23:05	9	6	Redirection	Android	39.26
19.08.21 23:55	10	3	Redirection	Android	18.69
20.08.21 00:08	10	6	Joystick	Android	36.24
20.08.21 00:21	10	9	Teleport	Android	50.31

Tabelle 7.1:

Ich bin damit einverstanden, dass meine Arbeit in den Bestand der Bibliothek eingestellt wird.

Ort, Datum Unterschrift

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit im Studiengang Mensch-Computer-Innteraktion selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel — insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen — benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.

_____ _____

Ort, Datum Unterschrift