Informatik
Hauptcampus

HOCHSCHULE TRIER

Motion Capture für VR-Anwendungen - Entwicklung eines Prototypen mit Perception Neuron und HTC Vive

Motion capture for VR applications - Development of a prototype with Perception Neuron and HTC Vive

Tobias Heiles

Bachelor-Abschlussarbeit

Betreuer: Prof. Dr. Christof Rezk-Salama

Trier, 28.09.2017

Vorwort

An dieser Stelle möchte ich der Hochschule Trier und meinem Betreuer Prof. Dr. Christof Rezk-Salama dafür danken, dass sie mir die im Folgenden verwendete technische Ausstattung zur Verfügung gestellt haben und sich mir dadurch die Möglichkeit bot, Erfahrungen in den Bereichen VR-Entwicklung und Motion Capture zu sammeln.

Zusätzlich möchte ich darauf hinweisen, dass aus Gründen der besseren Lesbarkeit auf die gleichzeitige Verwendung männlicher und weiblicher Sprachformen verzichtet wird. Sämtliche Personenbezeichnungen gelten gleichwohl für beiderlei Geschlecht.

Kurzfassung

In der vorliegenden Arbeit wird mithilfe von Unity ein Prototyp einer VR-Anwendung entwickelt, der das VR-Headset HTC Vive mit dem Motion Capture System Perception Neuron kombiniert. Ziel dabei ist es, mithilfe der erweiterten Interaktionsmöglichkeiten eine im Vergleich zu bestehenden VR-Anwendungen besonders immersive Steuerung zu entwickeln und deren Anwendung anhand von Beispielmechaniken aufzuzeigen. Zunächst werden die Hardware und technischen Voraussetzungen sowie die verwendeten Programme und Unity-Plugins beschrieben. Die für eine Synchronisierung von Vive und Perception Neuron möglichen Optionen werden diskutiert und die Variante, Perception Neuron an die Position der Vive anzupassen, im weiteren Verlauf der Arbeit verfolgt. Aufgrund der dabei auftretenden Probleme mit der Körpererfassung durch Perception Neuron wird eine Steuerung entworfen und anschließend implementiert, die auf die Controller der Vive vollständig verzichtet. Es werden die einzelnen Bestandteile der Steuerung und die dazu verfassten Skripte sowie deren Verwendung im Prototypen erläutert. Außerdem werden ergänzende Spielmechaniken wie das Sammeln von Items und der Aufbau der Unity Scene beschrieben. Die Arbeit schließt mit einer Bewertung des umgesetzten Projektes sowie einem Ausblick auf darauf aufbauende zukünftige Arbeiten.

The following paper is about the development of an Unity VR prototype that combines the VR headset HTC Vive with the motion capture system Perception Neuron. The purpose is to develop especially immersive controls based on the extended possibilities to interact and demonstrate their application with example mechanics. At first the hardware and technical requirements combined with the used programs and Unity plugins are described. The possible ways to synchronize Vive and Perception Neuron are discussed and the option to align Perception Neuron with the Vive is used in the rest of the paper. Due to occuring problems with the motion tracking of Perception Neuron the controls are implemented in such a way that the Vive controllers are not necessary. The controls are explained in detail along with the implemented scripts and their usage in the prototype.

Furthermore supplementary game mechanics like collecting items and the structure of the unity scene are described. The paper concludes with an assessment of the accomplished work and an outlook on future papers.

Inhaltsverzeichnis

1 Einleitung und Problemstellung 1

2 Hardware 2
2.1 VR-Headset 2
2.2 Motion Capture System 3

3 Software & Plugins 6
3.1 Programme 6
3.2 SteamVR 6
3.3 Perception Neuron 7

4 Theoretische Ansätze 10
4.1 Synchronisierung 10
4.2 Unzureichende Dokumentation 11

5 Umsetzung 12
5.1 Synchronisierung 12
5.2 Probleme mit dem Motion Capture 14
5.3 Steuerung 17
5.3.1 Grundlagen 17
5.3.2 Buttons 17
5.3.3 Menü & Gestenerkennung 19
5.4 Teleportieren 21

6 Ergänzende Mechaniken 24
6.1 Items 24
6.2 Ausrüstung & Events 26

7 Zusammenfassung und Ausblick 28

Literaturverzeichnis 30

Erklärung der Kandidatin / des Kandidaten 31

Abbildungsverzeichnis

2.1 Posen zur Kalibrierung 5

3.1 Animator Instance als Komponente des Spieler-Objektes 8

5.1 Rotationsanpassung in Axis Neuron 14
5.2 Ungenaues Fingertracking in Axis Neuron 15
5.3 Ungenauigkeiten beim Halten des Controllers 16
5.4 HoverButton: ungenutzt / getriggert 18
5.5 HoverInfo 19
5.6 Auswirkung der Teleport Differenz 22

6.1 Items 24
6.2 Inventarmenü 25
6.3 Ausrüstung: Schild & Schwert 27

1

Einleitung und Problemstellung

Ein seit einigen Jahren anhaltender und von der Öffentlichkeit mit großer Aufmerksamkeit verfolgter Trend der Technik- und Videospielbranche ist die Entwicklung von Geräten für „Virtual-Reality“ basierte Anwendungen und Spiele. Dazu werden s.g. Head-Mounted-Displays (HMDs) genutzt, die auf dem Kopf getragen werden und mit einem direkt vor den Augen platzierten Bildschirm einen mehr oder weniger großen Sichtbereich des Nutzers abdecken. Durch die von der Kopfrotation abhängige Bildausgabe wird dem Nutzer der Eindruck erweckt, sich in einer virtuellen Realität zu befinden. Dadurch soll eine im Vergleich zu klassischen Monitoren stärkere Immersion des Nutzers in die Anwendungsumgebung ermöglicht werden.

Ein kostengünstiger Ansatz ist die Verwendung eines Smartphones in Verbindung mit einer Kopfhalterung. Dieses Prinzip wird bspw. von Google Cardboard und Samsung Gear VR genutzt. Daneben werden von verschiedenen Herstellern dedizierte HMDs, s.g. VR-Brillen bzw. VR-Headsets, entwickelt und vertrieben. Seit dem Jahr 2016 stehen mit HTC Vive, Oculus Rift und PlayStation VR drei Systeme für Endverbraucher zur Verfügung. Die wesentlich leistungsfähigere Hardware im Vergleich zur Smartphone Lösung ermöglicht hier eine noch realistischere VR-Erfahrung. Zusätzlich ergeben sich durch spezielle Controller und Position Tracking im Raum weitere Interaktionsmöglichkeiten des Nutzers mit der virtuellen Umgebung.

Im Sinne einer vollständigen Immersion in die virtuelle Realität problematisch ist dabei jedoch, dass der Körper des Nutzers, abgesehen vom Tracking der Handbewegungen mithilfe der Controller, nicht in der virtuellen Umgebung abgebildet werden kann. Bisherige VR-Anwendungen lösen dieses Problem, indem sich die virtuelle Repräsentation des Nutzers auf die Hände beschränkt oder anhand der vorhanden Gerätedaten ein künstlicher Körper berechnet wird, was jedoch nur unzureichend genau funktioniert. Um dieser Einschränkung zu begegnen, wird in der folgenden Arbeit der Ansatz verfolgt, die technischen Möglichkeiten einer VR-Brille um die Körpererfassung eines Motion-Capture-Systems zu erweitern, um den vollständigen Körper des Nutzers zur Interaktion in der VR-Umgebung zur Verfügung zu stellen. Dazu soll eine an die zusätzlichen Eingabemöglichkeiten angepasste Steuerung entwickelt und einige darauf aufbauende Mechaniken in einem Unity-Prototypen umgesetzt werden.

2

Hardware

2.1 VR-Headset

Bei der in dieser Arbeit verwendeten VR-Brille handelt es sich um eine HTC Vive, im Folgenden als „Vive“ bezeichnet, die vom Hardware-Hersteller *High Tech Computer Corporation* in Zusammenarbeit mit dem Spielentwickler und Betreiber der Vertriebsplattform Steam *Valve Corporation* entwickelt und vermarktet wird.[Wika] Mit einer Displayauflösung von 1080 mal 1200 Pixel pro Auge, einer Bildwiederholungsrate von 90Hz und einer horizontalen Sichtfeldabdeckung von 110°[Wika] stellt die Vive neben der Oculus Rift das zum aktuellen Zeitpunkt technisch leistungsstärkste HMD auf dem Endkundenmarkt dar. Das Gerät kann die Position und Rotation jeweils in drei räumlichen Achsen bestimmen. Neben im Headset verbauten Beschleunigungssensoren und Gyroskopen wird dazu das „Lighthouse“ genannte Tracking-System genutzt. Dieses System nutzt zwei idealerweise an gegenüberliegenden Enden des abzudeckenden Bereiches aufzustellende Basisstationen, die in kurzen Intervallen Infrarot-Signale versenden.[VrJ] Die eingebauten Laser blitzen dazu kurz auf und schicken anschließend horizontal und vertikal ausgerichtete Strahlen entlang des Raumes. Mithilfe von Photorezeptoren, die am gesamten Headset angebracht sind, kann die Vive diese Signale empfangen. Anhand der zeitlichen Differenz zwischen dem Aufblitzen des Lasers und dem Empfangen der ausgerichteten Strahlen sowie der bekannten Positionierung der Photorezeptoren zueinander kann die räumliche Lage und Orientierung des Headsets bestimmt werden.[Wikb] Im Vergleich zu kamerabasierten Tracking-Verfahren bietet diese Vorgehensweise einige Vorteile. Da die Basisstationen lediglich ihre vordefinierten Signale aussenden und die eigentlichen Berechnungen vom Headset bzw. dem verwendeten Rechner durchgeführt werden, ist eine flexible Positionierung der Stationen im Raum möglich. Durch die Lagebestimmung auf Grundlage von Zeitdifferenzen entfallen berechnungsintensive Bilderkennungsverfahren. Dennoch ist eine exakte Erfassung im Sub-Milimeter-Bereich bei einer Frequenz von 60Hz möglich. Zudem ist die Erkennung auch in größeren Räumen zuverlässig. Kameras hingegen sind durch ihre Auflösung eingeschränkt, die mit zunehmender räumlicher Tiefe stark abnimmt. Mit einem bis zu 4,6 mal 4,6 Meter großen Bereich[Wika] ermöglicht das Lighthouse-System dem Anwender die im Vergleich zu anderen VR-Lösungen größte Bewegungsfreiheit.[Wikb] Für das in dieser Ar-

beit angestrebte immersive Erlebnis bietet es daher eine ideale Voraussetzung. Zur Interaktion kann der Nutzer zwei spezielle Vive-Controller verwenden, die analog zum Headset über das Lighthouse-System getrackt werden und jeweils neben einem Touchpad über mehrere Buttons sowie Vibrationsmotoren verfügen.

Die Verwendung der Vive setzt die Installation von SteamVR über die Vertriebsplattform Steam voraus. Dabei handelt es sich um ein von Valve entwickeltes Programm bzw. eine Plattform für VR-Anwendungen, die neben der Vive noch andere HMDs wie bspw. die Occulus Rift unterstützt. Über SteamVR kann der Nutzer das Tracking der Vive einrichten und weitere Einstellungen vornehmen.

2.2 Motion Capture System

Wie eingangs beschrieben, sollen in dieser Arbeit die vorhandenen Interaktionsmöglichkeiten eines VR-Headsets, in diesem Falle eine HTC Vive, um die Aspekte des Motion Capturing erweitert werden. Unter Motion Capture versteht man im Allgemeinen das Erfassen von Bewegungen und anschließende Umwandeln in ein für Computer lesbares Format, um diese zu analysieren oder weiterzuverarbeiten. Eine übliche Anwendungsmöglichkeit ist die Nutzung der Bewegungsdaten zur Animation von humanoiden 3D-Modellen, die in Filmen oder Computerspielen zum Einsatz kommen. Dazu wird in der Regel ein optisches Trackingverfahren genutzt, wobei auf dem Körper der zu erfassenden Person angebrachte Marker von einer Vielzahl an Kameras erfasst werden. Durch die Bewegungen der Marker in den einzelnen Kamerabilder kann deren Lage im Raum ermittelt werden.[Wikc]

Das in dieser Arbeit verwendete Motion Capture System ist das seit einigen Jahren von dem Unternehmen Noitom Ltd. vertriebene Perception Neuron, dessen Entwicklung u.a. mithilfe einer Kickstarter Kampagne im Jahr 2014 finanziert worden ist.[Koa] Die Besonderheit dieses Systems ist, dass es auf s.g. inertialen Messeinheiten (*Inertial Measurement Units*) statt auf optischem Kameratracking basiert. Diese *Neuron* genannten und im Folgenden als „Neuronen“ bzw. „Sensoren“ bezeichneten Messeinheiten bestehen aus einem 3-Achsen-Beschleunigungssensor, einem 3-Achsen-Gyroskop sowie einem 3-Achsen-Magnetometer. Mithilfe von Klettverschlussgurten werden an bestimmten Stellen wie bspw. der Schultern, Ober- und Unterarme, des Kopfes usw. Fassungen angebracht, in die jeweils ein beliebiger Neuron eingesetzt und darüber mit Strom versorgt wird. Diese Fassungen werden miteinander über bereits integrierte Stecker verkabelt und an einen mitgelieferten Hub angeschlossen, den der Nutzer ebenfalls an seinem Körper trägt. Die Daten sämtlicher Neuronen werden von diesem Hub synchronisiert und an einen Computer geschickt. Dazu wird der Hub entweder mithilfe eines USB Kabels mit dem Rechner verbunden oder sendet die Daten über eine WLAN-Verbindung, wobei die Stromversorgung dann über eine Powerbank erfolgen muss. Alternativ können die Bewegungsdaten über eine microSD-Karte aufgezeichnet werden.[Neuc]

Eine weitere Besonderheit von Perception Neuron stellt die Flexibilität hinsichtlich der Anzahl verwendeter Sensoren dar. Abhängig davon, welche Teile des Körpers der Nutzer tracken möchte, können unterschiedliche Konfigurationen mit

jeweils eigener Anzahl Gurten und Neuronen verwendet werden. So lassen sich bspw. ein einzelner Arm oder der gesamte Oberkörper erfassen. Auch kann entschiedenen werden, ob die einzelnen Finger einer Hand oder die Hand als Ganzes von Relevanz sind.[Neuc] Um das Vorhaben zu unterstützen, eine maximal mögliche Immersion zu erzeugen, wird in dieser Arbeit eine umfassende Konfiguration mit 31 von maximal 32 unterstützten Sensoren genutzt. Dies ermöglicht es, den gesamten Körper der Testpersonen inklusive der einzelnen Finger zu erfassen. Auf die Verwendung des 32. Neurons, der dazu geeignet wäre, ein Werkzeug oder eine Waffe zu tracken, wird hierbei verzichtet, da lediglich die Vive Controller genutzt werden, deren Erfassung bereits durch das Lighthouse-System sichergestellt ist.

Bei einer Verwendung von bis zu 18 Neuronen liegt die Erfassungsrate bei 120 Hz, bei darüber hinausgehender Anzahl bei 60Hz. Die Messgenauigkeit wird mit 0,02 Grad abgegeben. Da in den Neuronen hochempfindliche Magnetsensoren verbaut sind, sollten die Messungen in einer Umgebung mit möglichst schwachen magnetischen Feldern stattfinden. Zu sämtlichen elektronischen Geräten, u.a. Computern und Lautsprechern, muss ein Abstand von bis zu einem Meter eingehalten werden, um Beeinträchtigungen in der Genauigkeit zu vermeiden. Darüber hinaus kann eine längere Exposition der Neuronen in einem magnetischen Feld zu einer dauerhaften Ungenauigkeit der Sensoren führen, die dann aufwendig neu kalibriert werden müssen.[Neuc]

Auf dem Computer, der die Daten empfangen soll, muss die ebenfalls von Noitom entwickelte Software *Axis Neuron* gestartet sein. Das Programm optimiert die vom Hub gesendeten Daten und führt Driftkorrekturen durch, um anschließend mithilfe weiterer Algorithmen ein aus 59 Knochen bestehendes menschliches Skelett zu rekonstruieren. Die so erzeugten Skelettdaten und -bewegungen können aufgezeichnet und abgespeichert werden, um sie anschließend als .fxb-Datei für die weitere Verwendung zu exportieren. Mögliche Anwendungsfälle sind das Importieren der Animationen in eine 3D-Modellierungssoftware wie Autodesk Maya oder das unmittelbare Nutzen der Daten zur Animation von Charakteren in einer Game Engine. Alternativ dazu können die Daten im BVH-Format per Broadcast an das lokale Netzwerk gestreamt werden. Dieses ursprünglich von dem Unternehmen Biovision für Motion Capture Anwendungen entwickelte Format enthält hierarchisch geordnete Informationen über die Rotationen und Translationen der einzelnen Knochen.[Wis] Durch das Versenden der Daten per Netzwerk muss es sich bei dem Rechner, der die Daten vom Hub erhält und dem, der zu deren weiteren Verwendung genutzt werden soll, nicht um denselben handeln.[Neua]

Bevor die Bewegungsdaten erfasst werden können, muss der Nutzer nach dem Anlegen der Sensoren mithilfe von Axis Neuron eine Kalibrierung durchführen. Dazu müssen in Abhängigkeit von der genutzten Konfiguration bis zu vier Posen für einen Zeitraum von wenigen Sekunden eingenommen werden, mithilfe derer Axis Neuron die Positionen der einzelnen Neuronen zueinander bestimmen kann. Siehe dazu auch Abb. 2.1. Die „Steady“-Pose nimmt dabei eine Sonderrolle ein, da sie nur einmalig nach dem Einsetzen der Neuronen in die Fassungen durchgeführt werden muss, um den durch deren Verbindung entstehenden Bias zu korrigieren. Im Ge-

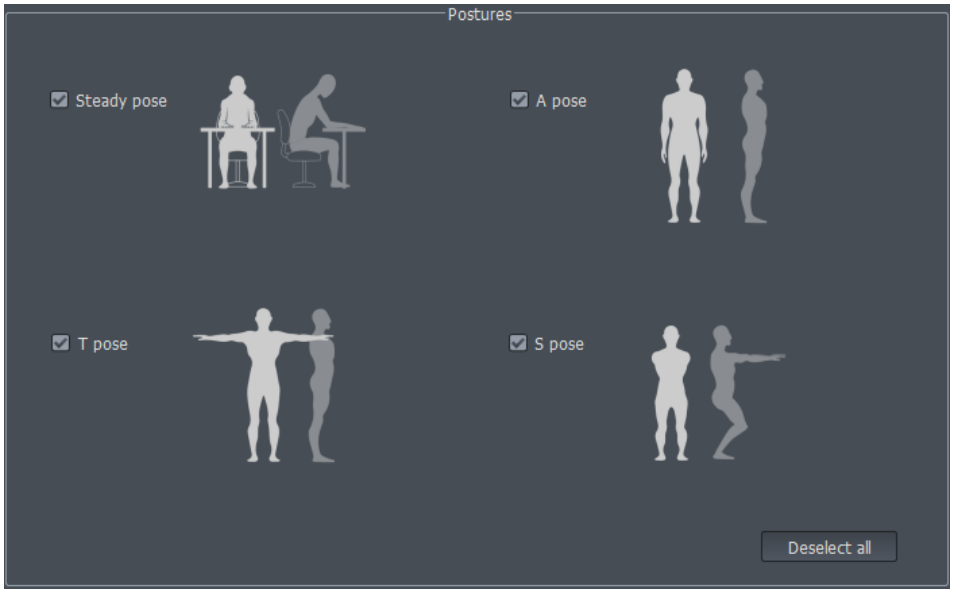

Abb. 2.1. Posen zur Kalibrierung

gensatz zu den übrigen Posen ist es dabei nicht zwingend erforderlich, die Sensoren bereits am Körper zu tragen. Für eine hohe Präzision sollte darauf geachtet werden, die verwendeten Neuronen möglichst korrekt an den vorgesehenen Stellen zu fixieren. Dadurch, dass bereits geringe Veränderungen zu Ungenauigkeiten führen können, sollte man zudem ein Verschieben der Gurte vermeiden. Da die Bewegungen der Sensoren auch von der Größe einer Person bzw. den Körperproportionen abhängig sind, können über den Body Size Manager in Axis Neuron die genauen Abstände der Neuronen zueinander eingestellt werden. Dies geschieht entweder über die Auswahl vordefinierter Konfigurationen für übliche Körpergrößen oder die manuelle Eingabe einzelner Abstände.

Die im Vergleich zu herkömmlicher Motion Capture Technik sehr niedrigen Anschaffungskosten sind eine weitere Besonderheit von Perception Neuron. Der reguläre Preis für die Vollausstattung mit 32 Neuronen liegt bei 1499 US-Dollar.[Neub] Der Hersteller OptiTrack, der Motion Capture Lösungen auf Basis von Kameratracking anbietet, verlangt für ein Setup der minimalen Ausstattung mit 6 Kameras des günstigsten Kameratyps „Flex 3“ bereits 7858 US-Dollar.[Opt] Mit Perception Neuron wird somit eine - wenn auch nur im direkten Vergleich - kostengünstige Alternative geboten. Davon dürften neben dem Endverbraucher insbesondere Entwicklerstudios mit geringen Budgets profitieren, die bisher aufgrund der immensionen Kosten auf eigene Motion Capture Aufnahmen verzichten mussten.

3

Software & Plugins

3.1 Programme

Zur Erstellung des im Folgenden dokumentierten Prototypen wurde die Game Engine Unity in Version 2017.1 Personal verwendet. Die Skripte wurden in der Programmiersprache C-Sharp mithilfe von Visual Studio 2017 Community Edition geschrieben. Die für die eingangs beschriebe Hardware benötigten Programme SteamVR und Axis Neuron wurden in der jeweils aktuellsten Ausführung, entsprechend dem Build vom 30. August 2017 bzw. Version 3.6.32, genutzt.

3.2 SteamVR

Unity bietet eine native Unterstützung für HMDs, die auf Basis von OpenVR betrieben werden. Dabei handelt es sich um eine von Valve für VR-Geräte entwickelte Programmbibliothek bzw. ein API, das u.a. auch als technische Grundlage für SteamVR dient. Allerdings bietet diese native Implementierung nur einen eingeschränkten Funktionsumfang, daher wird stattdessen das ebenfalls von Valve entwickelte SteamVR-Plugin für Unity in Version 1.2.2 genutzt. Neben vordefinierten Interaktionselementen und Komfortfunktionen erlaubt es einen umfangreicheren Zugriff auf die Controller und Funktionen der Vive. Insbesondere wird dadurch eine Unterstützung des raumfüllenen Lighthouse-Trackings in Kombination mit grafisch angezeigten Raumgrenzen ermöglicht.

Das Plugin ist darauf ausgelegt, dem Entwickler ohne aufwendige Konfiguration ein funktionierendes Setup anzubieten, durch das die VR-Unterstützung in der Anwendung gegeben ist, damit dieser sich auf die Erstellung von Inhalten fokussieren kann. Entsprechend sind auch in diesem Projekt nur wenige SteamVR-Skripte bzw. -Objekte direkt verwendet worden. Das in der Unity Scene platzierte **CameraRig**-Prefab enthält die für das Tracking des Headsets und der Controller benötigten Dinge. Dabei wird das Skript **SteamVR_TrackedObject** verwendet, das das Lighthouse-Tracking eines beliebigen Objekt, das mit kompatiblen Vive-Sensoren ausgestattet ist, ermöglicht. Das Rendern des VR-Inhaltes wird durch das Skript **SteamVR_Camera** von den Unity Kameras unterstützt, während für mögliche Soundausgaben das Skript **SteamVR_Ears** als Audiolistener fungiert.

Im Editor kann zudem der Spielbereich des getrackten Raumes durch **SteamVR_PlayArea** visualisiert werden, was es ermöglicht, die Anwendung daran angepasst zu gestalten. Zusätzlich dazu wird das **SteamVR**-Prefab verwendet, das einige globale SteamVR-Einstellungen verwaltet, insbesondere bezüglich des getrackten Bereiches. Funktionen, die im Speziellen für diesen Prototypen entwickelt worden sind, greifen über andere Skripte auf diese Objekte zu.

3.3 Perception Neuron

Die von Perception Neuron gesammelten Bewegungsinformation werden in Form der gestreamten BVH-Daten genutzt. Für die Integration in Unity stellt Noitom Ltd. ein Plugin zur Verfügung, das zuletzt in der Version 0.2.7 verfügbar war. Dieses Plugin enthält neben einem 3D-Mesh des auch in Axis Neuron genutzten Roboter-Modells mehrere Skripte, die Funktionen zum Empfangen der Daten und Animieren von Avataren in Unity bereitstellen. Die wichtigsten davon werden an dieser Stelle erläutert:

- **NeuronConnection** verwaltet die Verbindung zu Axis Neuron, wobei gleichzeitig mehrere Instanzen des Programms genutzt werden können.
- Mithilfe von **NeuronDataReaderManaged** werden dabei die als binäre Floatwerte vorliegenden BVH-Daten aus dem Netzwerk gelesen.
- In Axis Neuron können mehrere Akteure gleichzeitig erfasst werden. Für jeden dieser Akteure wird ein **NeuronActor** erstellt, der die aktuellsten Bewegungsdaten und zusätzliche Motion-Capture-Informationen speichert.
- Jeder Instanz von Axis Neuron wiederum wird eine **NeuronSource** zugeordnet. Diese verwaltet die NeuronActor-Objekte und entscheidet, ob diese aktiv oder inaktiv sind. Letzteres trifft bspw. bei Verbindungsproblemen oder einer geänderten Anzahl getrackter Personen auf. Die Animation eines Unity Avatars mit den Daten des NeuronActor-Objekts setzt eine AnimatorComponent voraus. Das verwendete 3D-Mesh muss als humanoides Skelett in T-Pose geriggt sein und das Bonemapping einem vorgegebenen Template entsprechen.
- Sind diese Voraussetzungen erfüllt, werden mithilfe einer **NeuronAnimatorInstance** die Bewegungsdaten auf die TransformComponents der Skelettknochen angewendet.
- NeuronAnimatorInstance ist dabei eine abgeleitete Variante von **NeuronInstance**, das als allgemeine Basisklasse für die individuelle Nutzung der Bewegungsdaten fungiert und selbst wiederum von *Unity.MonoBehaviour* erbt.

Abb. 3.1 zeigt einen Ausschnitt aus dem Unity Inspector, in dem verschiedene Einstellungen an der NeuronAnimatorInstance vorgenommen werden können. Die wichtigsten betreffen das Netzwerk, über das die BVH-Dateien empfangen werden sollen. *Adress*, *Port* und *SocketType* müssen mit den Einstellungen in Axis Neuron übereinstimmen. Zu beachten ist dabei auch, dass dort der BVH-Broadcast aktiviert sein muss. Über die *ActorID* lassen sich mehrere getrackte Personen innerhalb einer Axis Neuron Instanz identifizieren. Standardmäßig werden die Skelettknochen über unmittelbares Setzen der Transformationen manipuliert. Dies

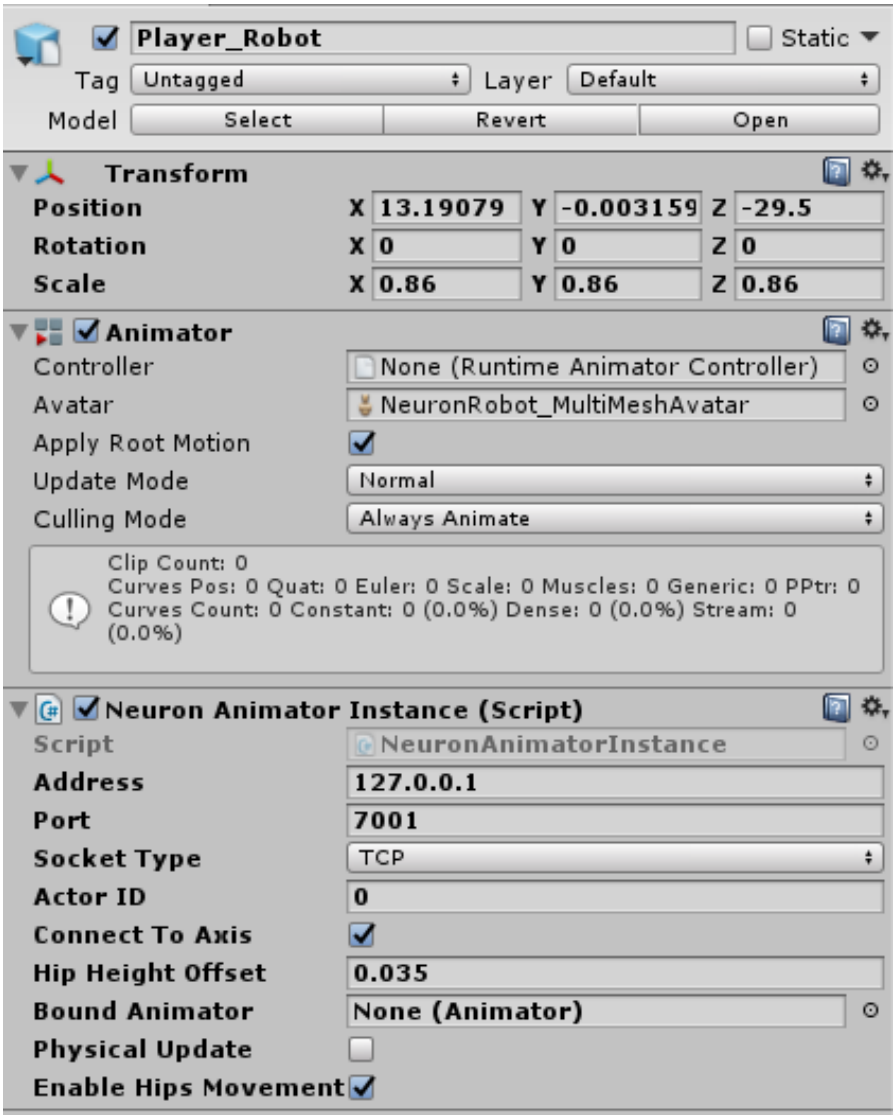

Abb. 3.1. Animator Instance als Komponente des Spieler-Objektes

kann jedoch im Zusammenhang mit Unitys Physik-Engine zu unerwünschten Effekten führen. Bspw. können Kollisionen zwischen dem Avatar und physikalisch simulierten Objekten nicht korrekt aufgelöst werden. Mit dem Aktivieren der Option *Physical Update* wird stattdessen für jeden Knochen eine für die Lageänderung benötigte Geschwindigkeit berechnet, die anschließend in der für physikalisch stabile Manipulationen geeignete *FixedUpdate()*-Methode auf die Rigidbodies der Knochen angewendet wird. Die Rigidbodies können bei einem korrekt aufgesetzten Skelett über die *Skeleton Tools* genannte Editorerweiterung des Plugins automatisch für alle Knochen hinzugefügt oder entfernt werden. Die in Abb. 3.1 gezeigte Konfiguration wird in dieser Form im Prototypen verwendet, wobei als Avatar in der AnimatorComponent das dem Perception Neuron Plugin beigefügte 3D-Mesh eines Roboters Verwendung findet. Das Mesh besteht dabei aus hierarchisch angeordneten Einzelmeshes für die unterschiedlichen Knochen bzw. Bestandteile des

Roboters. Dies ermöglicht es in einfacher Weise, bestimmte Teile zu manipulieren. In diesem Fall wurde dies insofern genutzt, dass der Kopf des Roboters, der aufgrund der durch die VR-Kameraführung bedingte Ego-Perspektive nicht relevant ist, deaktiviert worden ist, um Clipping-Probleme mit dem Mesh zu verhindern. Ist erwünscht, diesen bspw. für Schattenberechnungen aktiviert zu lassen, müssen ggf. auftretende Clipping-Probleme über Shadereinstellungen gelöst werden.

4

Theoretische Ansätze

4.1 Synchronisierung

Mit den beschriebenen Mitteln lassen sich sowohl die Vive als auch Perception Neuron in Unity nutzen, jedoch besteht zwischen diesen Systemen ohne Weiteres keinerlei Verbindung. Soll der Nutzer des Prototypen den Eindruck erhalten, mit dem eigenen Körper ein virtuelles Pendant steuern und sich mit diesem durch die Umgebung bewegen zu können, so ist es notwendig, die Vive, welche hierbei als Kamera fungiert, mit dem Körper des animierten Avatars zu synchronisieren. Exakter formuliert müssen Position und Rotation der Vive mit der des Avatars bzw. dessen Kopfes in Übereinstimmung gebracht werden. Dazu gibt es grundsätzlich zwei sich gegenseitig ausschließende Ansätze: Entweder gibt die Vive Position und Rotation vor und der Avatar wird entsprechend daran angepasst, oder umgekehrt wird die Vive an den Kopf des Avatars positioniert.

Im finalen Prototypen wird die erste dieser Optionen genutzt, da das Tracking der Vive mithilfe der Lighthouse-Basisstationen präziser und zuverlässiger funktioniert als das von Perception Neuron. Zwar ist auch damit unter optimalen Bedingungen eine relativ genaue Erfassung der Bewegungen möglich. Bedingt durch die Funktionsweise der Inertialsensoren ist jedoch die Langzeitstabilität der Messung gegenüber der optischen Lösung der Vive geringer. Dies resultiert aus dem permanent vorhanden Drift der Sensoren, der zu Ungenauigkeiten führt. Zwar lassen diese sich durch komplexe Berechnungen minimieren, doch würden sich aufgrund der beschränkten Genauigkeit der dafür genutzten Gleitkommazahlen selbst bei perfekten Algorithmen minimale Abweichungen ergeben, die sich langfristig aufsummierten. Zudem hat sich im Verlauf der Entwicklung gezeigt, dass das manuelle Positionieren der Vive nicht ohne weiteres möglich ist, da SteamVR ein in sich geschlossenes System darstellt und darin nicht vorgesehen ist, Position und Rotation der Vive durch die Anwendung statt über das Lighthouse-Tracking zu bestimmen. So hatte das Abändern der Methode OnNewPoses(TrackedDevicePose_t[] poses) im Skript *SteamVR_TrackedObject*, in der den Angaben des SteamVR Entwicklerforums zufolge Position und Rotation des getrackten Objekts, hierbei dem Vive Headset, gesetzt werden, keine Auswirkungen. Durch das Setzen der Variable *InputTracking.disablePositionalTracking*, die von Unity für das globale Deaktivieren der HMD-Positionserfassung vorgesehen ist, war es zwar möglich, das

Lighthouse-Tracking der Vive zu deaktivieren, allerdings konnte die Vive anschließend nicht an die Position des Perception Neuron Modells verschoben werden.

4.2 Unzureichende Dokumentation

An dieser Stelle sei angemerkt, dass für das SteamVR-Plugin abseits von einer knappen Anleitung, wie das Plugin in ein Unity-Projekt zu integrieren ist sowie einer Readme-Datei keinerlei Dokumentation verfügbar ist. Hinzu kommt, dass die Skripte kaum bis gar nicht kommentiert sind, wodurch das Zusammenspiel der einzelnen SteamVR-Objekte teilweise schwer nachzuvollziehen ist. Aufgrund des geringen Alters des Frameworks und der sehr dynamischen Entwicklung im Bereich der Virtual Reality finden zudem regelmäßig Änderungen an den Skripten statt, sodass ggf. Hilfestellungen in Entwicklerforen nach kurzer Zeit ihre Gültigkeit verlieren.

Aufgrund der ähnlich kurzen Historie fehlt zum aktuellen Zeitpunkt auch für Perception Neuron eine vollständige API Dokumentation, wie sie für Frameworks und Programmbibliotheken üblich ist. Im Vergleich zu SteamVR sind zwar die Skripte weniger komplex, dennoch erschwert auch hier eine lückenhafte Kommentierung des Codes das Verständnis. Hinzu kommt, dass die dem Unity Plugin beigefügte Anleitung vor der zuletzt veröffentlichten Version 0.2.7 fehlerhaft war, da sie nicht vollständig dem aktuellen Code entsprach und Skripte beschrieben wurden, die zum Teil gar nicht mehr oder unter geänderter Bezeichnung mitgeliefert worden sind. Auch war das Benutzerhandbuch zu Axis Neuron für mehrere Monate nicht unter der angegebenen Verlinkung verfügbar. Ein offizielles Forum ist zwar auf der Website von Noitom vorhanden, allerdings ist eine Betreuung durch die Entwickler nicht in allen Fällen gegeben, sodass viele dort formulierte Problemstellungen ungelöst sind und auch speziell für diese Arbeit dort eingereichte Fragen nicht beantwortet wurden.

5

Umsetzung

5.1 Synchronisierung

Im folgenden Kapitel werden die Skripte beschrieben, die im Rahmen dieser Arbeit entstanden sind. Durch den Präfix *NeuronVR_* sind diese leicht vom übrigen Code zu unterscheiden. Die für diese Arbeit zentrale Synchronisierung zwischen Vive und Perception Neuron findet in der Klasse *NeuronVR_ViveSync* statt. Wie im Abschnitt 4.1 erläutert, werden dabei die Position und Rotation der Vive als Vorgabe betrachtet, an die der durch Perception Neuron gesteuerte Avatar, für den das Modell eines Roboters genutzt worden ist, angepasst wird. Diese Anpassung muss für jeden Update-Zyklus von Unity stattfinden, um eine dauerhaft aufrechterhaltene Synchronisierung zu garantieren. Dabei sei Folgendes gegeben: Das Tracking von Vive und Perception Neuron wird jeweils von dem zugehörigen Plugin bzw. Skript erledigt, das wiederum im Verlauf einer Iteration der Unity Kernschleife über die *Update()*-Methode aktualisiert wird. Somit befinden sich die Kamera und der Avatar nach den Berechnungen der Update()-Methode in dem für ihr System jeweils korrekten und aktuellsten Zustand. Zu diesem Zeitpunkt sollte die Synchronisierung stattfinden. Um dies zu garantieren, wird in der Klasse *NeuronVR_ViveSync* die Methode *LateUpdate()* der Basisklasse *MonoBehaviour* überschrieben. Unity garantiert, dass diese Methode erst aufgerufen wird, nachdem sämtliche Update()-Methoden aller Skripte ausgeführt worden sind. In Listing 5.1 ist dazu ein Ausschnitt der LateUpdate()-Methode zu sehen. Die eigentliche Synchronisierung findet dabei in den Zeilen 16 und 17 statt. Da die Knochentransformationen des Roboters hierarchisch angeordnet sind, wird dieser über das Setzen der in der Hierarchie höchsten Transformation positioniert. Sowohl im in Axis Neuron simulierten Bewegungsmodell als auch in dem hierbei verwendeten 3D-Mesh ist das die Transformation der Hüfte. In Zeile 17 wird dazu die Position der *neuronHips*-Transform gesetzt. Aufgrund der Tatsache, dass die Vive sich nicht an der Hüfte, sondern am Kopf des Nutzers befindet, muss allerdings noch eine Verschiebung der Hüfte stattfinden. Diese Verschiebung entspricht dem Abstand zwischen Kopf und Hüfte des Spielermodells. Die Berechnung dazu findet in Zeile 16 statt. Zu beachten ist hierbei, dass *neuronHead* nicht exakt der Transform des Roboterkopfes entspricht, sondern der eines in Blickrichtung leicht verschobenen

leeren Gameobjects, das der Position der Vive relativ zum Kopf besser entspricht.

```
1  // Script: NeuronVR_ViveSync.cs (gekürzt)
2
3  private void LateUpdate()
4  {
5     // Vive hmd position
6     public Transform viveCameraEye;
7     // Head/hmd holder position on the player's robot
8     public Transform neuronHead;
9     // Hip position of the player's robot
10    public Transform neuronHips;
11
12    // Position and rotation tracking is done by the vive
13    if (mode == SyncMode.ViveParent)
14    {
15       // Since the hips are the root of the player's robot you have
             to position the hips
16       Vector3 headHipOffset = neuronHead.position -
             neuronHips.position;
17       neuronHips.position = viveCameraEye.position - headHipOffset;
18    }
19 }
```

Listing 5.1. Synchronisierung Vive / Perception Neuron

Die Rotation des Avatars muss hierbei nicht gesetzt werden. Während der Entwicklung hat sich herausgestellt, dass die Rotation des Körpers von Axis Neuron ausreichend genau erfasst wird. Sollte sich der Spieler relativ zur vertikalen Achse drehen und damit die Vive um diese Achse rotieren, wird der Avatar in Unity durch Axis Neuron in die korrekte Ausrichtung versetzt. Es ist jedoch einmalig zu Beginn der Simulation notwendig, die Ausrichtung der beiden Systeme zu synchronisieren, damit die Rotationen relativ zum gleichen Ursprung stattfinden. Eine einfache und in dieser Arbeit verwendete Lösung besteht darin, die Rotation des Spielers mithilfe von Axis Neuron anzupassen. Über den Reiter *Model Alignment* kann die Ausrichtung des Modells in drei Achsen angepasst werden. Die dadurch geänderte Rotation wirkt sich sowohl auf die Darstellung in Axis Neuron als auch auf die erstellten Bewegungsdaten und damit die Darstellung in Unity aus. Abb. 5.1 zeigt dazu einen entsprechenden Ausschnitt aus Axis Neuron. Die Rotation analog zur globalen Y-Achse in Unity wird über den *Yaw*-Wert des Modells angepasst. Um während der laufenden Simulation auf Axis Neuron zuzugreifen, kann über das SteamVR Overlay, das durch die Verwendung des zugehörigen Plugins in der Unity-Anwendung mittels Vive Controller verfügbar ist, der Desktop des Rechners bedient werden. Alternativ und für den Fall, dass Axis Neuron auf einem anderen Rechner läuft, muss dies von einer zusätzlichen Person erledigt werden. Die Anpassung mittels SteamVR-Overlay hat sich während der Entwicklung trotz zunächst ungewöhnlich erscheinender Vorgehensweise als unkomplizierte Lösung erwiesen.

Eine kleine Einschränkung der Genauigkeit besteht allerdings für den Fall, dass der Spieler, während er stehen bleibt, mit der Vive auf den Boden schaut und dabei den Kopf bewegt. Dadurch, dass die Beugung des Halses nicht exakt von Percepti-

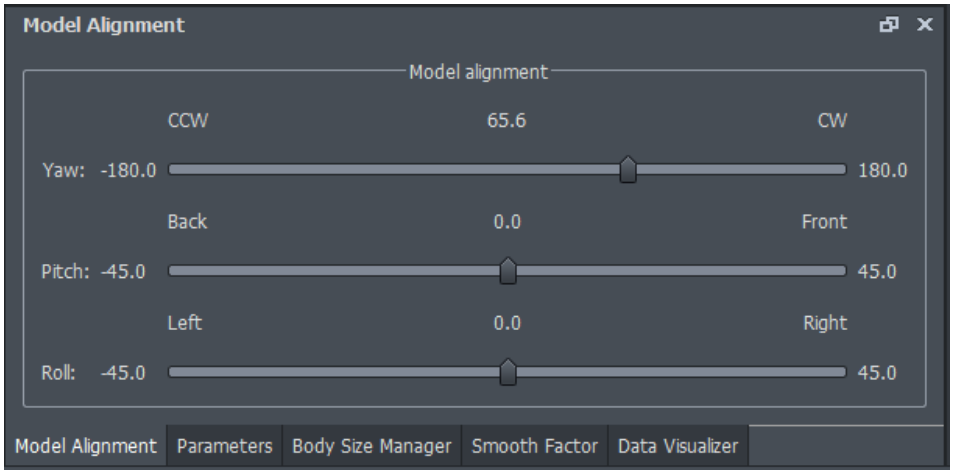

Abb. 5.1. Rotationsanpassung in Axis Neuron

on Neuron abgebildet beziehungsweise vom Programm nicht berücksichtigt wird, ist die durch den Abstand zwischen Kopf und Hüfte definierte Verschiebung der Hüfttransformation nicht exakt. In der laufenden Simulation ist dies dadurch zu beobachten, dass die Füße bzw. Beine des virtuellen Körpers sich leicht durch Änderung der Kopflage bewegen, auch wenn die Füße des Nutzers auf dem Boden ruhen. Da die Beine jedoch für die Mechaniken des Prototypen keine Relevanz haben und der Effekt nur in der genannten Situation wahrnehmbar ist, wird diese Ungenauigkeit im weiteren Verlauf dieser Arbeit als vernachlässigbar betrachtet.

5.2 Probleme mit dem Motion Capture

Die funktionierende Verknüpfung von Vive und Perception Neuron sollte nun genutzt werden, um Mechaniken eines potentiell darauf aufbauenden Spieles zu entwerfen und Prototypen davon zur Verfügung zu stellen. Es war zunächst geplant, eine Steuerung zu implementieren, die auf einer Kombination der Vive Controller und der Körperbewegungen des Nutzers, insbesondere der Hände und Arme basiert. Das Tracking des Körpers inklusive der Beine impliziert dabei zwar, dass der Spieler sich auch innerhalb eines größeren Spielbereiches bewegen kann, doch würde die endliche Größe dieses Bereiches für die Mehrheit der Entwickler eine Einschränkung darstellen. Daher sollte es, wie in anderen VR-Anwendungen üblich, die Möglichkeit geben, sich mithilfe der Vive Controller durch die Umgebung zu teleportien. Neben dem Teleportieren sollten die Controller als Werkzeug fungieren, um unterschiedlichste Tätigkeiten in der virtuellen Realität zu bewerkstelligen. Die Hände stattdessen sollten für andere Dinge, wie dem Greifen von Gegenständen genutzt werden und darüber hinaus die Immersion des Spielers verstärken, der die Controller nicht mehr bloß als schwebende Gegenstände vor sich sieht, sondern als real erscheinende Werkzeuge, die er in seinen eigenen Händen hält.

Voraussetzung für eine gelungene Immersion ist dabei, dass die Bewegung des Spielers exakt mit der des Avatars in der Simulation übereinstimmt. Im Rahmen

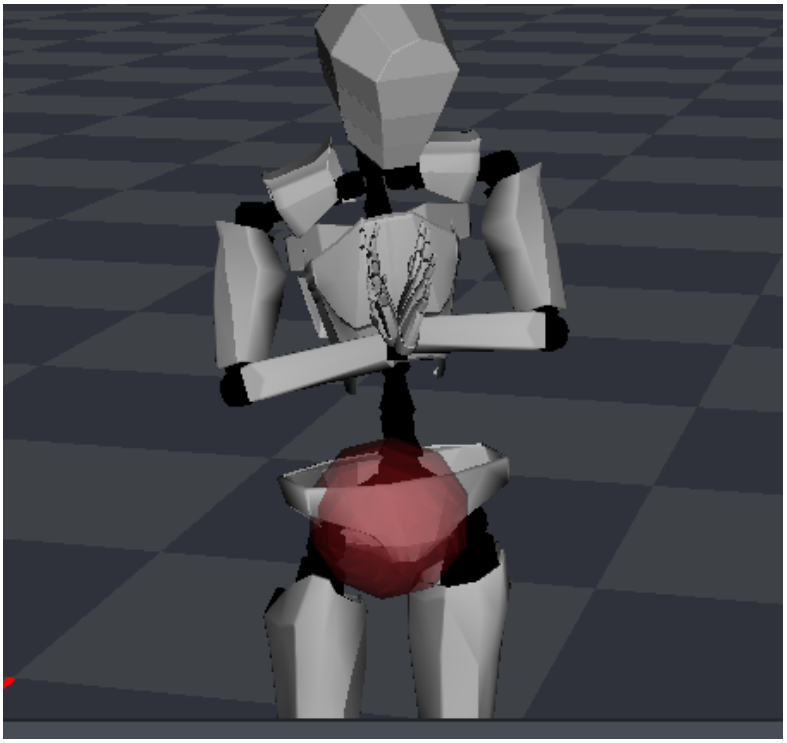

Abb. 5.2. Ungenaues Fingertracking in Axis Neuron

dieser Arbeit wurden jedoch Probleme bezüglich der präzisen Erfassung mit Perception Neuron deutlich. Insbesondere die Hände und Arme konnten damit nicht akkurat abgebildet werden. Zum einen ist bereits in Axis Neuron eine Abweichung zu beobachten. Abb. 5.2 zeigt dazu einen Ausschnitt. In der dargestellten Szene hat die Testperson ihre Hände mit den flachen Handflächen aneinander gelegt. Wie in der Abbildung zu sehen ist, erkennt die Software dies nicht korrekt, sodass die Hände fälschlicherweise ineinander zu greifen scheinen. Zum anderen kann aber auch nicht ausgeschlossen werden, dass die Bewegungsdaten in Unity nicht korrekt genutzt wurden. In Axis Neuron kann über den Body Size Manager exakt festgelegt werden, in welchem Abstand zueinander die Sensoren auf dem Körper der zu erfassenden Person angebracht sind, um dadurch die Präzision zu erhöhen. Das in Unity verwendete 3D-Modell passt sich jedoch nicht dynamisch an individuelle Körperproportionen an. Um mit der Größe der Testperson übereinzustimmen, wurde lediglich das Modell des Roboters in Unity soweit skaliert, bis die Bewegungen des Modells in etwa den realen entsprachen. Potentiell entsteht dadurch eine zusätzliche Diskrepanz zwischen tatsächlicher Bewegung und in Unity resultierender Animation. Solange der Spieler die Hände nicht unmittelbar nebeneinander positioniert, ist das für diesen möglicherweise zwar nicht bemerkbar, sodass die ungefähre Übereinstimmung genügen würde. Im Zusammenspiel mit den Vive Controller, die der Spieler für die Steuerung zwingend in den Händen halten soll, wird die ungenügende Präzision des Trackings aber unmittelbar deutlich.

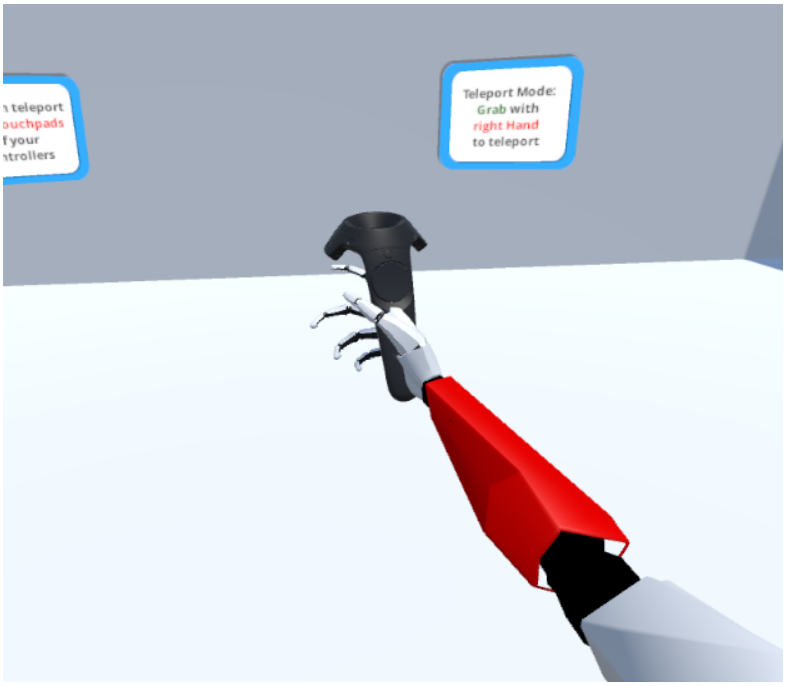

Abb. 5.3. Ungenauigkeiten beim Halten des Controllers

In Abb. 5.3 ist ein Ausschnitt aus dem laufenden Prototypen zu sehen. Der Spieler hält hierbei einen Vive Controller in den Händen. Durch das Lighthouse-Tracking wird dieser präzise erfasst und an der korrekten Position in der Unity Scene angezeigt. Wie in der Abbildung zu sehen ist, greifen die Hände des Spielers in der virtuellen Umgebung jedoch nicht um den Controller bzw. durch diesen hindurch. Zudem ändert sich die relative Distanz zwischen Controller und Händen, wenn der Spieler die Arme zur Seite bewegt. Durch diesen offensichtlichen Fehler wird die Immersion deutlich beeinträchtigt. Da Hände und Controller die primären Interaktionsmittel darstellen, wird der Spieler permanent mit dieser technischen Problematik konfrontiert.

Es ist im Rahmen der Arbeit nicht gelungen, eine Lösung für dieses Problem zu finden. Im offiziellen Forum empfehlen die Entwickler von Perception Neuron, das Problem der unpräzisen Handerfassung dadurch zu lösen, die Bewegungsdaten abzuspeichern und nachträglich in einer 3D-Modellierungssoftware wie Maya oder 3ds Max zu korrigieren. Diese Vorgehensweise stellt aufgrund der in dieser Arbeit an das Motion Capture System gestellten Echtzeitanforderungen zwar keine valide Lösung dar. Allerdings deutet es darauf hin, dass die Probleme mit der Handerfassung weniger aus einer fehlerhaften Nutzung der Daten durch den Verfasser der Arbeit entstanden sind, sondern dass mit der aktuellen Version von Perception Neuron unabhängig davon kein ausreichend präzises Tracking möglich ist.

5.3 Steuerung

5.3.1 Grundlagen

Die Entwicklung des Prototypen sollte trotz der beschriebenen Probleme fortgesetzt werden. Da das Motion Tracking integraler Bestandteil dieser Arbeit ist, kann auf die Verwendung von Perception Neuron nicht verzichtet werden. Es galt daher, eine Möglichkeit zu finden, die aus der unpräzisen Arm- und Handerfassung resultierende Minderung der Immersion zu reduzieren bzw. kompensieren. Wie im Zusammenhang mit Abb. 5.3 dargelegt, hat das Zusammenspiel von Controller und Hand den schwerwiegendsten Immersionsverlust zur Folge. Als Konsequenz wurde eine Steuerung entworfen und umgesetzt, die auf die Vive Controller vollständig verzichtet. Lediglich das Öffnen des SteamVR-Overlays zum Anpassen der Rotation in Axis Neuron erfolgt weiterhin darüber.

Ohne die Eingabemöglichkeiten der Controller verbleiben ausschließlich die Hände, über die nun sämtliche Interaktionen durchgeführt werden müssen. Die Steuerung muss es dem Spieler dabei sowohl erlauben, unmittelbar mit der Umgebung zu interagieren, als auch Menüs und Interfaces allein mit den Bewegungen seiner Hände zu bedienen. Zwischen diesen Möglichkeiten sollte der Spieler zudem ohne Umwege und in kurzer Zeit wechseln können, was eine dynamische Erkennung der Steuerungseingaben erforderlich macht. Zusätzlich bedarf es einer Portierung der Teleport-Funktion von der Controller- auf die Handsteuerung.

Im Rahmen der Arbeit wurde schließlich eine Steuerung entwickelt, die sämtliche der genannten Anforderungen erfüllt. Grundlegender Bestandteil dieser Steuerung ist die Implementierung einer einfachen Gestenerkennung der Hände. Diese dient dazu, unterschiedliche Menüs aufzurufen, über die der Spieler weitere Interaktionsmöglichkeiten aufrufen kann. Zusätzlich werden bestimmte Gesten dazu verwendet, in der Umgebung befindliche Gegenstände zu greifen und einzusammeln. Der Spieler kann zudem Buttons, die im Prototyp bspw. für das Öffnen von Türen genutzt werden, durch das Hineinführen seiner Hände aktivieren. Im Folgenden sollen dazu die einzelnen Steuerungselemente und die dazugehörigen Skripte näher erläutert werden. Zudem wird die im finalen Prototypen daraus entwickelte Menüführung beschrieben.

5.3.2 Buttons

Eine unabhängig von der konkreten Eingabemöglichkeit simple Interaktion stellt das Benutzen eines Buttons dar, um bspw. eine Aktion in der Spielwelt zu starten oder innerhalb eines Menüs eine Bestätigung durchzuführen. Dieses einfache, aber vielseitig verwendbare Steuerungselement wurde im Prototypen in Form des **HoverButtons** mithilfe des gleichnamigen Skriptes umgesetzt. In Abb. 5.4 ist ein solcher Button zu sehen. Es handelt sich dabei im Grunde um ein interaktives Billboard, also eine 2-dimensionale Grafik, die stets zur Kamera hin ausgerichtet ist. Im Zentrum der Grafik befindet sich ein Text, der die durch den Button auszuführende Aktion beschreibt. Aufgrund der durch Perception Neuron bedingten

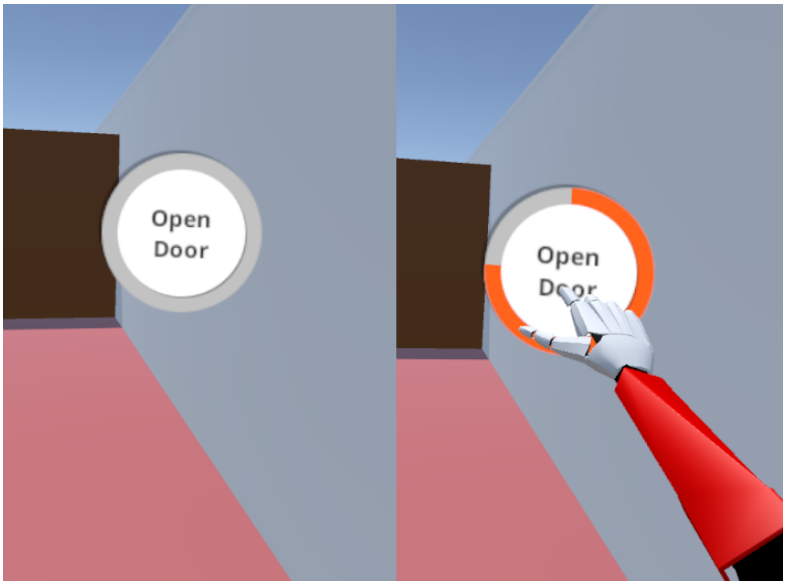

Abb. 5.4. HoverButton: ungenutzt / getriggert

Ungenauigkeit ist das Betätigen des Buttons mittels einzelner Finger nicht zuverlässig genug. Das Billboard ist stattdessen in etwa handgroß und wird durch das Berühren mit einer beliebigen Hand ausgelöst. Um ein versehentliches Betätigen des Buttons durch eine unachtsame Bewegung zu verhindern, muss der Spieler die Hand dabei für einen gewissen Zeitraum hinein halten. Der Button wird dadurch gewissermaßen aufgeladen, was durch einen kreisförmigen Fortschrittsbalken am äußeren Rand der Grafik visualisiert wird. Sobald dieser Balken vollständig gefüllt ist, wird die mit dem Button verknüpfte Aktion ausgeführt.

```
1  // Script: NeuronVR_HoverButton.cs
2
3  private void LateUpdate()
4  {
5      Vector3 myPos = transform.position;
6      Vector3 camPos = camTransform.position;
7
8      // Only rotate around y axis
9      camPos.y = myPos.y;
10
11     Vector3 camToMyPos = myPos - camPos;
12     transform.rotation = Quaternion.LookRotation(camToMyPos);
13 }
```

Listing 5.2. Billboard Effekt

Sowohl die Hände des Spielers als auch der HoverButton sind mit Trigger-Collidern versehen. Über die *OnTriggerEnter()*- und *OnTriggerExit()*-Methoden verwaltet der Button, mit wie vielen Händen er kollidiert. Solange die Anzahl größer Null ist, wird im nächsten *Update()*-Aufruf der Balken gefüllt, andernfalls leert sich dieser wieder. Die mit dem Button verbundene Aktion ist als *Unity Event* implementiert. Über den Inspector lassen sich diesem eine beliebige Anzahl an Me-

thoden zuordnen, die bei Auslösen des Events aufgerufen werden sollen. Dadurch kann der Button vom Level Designer sehr flexibel mit gewünschter Spiellogik verknüpft werden. Für den Billboard-Effekt wird das Sprite des Buttons orthogonal zur Kamera gedreht. Damit dazu stets die aktuellste Position genutzt wird, werden die dazu notwendigen Berechnungen in der *LateUpdate()*-Methode durchgeführt. Die entsprechende Implementierung ist in Listing 5.2 nachzuvollziehen.

Eine Abwandlung der HoverButtons stellen die **HoverInfos** dar. Diese dienen zur Informationsvermittlung und übernehmen die Billboard- und Text-Elemente der Buttons, verzichten allerdings auf die Interaktionsmöglichkeit. Durch eine rechteckige statt kreisförmige Form sowie einer anderen Randfarbe sind sie für den Spieler deutlich von den HoverButtons zu unterscheiden, sodass dieser nicht versehentlich versucht, eine Aktion durch Berühren der Info durchzuführen. In Abb. 5.5 ist die Verwendung einer solchen HoverInfo im Prototypen zu sehen.

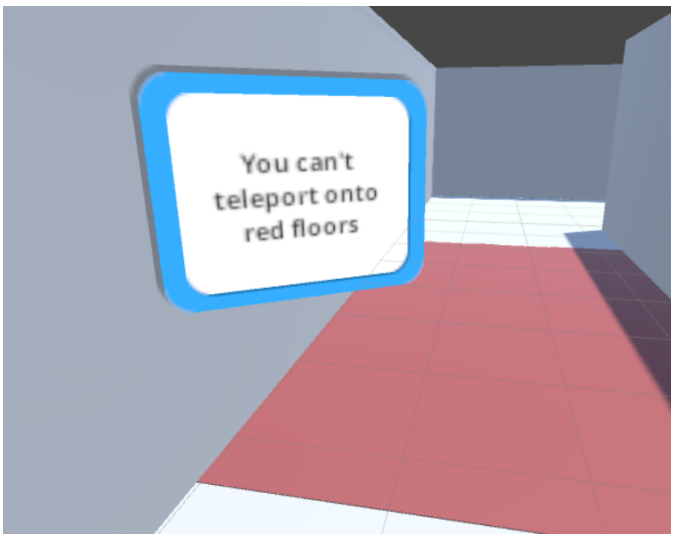

Abb. 5.5. HoverInfo

5.3.3 Menü & Gestenerkennung

Die HoverButtons können universell eingesetzt werden. Im Prototypen kommen sie abseits des Öffnens von Türen als grundlegender Bestandteil der Menüs zum Einsatz. Über das Hauptmenü kann der Spieler bestimmte Aktionen wie bspw. das Teleportieren auswählen und sich dabei durch unterschiedliche Menüzustände navigieren. Ein weiteres Menü dient der Inventarverwaltung im Zusammenhang mit Items, die im späteren Verlauf der Arbeit erläutert werden. Das Öffnen und Schließen der Menüs erfolgt dabei über bestimmte Gesten der Hände. Die Gestenerkennung ist in der Klasse **MenuManager** implementiert, die zudem den aktuellen Menüzustand und das Öffnen und Schließen der Menüs verwaltet.

Zum Öffnen der Menüs muss die Hand des Spielers mit der offenen Handfläche nach oben gehalten werden. Analog zur Aktivierung der Buttons erfolgt das Öffnen

eines Menüs nicht augenblicklich, sondern erst, nachdem der Spieler die Menügeste für eine gewisse Zeit gezeigt hat. Anschließend bleibt das Menü solange offen, wie die Geste weiterhin erkannt wird. Da die visuellen Bestandteile der Menüs vor dem Öffnen nicht sichtbar sind, fehlt eine entsprechende Anzeige, die das korrekte Halten der Menügeste verdeutlicht. Daher wird die Handinnenfläche des Roboters blau gefärbt, solange die richtige Geste erkannt wird. Durch den MenuManager wird sichergestellt, dass der Spieler nur ein Menü gleichzeitig öffnen kann, da jeweils die andere Hand benötigt wird, um mit dem Menü zu interagieren. Das Hauptmenü ist dabei nur mit der linken Hand zu öffnen; das Inventarmenü analog dazu nur mit der rechten. Die Menüs bestehen jeweils aus mehreren Buttons oder Informationstafeln, die nebeneinander angeordnet über der entsprechenden Hand schweben. Dazu werden die bereits beschrieben HoverButtons und -Infos genutzt, die für die Menüs lediglich klein skaliert wurden. Folgende Funktionen sind über das Hauptmenü aufrufbar: *Exit Game* schließt die Anwendung. Alternativ ließe sich hier auch ein Neustart des Spiels verwenden. Da die Vive Controller im Spiel nicht aktiv genutzt werden, wird der Spieler diese i.d.R. auf dem Boden abgelegt haben. Möchte er sie bspw. zum Aufrufen des SteamVR-Overlays wieder aufheben, kann er sie durch den Button *Find Controllers* leichter wiederfinden. Dadurch wird das Controller Menü geöffnet, das mittels einer Infotafel über der linken Hand anzeigt, wie viele Controller von der Anwendung erkannt werden. Diese vibrieren für einen kurzen Augenblick, damit der Spieler sie durch das akustische Signal besser orten kann. Falls beide Controller aktiv sind, geschieht die Vibration zudem zeitversetzt, um dem Spieler ein bessere Unterscheidung zu ermöglichen. Die dritte Möglichkeit ist der Aufruf des *Teleport* Menüs. Analog zum Controller Menü wird in der linken Hand eine Infotafel angezeigt, die dem Spieler Auskunft über die Steuerung gibt. Solange das Menü offen ist, erscheint am rechten Unterarm ein Laserpointer, mit dem der Spieler bestimmen kann, wohin er teleportieren möchte. Aktiviert wird der Teleport durch das Zeigen einer Greifgeste mit der rechten Hand. Um zu verhindern, dass die Greifgeste bzw. der Wechsel zwischen erkannter und nicht erkannter Geste dazu führt, dass der Teleport mehrfach hintereinander ausgeführt wird und den Spieler an eine ungewollte Position bringt, hat dies eine Abklingzeit von einigen Sekunden. Aus Controller und Teleport Menü gelangt der Spieler nicht zurück in das Hauptmenü. Dieses muss nach vorherigem Schließen des aktuellen Menüs wie oben beschrieben durch Halten einer Geste geöffnet werden.

Ob der Spieler die Menügeste zeigt, wird über den Winkel zwischen der Handinnenfläche und der in Unity durch die globale z- und x-Achse aufgespannte Ebene bestimmt. Dazu werden dem MenuManager die Transform-Komponenten der Hände übergeben. Außerdem erhält dieser Referenzen auf die einzelnen Fingerabschnitte, um anhand der Winkel zwischen diesen Abschnitten zu prüfen, ob der Spieler eine Greifgeste zeigt. Das Greifen wird im Zusammenhang mit den Items benötigt, die in Abschnitt 6.1 näher beschrieben werden. Um analog zur Menügeste zu visualisieren, ob die Anwendung ein Greifen erkennt, werden die im normalen Zustand roten Unterarme des Roboters bei gültiger Geste grün eingefärbt.

5.4 Teleportieren

Da zu Beginn der Arbeit geplant war, die Vive Controller zu verwenden, wurde die Teleportfunktion zunächst auf Basis der Controller im Skript **NeuronVR_LaserPointerController.cs** implementiert. Berührt der Spieler das Touchpad eines Vive Controllers, wird in der *Update()*-Methode eine Raycast gestartet, dem als LayerMask der Layer „Teleport“ übergeben wird. Sollte innerhalb der angegeben Maximalreichweite ein Objekt von diesem Raycast getroffen werden, wird ein vom Controller ausgehender Laserstrahl aktiviert, dessen Länge abhängig von der Entfernung der getroffenen Oberfläche ist. Auf dem Layer „Teleport“ befinden sich sämtliche Objekte und Strukturen wie Wände und Böden, die der Laserstrahl nicht durchdringen soll und daher mit ihnen kollidieren muss. Da der Spieler sich nur auf bestimmte Bodenflächen teleportieren darf, wird im Falle eines erfolgreichen Raycasts zusätzlich geprüft, ob der Tag des getroffenen Objektes „CanTeleport“ entspricht. Falls dies zutrifft, wird an dieser validen Position ein ringförmiges Fadenkreuz angezeigt, um dem Spieler zu visualisieren, dass hier ein Teleport möglich ist. Lässt der Spieler das Touchpad des Controllers los, während der Laserstrahl eine gültige Fläche trifft, wird der Spieler teleportiert. Dazu wird die Methode *Teleport()* aufgerufen, deren Implementierung in Listing 5.3 nachzuvollziehen ist.

```
1  // Script: NeuronVR_LaserPointerController.cs
2
3  private void Teleport()
4  {
5      canTeleport = false;
6      reticle.SetActive(false);
7      // Difference is required to preserve the player's position
           relative to the cameraRig
8      Vector3 difference = cameraRigTransform.position -
           headTransform.position;
9      difference.y = 0;
10     cameraRigTransform.position = hitPoint + difference;
11 }
```

Listing 5.3. Teleport

Die Position des Spielers entspricht der des Vive Headsets, welches als Child-Objekt abhängig vom SteamVR CameraRig ist. Um den Spieler zu teleportieren, wird daher das CameraRig an die Stelle positioniert, auf die der Laserpointer gerichtet war. Zusätzlich muss das CameraRig um die zum Spieler bestehende Distanz verschoben werden. Dies ist darin begründet, dass der Pivot Punkt des CameraRigs in dessen Zentrum liegt und ohne die zusätzliche Verschiebung nicht der Spieler, sondern das CameraRig zental auf die Teleportposition gesetzt würde. Die Höhendifferenz ist dabei nicht relevant und wird auf Null gesetzt. In Abb. 5.6 ist die Auswirkung der Differenz dargestellt.

Für die im weiteren Verlauf der Arbeit konzipierte Hand- und Gestensteuerung wurde die Teleportfunktion in angepasster Form im Skript **NeuronVR_LaserPointerController.cs** implementiert. Die Methoden zur Anzeige des Laserstrahls und dem Überprüfen einer validen Teleportposition wurden dabei weitgehend

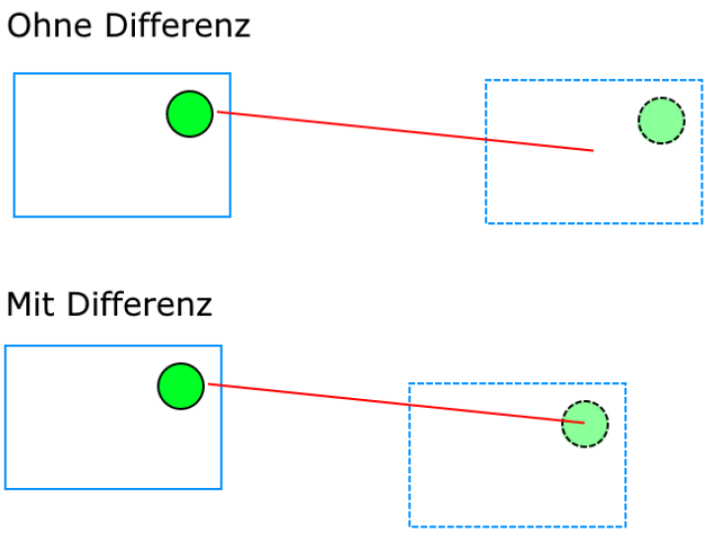

Abb. 5.6. Auswirkung der Teleport Differenz

übernommen. Allerdings muss bei der Handsteuerung beachtet werden, dass der rechte Unterarm, von dem die Position und Ausrichtung des Lasers abhängig ist, durch die Synchronisierung zwischen Vive und Perception Neuron verschoben wird. Da die Vive Controller von der Synchronisierung nicht betroffen sind, musste dies in der ersten Implementierung der Teleport-Mechanik nicht berücksichtigt werden. Die Position des Spieleravatars wird mit jeder Iteration der *Update()*-Methode auf die eigentliche von Perception Neuron bestimmte Position (zurück-)gesetzt und erst im Anschluss daran in der *LateUpdate()*-Methode mit der Vive synchronisiert. Um den Zielpunkt des Teleports mit der korrekten synchronisierten Position zu bestimmen, wird die Berechnung innerhalb des Skripts **NeuronVR_ViveSync.cs** durchgeführt, nachdem die Synchronisierung erfolgt ist. Im Listing 5.4 ist in Zeile 14 der entsprechende Methodenaufruf zu sehen. Durch diesen wird neben der Position des Zielpunktes bestimmt, ob der Spieler eine gültige Fläche zum Teleportieren anvisiert. Damit der Teleport auslöst, muss die rechte Hand eine Greifbewegung durchführen. Da dies erst in der darauffolgenden Iteration innerhalb der *Update()*-Methode des MenuManagers geprüft wird, speichert der Laserpointer die zuletzt anvisierte Position. Wird anschließend die Greifbewegung erkannt, führt der MenuManager den Teleport mit der gespeicherten Position durch Aufruf der *Teleport()*-Methode durch. Deren Implementierung ist in Listing 5.5 nachzuvollziehen.

```
1  // Script: NeuronVR_ViveSync.cs
2
3  private void LateUpdate()
4  {
5     // Position and rotation tracking is done by the vive
6     if (mode == SyncMode.ViveParent)
7     {
8        // Since the hips are the root of the player's robot you have
             to position the hips
9        Vector3 headHipOffset = neuronHead.position -
             neuronHips.position;
10       neuronHips.position = viveCameraEye.position - headHipOffset;
11
12       // Using the laserpointer needs to be checked after position
             syncing since the laser's position is relative to the
             player's position
13       if (laserPointer.gameObject.activeInHierarchy)
14          laserPointer.CheckLaserWithSyncOffset();
15
16       // Items need to be respawn after the position syncing
17       // since the respawn position is relative to the player's
             position
18       foreach (var item in items)
19       {
20          if(item.HasRespawned())
21          {
22             item.Respawn(respawnPosition.position);
23          }
24       }
25    }
26    // (...)
27 }
```

Listing 5.4. Verzögerter Teleport & Item Respawn

```
1  // Script: NeuronVR_LaserPointerHand.cs
2
3  public bool Teleport()
4  {
5     if(canTeleport)
6     {
7        canTeleport = false;
8        // Difference is required to preserve the player's position
             relative to the cameraRig
9        Vector3 difference = cameraRigTransform.position -
             headTransform.position;
10       difference.y = 0;
11       teleportPosition = hitPoint + difference;
12
13       cameraRigTransform.position = teleportPosition;
14       return true;
15    }
16    return false;
17 }
```

Listing 5.5. Teleport mit gespeicherter Position

6

Ergänzende Mechaniken

6.1 Items

Wie im vorigen Kapitel beschrieben, ist die Erkennung einer Greifgeste durch den MenuManager implementiert worden. Der Spieler kann mithilfe dieser Geste Items aufheben, um sie zu bewegen und einzusammeln. In Abb. 6.1 sind die im Prototypen vorhanden Items dargestellt. Um diese aufzuheben, muss der Spieler eine beliebige Hand in bzw. sehr nahe an das Item heran bewegen und anschließend eine Greifbewegung durchführen. Bewegung meint dabei, dass der Spieler nach vorigem Nichtzeigen mit dem Zeigen der Greifgeste beginnen muss. Es genügt demnach nicht, eine bereits greifende Hand in das Item hinein zu bewegen. Durch diese Mechanik soll eine natürliche Greifbewegung mit der Hand auf das virtuelle Greifen der Items übertragen werden. Nach dem Ausführen der Bewegung wird das Item mit der greifenden Hand solange mit bewegt, bis der Spieler aufhört, die Greifgeste zu zeigen.

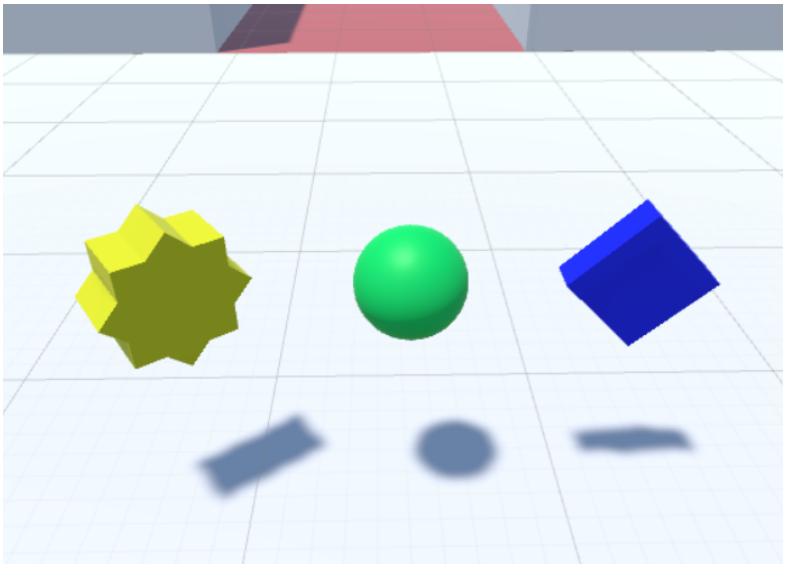

Abb. 6.1. Items

Die Items dienen im Prototypen dazu, eine Tür am Ende des Levels zu öffnen, indem sie in einen jeweils zugeordneten Sockel abgelegt werden. Der Spieler könnte die Items einzeln bis zu diesem Sockel in der Hand tragen. Eine Alternative, die zudem einen weiteren Verwendungszweck für die entwickelte Menüstruktur aufzeigen soll, stellt das Inventar dar, in das der Spieler die Items ablegen und aus dem heraus er diese wieder erscheinen lassen kann. Um ein Item einzusammeln, muss dieses mit einer Hand gegriffen und zum Bauchbereich des Spielers bewegt werden. Während ein Item gegriffen wird, ist dieser durch einen rötlichen Quader visualisiert. Lässt der Spieler das Item los, während es sich in diesem Bereich befindet, wird es in das Inventar aufgenommen und verschwindet, wobei dies in Unity so umgesetzt ist, dass das Item an eine für den Spieler unzugängliche Position verschoben wird. Das Inventar ist durch Halten der Menügeste mit der rechten Hand zu öffnen. Für das Inventarmenü werden **InventoryButtons** genutzt, die weitgehend den HoverButtons entsprechen, jedoch in der Mitte anstelle des Textes eine Grafik anzeigen, die darüber informiert, ob der zugehörige Gegenstand sich im Inventar befindet. Abb. 6.2 zeigt das geöffnete Inventarmenü nachdem der Spieler das Star-Item eingesammelt hat.

Abb. 6.2. Inventarmenü

Sollte das dem InventoryButton zugeordnete Item eingesammelt worden sein, kann der Spieler diesen benutzen, um das Item vor sich erscheinen zu lassen. Anschließend kann dieses wie bereits beschrieben herumgetragen oder erneut eingesammelt werden. Wie auch beim Teleport ist es aufgrund der zum Spieler relativen Position, an der die Items erscheinen sollen, notwendig, dass dies *nach* dem Synchronisieren von Vive und Perception Neuron geschieht. In Listing 5.4 ist der Aufruf der *Respawn(Vector3 respawnLocation)*-Methode für die Items zu sehen. Im

Zusammenhang der Itemmechaniken werden weiterhin die folgenden Skripte bzw. Klassen verwendet:

- **NeuronVR_InventoryDropZone** verwaltet das Rendern des Bauchbereiches, falls Items aufgehoben oder losgelassen werden.
- **NeuronVR_Item** ist den Item-Objekten angehangen und prüft mithilfe von Trigger-Events, wann ein Item sich in der InventoryDropZone befindet. Lässt der Spieler das Item los, wird die Methode *Release()* aufgerufen, in der ggf. dem MenuManager mitgeteilt wird, dass ein Item in das Inventar abgelegt wurde, um den entsprechenden InventoryButton anzupassen.
- Die Trigger-Collider an den Händen des Spielers werden vom **NeuronVR_Hand**-Skript dazu verwendet, eine Kollision mit Items festzustellen. Der MenuManager prüft anhand der Knochentransformationen, ob die Hände eine Greifgeste zeigen, und aktualisiert darauf hin das NeuronVR_Hand-Skript über den Aufruf der *SetGrabbing(bool grabbing)*-Methode. Da das Skript verwaltet, mit welchem Item die Hand kollidiert und ob im letztem Frame ein Greifen erkannt wurde, kann über den aktualisierten Greifzustand *grabbing* entschieden werden, ob der Spieler ein Item aufhebt oder loslässt.

6.2 Ausrüstung & Events

Neben den Items wurden als weitere Mechanik Schwert und Schild implementiert, die als Ausrüstungsgegenstände dienen. Um die Gegenstände zu benutzen, muss der Spieler sie greifen und festhalten. Sie werden dabei im Gegensatz zu den Items nicht frei mit der Hand mit bewegt, sondern rasten an vorgegebenen Positionen ein. Der Schild kann nur von der linken Hand benutzt werden und wird an den linken Unterarm positioniert; analog dazu dient die rechte Hand zum Greifen des Schwertes, welches in selbiger gehalten wird. Die Ausrüstungsgegenstände sollen durch die natürlichen Bewegungen bei ihrer Verwendung als immersionsverstärkendes Element wirken. Das Schwert benötigt der Spieler, um gegenerische Charaktere zu besiegen, indem er damit zuschlägt oder -sticht. Mithilfe des Schildes, den der Spieler vor seinen Körper halten muss, können Kanonenkugeln abgewehrt werden. Um die Immersion zusätzlich zu erhöhen, verschwinden die Ausrüstungsgegenstände beim Ablegen nicht ohne weiteres, sondern werden an vorgesehene Stellen des virtuellen Körpers positioniert, die der realen Verwendung von Schwert und Schild entsprechen. So trägt der Spieler den Schild am Rücken, während sich das Schwert an der linken Hüfte befindet. Um die Gegenstände erneut zu verwenden, muss der Spieler an Rücken bzw. Hüfte fassen und so eine weitere natürliche Bewegung ausführen. Dies soll dem Spieler ein Gefühl für den eigenen Körper in der virtuellen Welt vermitteln und ihm eine Erfahrung ermöglichen, die durch andere VR-Anwendungen aufgrund der beschränkten Controller-Steuerung nicht umsetzbar ist. In Abb. 6.3 ist die Verwendung von Schwert und Schild zu sehen.

Sämtliche in dieser Arbeit vorgestellten Mechaniken wurden in einem Prototypen kombiniert, um deren exemplarische Verwendung aufzuzeigen. Der Protoyp besteht aus verschiedenen Räumen, die dem Spieler jeweils eine implementierte

Mechanik näherbringen. Zur Informationsvermittlung wird dabei auf die bereits vorgestellten HoverInfos zurückgegriffen. Um die Verwendung der HoverButtons zu erzwingen, wurden Türen in die Gänge zwischen den Räumen platziert. Da die Fläche unter einer solchen Tür nicht zum Teleportieren genutzt werden und der Spieler aufgrund der Länge der Fläche nicht durch die Tür durchlaufen kann, muss dieser die Tür durch Aktivieren eines HoverButtons öffnen. Der finale Raum wird durch eine große, dreiteilige Tür versperrt, die erst geöffnet werden kann, wenn der Spieler die drei Items in dafür vorgesehene Sockel ablegt. Ein Item findet der Spieler auf dem Weg zur großen Tür, die übrigen werden durch das Abschließen von Events zugänglich. In den Eventräumen findet der Spieler jeweils einen der Ausrüstungsgegenstände. Durch Aufheben des Gegenstandes schließt sich die Tür des Raumes und das Event wird gestartet. In einem der Räume findet der Spieler das Schwert, womit er eine Gruppe gegnerischer Charaktere besiegen muss. Der andere Raum enthält zwei Reihen von Kanonen, die den Spieler nach Start des Events beschießen. Mithilfe des Schildes kann der Spieler die Kanonenkugeln abwehren und zurück auf die Kanonen lenken, wodurch sich diese zerstören lassen. Sind alle Gegner und Kanonen besiegt, erscheint im Raum jeweils ein Item und die Tür öffnet sich wieder. Anschließend kann der Spieler zurück in den Raum mit den Sockeln und mithilfe der eingesammelten Items die große Tür öffnen.

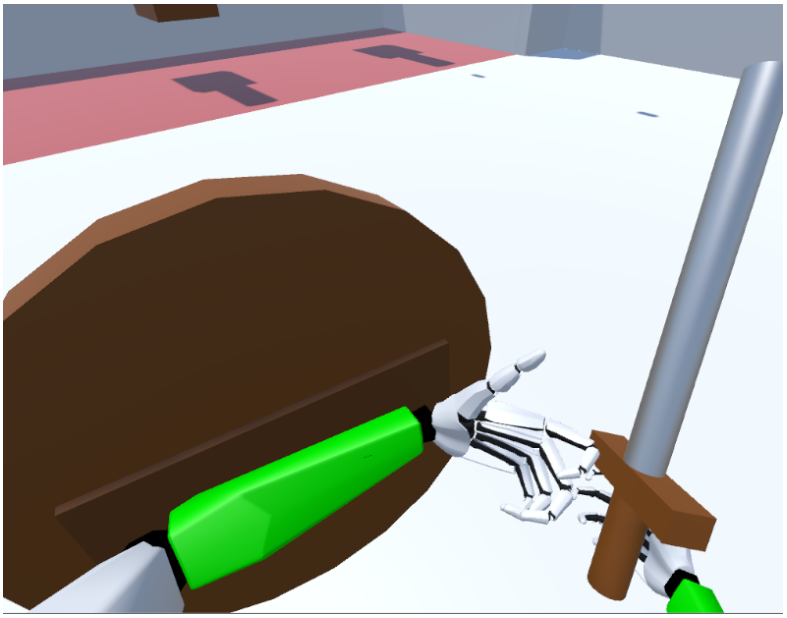

Abb. 6.3. Ausrüstung: Schild & Schwert

7

Zusammenfassung und Ausblick

Das zu Beginn dieser Arbeit formulierte Ziel, mithilfe des Motion Capture Systems eine besonders immersive VR-Erfahrung zu entwickeln, konnte trotz aufgetretener Probleme in Teilen erreicht werden. Neben den technischen Rahmenbedingungen wurde zunächst eine beispielhafte Verwendung der einzelnen Hardware Komponenten HTC Vive und Perception Neuron aufgezeigt. Zur Synchronisierung der Komponenten und grundsätzlicher Steuerung mittels einer Kombination von Controllern und Körpererfassung wurden umfangreiche theoretische Überlegungen getroffen. Die anschließende Umsetzung ist jedoch an der unzureichenden Genauigkeit der von Perception Neuron erzeugten Bewegungsdaten gescheitert. Ob diese Ungenauigkeiten auf die Hardware oder eine fehlerhafte Verwendung zurückzuführen sind, kann nicht abschließend bewertet werden. Dies ist nicht zuletzt auf die insbesondere für professionell angebotene Technik- und Softwareprodukte mangelhaften Dokumentationen zurückzuführen, die zudem die Einarbeitung in die Thematik erschwert haben.

Durch den Verzicht auf die Controller und die Konzeption einer angepassten Steuerung ist es jedoch gelungen, die genannten Probleme zu umgehen und eine ausreichend überzeugende Interaktion allein mithilfe des Motion Capture Systems zu implementieren. Dabei konnten wichtige Mechaniken, wie Teleportieren und Verwalten eines Inventars, auf Grundlage dieser Steuerung umgesetzt werden. Die modular aufgebauten Menüs und die HoverButtons und -Infos erlauben es zudem, den Prototypen auf einfache Weise zu erweitern oder die Steuerung mit geringem Aufwand auf ein neues Projekt zu übertragen. Das durch die Gestenerkennung ermöglichte Greifen der Items und die Implementierung von Schwert und Schild, die in ihrer Verwendung natürliche Bewegungen erfordern, bieten dem Nutzer Interaktionsmöglichkeiten mit der Umgebung, die sich in ihrem Grad der erzeugten Immersion in positiver Weise von üblichen VR-Anwendungen unterscheiden.

Eine produktive Verwendung der hier entwickelten Mechaniken ist jedoch aufgrund der weiterhin bestehenden Probleme mit der Hardware nicht zu empfehlen. So ist die Erkennung der Greifgesten trotz großer Toleranzbereiche zeitweise so unzuverlässig gewesen, dass die Verwendung der Ausrüstungsgegenstände nicht möglich war. Ein Endkunde erwartet jedoch, dass die Steuerung zu jedem Zeitpunkt einwandfrei funktioniert. Daher sollten weiterführende Arbeiten sich primär damit befassen, eine ausreichende Zuverlässigkeit der Steuerung zu erreichen. Auf-

grund der hohen Dynamik des gesamten VR-Bereiches und der laufenden Entwicklungen an Perception Neuron und SteamVR ist es nicht unwahrscheinlich, dass die in dieser Arbeit bestehenden Probleme zukünftig bereits von den Herstellern behoben werden. Darauf aufbauend könnte insbesondere eine weiterentwickelte Gestensteuerung, die bspw. auf mathematischen Modellen basiert, eine sinnvolle Erweiterung der hier implementierten Steuerung darstellen. Auch könnten vollständigere Dokumentationen künftigen Autoren dabei helfen, die Ansätze dieser Arbeit im Bezug auf die technischen Voraussetzungen umfassender zu bewerten, als dies zum aktuellen Zeitpunkt möglich ist.

Literaturverzeichnis

Koa. *The Koalition - Website.* http://thekoalition.com/2015/hands-on-with-the-perception-neuron-motion-capture-vr-system, abgerufen am 25.09.2017.

Neua. *Perception Neuron - Axis Neuron Manual.* https://neuronmocap.com/content/axis-neuron-standard, abgerufen am 25.09.2017.

Neub. *Perception Neuron - Online Shop.* https://neuronmocap.com/content/product/32-neuron-alum-edition, abgerufen am 25.09.2017.

Neuc. *Perception Neuron - Website.* https://neuronmocap.com/products/perception_neuron, abgerufen am 25.09.2017.

Opt. *OptiTrack - Website.* http://optitrack.com/systems/#animation/flex-3/6, abgerufen am 25.09.2017.

VrJ. *VR JUMP - Website.* https://vrjump.de/lighthouse-erklaert, abgerufen am 25.09.2017.

Wika. *Wikipedia - Online Lexikon.* https://en.wikipedia.org/wiki/HTC_Vive, abgerufen am 25.09.2017.

Wikb. *Wikipedia - Online Lexikon.* https://de.wikipedia.org/wiki/HTC_Vive, abgerufen am 25.09.2017.

Wikc. *Wikipedia - Online Lexikon.* https://de.wikipedia.org/wiki/Motion_Capture, abgerufen am 25.09.2017.

Wis. *University of Wisconsin-Madison - Website.* https://research.cs.wisc.edu/graphics/Courses/cs-838-1999/Jeff/BVH.html, abgerufen am 25.09.2017.

A

Erklärung der Kandidatin / des Kandidaten

☐ Die Arbeit habe ich selbstständig verfasst und keine anderen als die angegebenen Quellen- und Hilfsmittel verwendet.

☐ Die Arbeit wurde als Gruppenarbeit angefertigt. Meine eigene Leistung ist ...
Diesen Teil habe ich selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.
Namen der Mitverfasser: ...

Datum

Unterschrift der Kandidatin / des Kandidaten