FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Virtual Reality Content Generator

A Generic Content Creation and Placement Tool for Simulations and Storytelling in Unity 3D

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Medieninformatik und Visual Computing

eingereicht von

BSc. Tobias Froihofer
Matrikelnummer 0828457

an der Fakultät für Informatik
der Technischen Universität Wien
Betreuung: Ao.Univ.Prof. Mag. Dr. Horst Eidenberger

Wien, 29. September 2015

Tobias Froihofer	Horst Eidenberger

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Virtual Reality Content Generator

A Generic Content Creation and Placement Tool for Simulations and Storytelling in Unity 3D

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Media Informatics and Visual Computing

by

BSc. Tobias Froihofer
Registration Number 0828457

to the Faculty of Informatics

at the Vienna University of Technology

Advisor: Ao.Univ.Prof. Mag. Dr. Horst Eidenberger

Vienna, 29th September, 2015

Tobias Froihofer	Horst Eidenberger

Erklärung zur Verfassung der Arbeit

BSc. Tobias Froihofer
8654 Fischbach, Fischbach 69A

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 29. September 2015

Tobias Froihofer

Danksagung

An dieser Stelle möchte ich mich bei allen Personen bedanken, die mich bei der Erstellung dieser unterstützt haben.

Ganz besonders gilt dieser Dank Herrn Prof. Dr. Horst Eidenberger, der meine Arbeit betreut hat.

Auch möchte ich mich bei meinen Studienkollegen Luca Maestri und Juri Berlanda bedanken. Ohne sie wäre diese Arbeit nicht zustande gekommen.

Ebenfalls bedanke ich mich bei meinen Freunden, die während des letzten halben Jahres auf viel gemeinsame Zeit verzichten mussten und stets ein offenes Ohr für mich hatten.

Nicht zuletzt gebührt meinen Eltern Dank, ohne die dieses Unternehmen bereits von vornherein nicht möglich gewesen wäre.

Kurzfassung

In den vergangenen Jahren haben Anwendungen im Bereich der virtuellen Realität einen starken Zuwachs erfahren. Dies ist vor allem durch die verstärkte Forschung im Bereich der 3D-Brillen und anderer Hardware bedingt und dieser Trend wird sich vermutlich in den nächsten Jahren fortsetzen. Ziel dieser Arbeit war es, ein geeignetes Werkzeug zu erstellen, mit dem virtuelle Simulationen effizient erzeugt werden können. Das in der vorliegenden Arbeit beschriebene System, wurde in der Entwicklungsumgebung Unity3D erstellt. Neben einem System zur Erzeugung eines abgerundeten Pfades werden von diesem Werkzeug Methoden bereit gestellt, um Objekte mit multimedialen Inhalten zu füllen und diese so auf dem Pfad zu positionieren, dass ein sich vorbei bewegender Benutzer alle Informationen aufsammeln kann, ohne dass es dabei zu Überlagerungen des multimedialen Inhaltes kommen kann. Zusätzlich können die Informationsträger mit Verhaltensweisen versehen werden, um diese für die Gegebenheiten der jeweiligen Simulation anzupassen. Das Werkzeug wurde im Verlauf mehrerer Monate erstellt und regelmäßig an einer bestehenden Simulation, in der virtuelle Fallschirmsprünge gemacht werden können, getestet und weiterentwickelt. Schließlich wurde das System an Unity-Entwickler weitergereicht, sowie auf Verwendbarkeit und Einfachheit überprüft. Die Entwickler konnten sich schnell in das System einarbeiten und innerhalb geringer Zeit eine virtuelle Simulation erstellen, in der sich vier Informationsträger befinden. Mit dem bestehenden System lassen sich viele verschiedene Simulationen, in denen sich der Benutzer auf einem vordefinierten Pfad bewegt und währenddessen Informationen präsentiert bekommt, schnell und einfach erzeugen.

Abstract

In the recent years, applications in the area of virtual reality have become very popular. This is due to the fact that researchers are developing ever more efficient hardware and 3D-glasses. We assume that this trend will keep up for the next years. Likewise there is a trend for the research of virtual simulations at Universities. Therefore it has become necessary to develop a tool, which can be used to efficently create virtual reality simulations. The system described in this work was created in the developement environment Unity3D. Besides a system which can handle the creation of a smooth path in a threedimensional space, a method is being introduced to fill gameobjects with multimedia-based content and to place them on the path in such a way that a bypassing user can perceive all the information without having to concentrate on more than one object at a time. Additionally the gameobjects can be equipped with different types of behaviour required by the current simulation. This system has been under developement for several months and was regularly tested and improved in an existing simulation in which users can perform a virtual parachute jump. Finally, the system was introduced to developers who were familiar with Unity3D. It was tested, how well the developers could handle the system and if it was intuitive. The results showed that the developers were able to create a new simulation in an existing environment in a short amount of time. The system allows developers to easily create a variety of simulations in which the user follows a predefined path while perceiving information.

Inhaltsverzeichnis

KAPITEL 1

Einleitung

Das Ziel dieser Diplomarbeit ist es, ein System zu entwickeln, mit dessen Hilfe es möglich ist, virtuelle Simulationen mit wenig Aufwand zu erstellen. Zu diesem Zweck ist es erforderlich, ein Werkzeug bereit zu stellen, mit dem der Softwareentwickler zumindest folgende Möglichkeiten hat.

- Einen vordefinierten, geglätteten Pfad in einem dreidimensionalem Raum erstellen, dem ein Spieler-Objekt folgen kann.
- Objekte in der virtuellen Welt mit multimedialem Inhalt (Bilder, Text und Audio) füllen und sie so zu Informationsträgern in der virtuellen Welt machen.
- Den multimedialen Inhalt und die Gesamtdauer der Simulation zeitlich begrenzen.
- Die Informationsträger so auf dem vordefinierten Pfad platzieren, dass während der Spieler den Pfad durchläuft keine Überschneidungen des multimedialen Inhalts auftreten.
- Spezifische Verhaltensweisen für die Objektträger deklarieren, um sie dynamisch an die Erfordernisse der Simulation anpassen zu können.
- Zusätzlich soll die Möglichkeit bestehen, auf einfache Art und Weise ein HMD (Head-Mounted Display, d.h. eine Virtual-Reality-Brille) zu integrieren, sodass die Simulation in 3D erlebt werden kann.

Die Notwendigkeit eines solchen Systems haben wir im Zuge einer Projektarbeit an der Technischen Universität Wien erkannt. Dabei wurde ein Simulator entwickelt, mit dessen Hilfe es möglich ist, virtuelle Fallschirmsprünge über der Stadt Wien zu absolvieren. Da dieses Projekt im Rahmen des 200-Jahr-Jubiläums der Technischen Universität Wien

entstand, wurde die Simulation auch dazu genutzt, dem Benutzer während des Fallschirmsprunges Informationen über die vergangenen 200 Jahre der Technischen Universität zu vermitteln. Dies geschah über die Darstellung von Bildern auf dreidimensionalen Objekten mit Musikuntermalung.

Während der Entwicklung dieses Fallschirmsprung-Simulators mussten wir als Entwickler leider feststellen, dass die korrekte Platzierung der Informationsträger eine sehr zeitraubende Angelegenheit war. Schließlich mussten die Objekte genau so in der virtuellen Szene platziert werden, dass es während des Fluges zu keinen Überschneidungen der Informationen kam. Kleine Änderungen an der Flugbahn des Fallschirmspringers hatten große Auswirkungen auf die Positionen der Informationsträger, sodass diese häufig aktualisiert werden mussten. Es wurde rasch erkannt, dass hierzu ein geeignetes Werkzeug benötigt würde, um Zeit zu sparen. Auch in Hinsicht auf weitere ähnliche Simulationen, wie Flüge im Weltall oder Tauchgänge in Ozeanen, sowie Museumssimulationen war klar, dass man in die Positionierung der Informationsträger nicht viel Zeit investieren wollte. Für komplexere Anwendungen würde es auch notwendig werden, die Flugbahn bzw. den Pfad, auf dem sich der Benutzer bewegt, besser anpassen zu können und dafür zu sorgen, dass der Pfad (der aus geraden Verbindungen zwischen vordefinierten Wegpunkten bestanden hat) schön geglättet würde.

Für die Umsetzung dieses Werkzeugs haben wir uns dazu entschieden, die Spiele-Engine Unity3D[1] zu verwenden. Bereits der Fallschirmsprung-Simulator wurde in derselben Entwicklungsumgebung entwickelt und so konnten wir auf unsere bereits gemachte Erfahrung zurückgreifen. Nachdem die Hauptursachen unserer Probleme erkannt waren, begannen wir mit einer tiefgreifenden Literaturrecherche, um Lösungsansätze in ähnlichen Anwendungen zu finden. Bedauerlicherweise ist unsere Anwendung sehr spezifisch, sodass zwar verwandte Arbeiten gefunden wurden, die unserer Applikation ähnlich sind, jedoch jeweils nur Teilgebiete davon abdecken. Vor allem über Informationsträger, die zeitlich genau abgestimmt auf einem vordefinierten Pfad liegen, konnten wir keine Literatur finden.

Unser Ansatz war, in Unity3D ein eigenes Werkzeug zu entwickeln, mit dem es möglich ist, auf einfache und schnelle Weise im dreidimensionalen Raum einen geglätteten Pfad zu definieren, auf dem sich der Benutzer entlang bewegt. Nachdem der Pfad festgelegt wurde, können Informationsträger-Objekte bestimmt werden, auf denen schließlich Bilder und Texte angezeigt werden. Auch das Abspielen von Hintergrundmusik wird durch diese Träger-Objekte möglich. Ein eigener Algorithmus wurde entwickelt, um diese Objekte passend auf dem vordefinierten Pfad zu platzieren, ohne dass es zu Überschneidungen der multimedialen Informationen kommt. Des Weiteren konnten über das Unity3D Komponentensystem Verhaltensweisen für die Informationsträger-Objekte erstellt werden, womit sich eine Vielzahl von Anwendungsmöglichkeiten in weiteren Simulationen eröffnet: Die Träger-Objekte können zum Beispiel nun auch zu dynamischen Umgebungsobjekten umgeformt werden, wie etwa sich bewegende Fahrzeuge, Fisch- oder Vogelschwärme, rotierende Planeten und Ähnliches.

[1] `https://unity3d.com/` (Zugriff am 06.08.2015)

Im Kapitel 2 diskutieren wir verwandte Arbeiten und beleuchten, inwiefern unsere Anwendung davon profitieren konnte. Auch geben wir einen Einblick in zwei häufig verwendete 3D-Spiele-Engines und deren Anwendungsbereiche. Spezifischer werden wir hier auf Unity3D eingehen, das auch wir zur Entwicklung unseres Content-Generators verwendet haben. Ein weiterer Punkt, auf den wir eingehen, ist Cybersickness, ihre Bedeutung und die Konsequenzen für unsere Anwendung. Abschließend diskutieren wir über Bezier-/und Spline-Kurven und über ihre Vor- und Nachteile im Hinblick auf die Erstellung von virtuellen Pfaden.

Im Kapitel Projektdefinition beschreiben wir, wie es zu der Entwicklung des Virtual Reality Content Generators gekommen ist und aus welchen Gründen dieser Generator so wichtig für unseren Anwendungsbereich ist. Des Weiteren gehen wir auf den Entstehungsprozess ein, sowie auf häufige Fehler, die in der Entwicklungsumgebung Unity3D gemacht werden.

Das Kapitel Systembeschreibung gibt eine genaue Beschreibung über die Funktionsweise des Content Generators und seine Komponenten. Genauer eingegangen wird auf das Pfadsystem des Benutzers, die Contentstruktur, sowie auf die Verhaltensmerkmale der Informationsträger-Objekte.

Im Kapitel 5 fassen wir unsere Arbeit und unsere Ergebnisse zusammen und geben einen Ausblick auf zukünftige Arbeiten und Verbesserungsmöglichkeiten, sowie auf bereits geplante weitere ähnliche Simulationen, wie den erwähnten Fallschirmsprung-Simulator.

KAPITEL 2

Literaturübersicht

Während der Literaturrecherche haben wir uns verstärkt darauf konzentriert, bestehende Arbeiten zu finden, die verwandte Themengebiete und Teilbereiche unseres geplanten Systems abdecken. Zu einem größeren Teil haben wir uns auf Anwendungen konzentriert, in denen auf Pfaderstellungen und Benutzerbewegungen in dreidimensionalen virtuellen Welten eingegangen wird [ea14a][ea08c][ea08b][YK14][RP06]. Des Weiteren konnten wir Arbeiten finden in denen Benutzer unterschiedliche Möglichkeiten haben, mit Objekten in einer virtuellen Umgebung zu interagieren und Informationen aus diesen zu extrahieren [ea08a][EP14]. Auch über die Effekte von Sound [ea12a] in virtuellen Umgebungen, sowie über eine ausgereifte Implementierung eines generischen objekt-erzeugenden Werkzeugs [Meg15] für Unity3D[1] konnten wir uns informieren.

In den Unterkapiteln '2.2' und '2.3' wird beleuchtet, welche Entwicklungssoftware für Spiele und Simulationen derzeit stark verbreitet sind und wozu sie außerdem verwendet werden können. Speziell auf die beiden Spiele-Engines Unity3D und Unreal Engine[2] werden wir detaillierter eingehen, da sie derzeit einen besonders großen Marktanteil besitzen.

Da wir in unserer Applikation mit einem HMD[3] (Head Mounted Display) arbeiten und die Benutzer in eine virtuelle Welt eintauchen lassen, müssen wir uns auch mit den Themen Virtuelle Realität und Cybersickness befassen. Es ist wichtig, dass wir es in unserer Anwendung vermeiden, Effekte zu erzeugen, die beim Anwender zum Beispiel zu Schwindelgefühlen oder Ähnlichem führen können. Dieser Punkt führte dazu, dass wir eine weitere Recherche in Richtung Bezier- und Spline-Kurven-Algorithmen gemacht haben, welche für unsere finale Pfaderstellung von großer Bedeutung war. In der Ausarbeitung gehen wir genauer auf die unterschiedlichen Merkmale von Bezier- und Spline-Kurven ein.

[1] `https://unity3d.com/` (Zugriff am 05.07.2015)
[2] `https://www.unrealengine.com/` (Zugriff am 05.07.2015)
[3] `https://de.wikipedia.org/wiki/Head-Mounted_Display` (Zugriff am 05.07.2015)

2.1 Verwandte Arbeiten

In der Recherche nach verwandten Arbeiten konzentrierten wir uns zunächst darauf, welche Lösungen für eine Pfaderzeugung es gibt. Auf dem erstellten Pfad sollte sich das Spieler-Objekt bewegen können. Außerdem wollten wir erfahren, welche Arten von Bewegungen für die Aufnahme und Verarbeitung von Informationen während einer virtuellen Reise von Vorteil sind, da wir in unseren Applikationen auch Bild, Text und Ton verwenden wollten.

Jean-Baptiste Barreau et al. haben in ihrer Arbeit [ea14a] auf Unity3D's Navigationstool zurückgegriffen. Dieses erzeugt basierend auf der vordefinierten Landschaft in der virtuellen Welt eine Navigationskarte, auf der sich Objekte auf dem Boden bewegen können. In der ersten Anwendung wurde für die Bewegung durch den dreidimensionalen Raum ein Joystick bzw. eine 3D Maus verwendet. Damit konnte der Benutzer die Richtung, in der er sich bewegt, beeinflussen und sich frei durch die Luft bewegen. Eine andere Funktion ermöglichte es den Benutzern, eine so genannte 'Point of Interest Navigation' durchzuführen. Das heißt, es sind bereits einige wichtige Standorte in der virtuellen Szene vorgegeben. Will der Anwender zu einem dieser Punkte hinreisen, wird durch das Navigationstool automatisch ein passender Weg berechnet, sodass die virtuelle Kamera zu diesem Punkt reisen kann (siehe Abbildung 2.1). Als Reisegeschwindigkeit wurde hier Schritttempo gewählt.

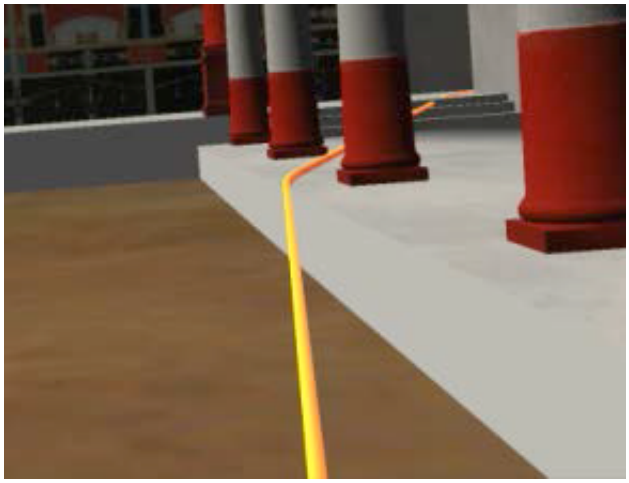

Abbildung 2.1: Ein Wegpunktsystem. Der Benutzer bewegt sich auf der gelben Linie zum nächsten Wegpunkt. [ea14a]

Entwickelt wurde diese Anwendung, um Archäologen die Möglichkeit zu bieten, Kulturerbe-Stätten untersuchen zu können, ohne sich physisch vor Ort begeben zu müssen. Die freie Navigation über den Joystick bzw. über die 3D-Maus fühlte sich laut Angaben der Testpersonen sehr natürlich an. Wie in Abbildung 2.2 zu sehen, kamen bei diesem Projekt auch 3D-Brillen zum Einsatz, um den Benutzern einen guten Eindruck der dreidimensionalen Szenen zu bieten.

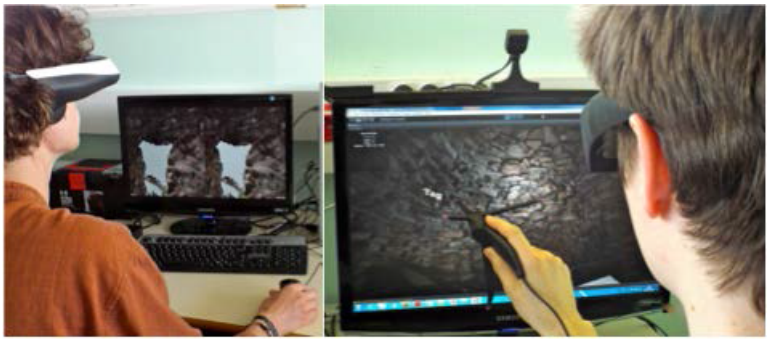

Abbildung 2.2: Mithilfe einer VR-Brille kann die virtuelle Umgebung in 3D erlebt werden. [ea14a]

Eine andere Art der Bewegung und Interaktion mit Objekten in einer virtuellen Umgebung haben Jie Jiang und Yang Kuang [YK14] implementiert. Der Benutzer kann sich mithilfe der Tastatur und der Maus in der Szene bewegen. Eine große Videoleinwand kann durch das Betätigen der Leertaste ein- und ausgeschaltet werden. Auch mit anderen Objekten, wie mit einer Flagge, kann interagiert werden, indem das Objekt angesehen und die Maustaste gedrückt wird.

Unsere weitere Recherche führte uns zu dem Thema 'Effekte von aktiver und passiver Bewegung'. Wie Gaën Plancher et al. [ea08b] in ihrer Publikation berichten, gab es dazu bereits mehrere Experimente. Es wird dabei untersucht, ob sich Probanden leichter an Dinge erinnern können, wenn sie aktiv an einem Gerät interagieren - und so den Vorgang selber steuern - oder ob es besser ist, passiv am Geschehnis teilzuhaben. Auch Grégory Wallet et al. [ea08c] haben in diese Richtung Forschungen betrieben. Sie haben 11 Experimente zwischen 1995 und 2007 untersucht und konnten feststellen, dass bei sieben davon das aktive Erleben von virtuellen Welten einen Vorteil für das Gedächtnis brachte. Vor allem war es für die aktiven Probanden leichter, sich zum Beispiel an das Layout der virtuellen Welt zu erinnern. Auch die Wegfindung, oder das Merken von Objekten war für die aktiven Teilnehmer leichter. In ihrem eigenen Experiment [ea08c] konnten Grégory Wallet et al. vermerken, dass es in einer virtuellen Umgebung eine Rolle spielt, ob man sich aktiv oder passiv bewegt. Aktive Interaktion mit dem System brachte bessere Ergebnisse hervor als passive. Auch Gaën Plancher et al. [ea08b] konnten ähnliche Ergebnisse erbringen. Sie untersuchten die Auswirkungen von aktiver und passiver Bewegung auf Personen unterschiedlichen Alters. Die Jüngeren waren durchschnittlich etwa 21 Jahre alt, die Älteren etwa 63 Jahre. Es wurde festgestellt, dass es für beide Gruppen in etwa gleich leicht war, sich Objekte in der virtuellen Welt zu merken. Allerdings war es für die ältere Generation schwieriger, die Reihenfolge und den Standort der Objekte in der virtuellen Umgebung im Nachhinein zu bestimmen.

Laut diesen Forschungsergebnissen können wir davon ausgehen, dass aktive Bewegung und Kontrolle für das Merken von Informationen besser ist. Außerdem fällt es jüngeren Menschen leichter, sich an die Standorte von spezifischen Objekten in einer Szene zu

erinnern. In den gegenständlichen Simulationen, können wir den Benutzern leider keine tiefgreifenden Interaktionen ermöglichen, da die Anwender keine Eingabegeräte erhalten werden - außer einer 3D Brille. Damit lässt sich zwar immer noch die Blickrichtung steuern, allerdings kann der Weg durch die Welt nicht beeinflusst werden.

Der wichtigste Punkt in unserer Literaturrecherche war nun, wie wir einen solchen Weg durch die virtuelle Welt erzeugen sollten, sodass die Simulation sowohl für den entwickelnden Designer, als auch für den Anwender ein angenehmes Erlebnis wird. Um einen solchen Pfad vordefinieren zu können und bei jedem Durchlauf der Simulation dasselbe Ergebnis zu haben, ist es wichtig, so genannte Wegpunkte zu definieren, zu denen sich die virtuelle Kamera sequentiell bewegen kann.

Es ist natürlich sehr wichtig, dass während der Simulation keine ungewollten unrealistischen Effekte auftreten, damit der Eindruck, dass man sich in einer wirklichen Welt befindet, gewahrt werden kann. Ein solch unerwünschter Effekt ist, dass sich die virtuelle Kamera, die sich durch den Raum bewegt, durch eine Mauer oder ein anderes stabiles Objekt hindurch bewegt. Da man in der Realität selbstverständlich nicht durch Wände gehen kann, ist es wichtig, nur solche Pfade zwischen den Wegpunkten zu erzeugen, die nicht durch feste Objekte durchgehen. Ross Ptacek und John K. Johnstone [RP06] haben sich in ihrer Arbeit mit diesem Problem befasst. Auch in ihrer Anwendung ist der Kamerapfad bereits durch Wegpunkte vordefiniert und somit bekannt, zu welchen Punkten sich die Kamera bewegen muss. Ihr Ansatz ist, neue Wegpunkte dort einzufügen, wo Kollisionen mit Objekten in der virtuellen Welt entstehen. Wenn in der Umgebung der Kollision nun immer weitere Wegpunkte eingefügt werden, wird der Pfad irgendwann kollisionsfrei sein und die Kamera wird sich nicht mehr durch Wände hindurch bewegen. Ohne Optimierungen führt dieser Ansatz allerdings dazu, dass sich die Kamera sehr nahe an den Wänden vorbei bewegt, was nicht immer gewollt ist. Es ist wichtig, ausreichend Wegpunkte (mit ausreichen Abstand zu festen Objekten) zu setzen, um solche Effekte zu vermeiden.

Der Ansatz von Ross Ptacek und John K. Johnstone [RP06] schien zwar sehr erfolgversprechend zu sein, allerdings ist der Algorithmus auch etwas fehlerbehaftet. Zum Beispiel kann es unter Umständen passieren, dass zufälligerweise kein passender Weg um ein Objekt gefunden werden kann, oder dass der Algorithmus in die falsche Richtung arbeitet und einen längeren Weg findet, als notwendig wäre. Auch die Tatsache, dass der neue Pfad sehr nahe an Wänden oder anderen Objekten vorbeiführen wird, war für uns Grund genug, nach weiteren Lösungswegen zu suchen. Im Projekt MegaShapes [Meg15] wird ein umfangreiches Werkzeug für Unity3D entwickelt, mit dessen Hilfe auf einfache Weise sehr komplexe Gebilde, die auf 3D-Bezier-Splines basieren, erstellt werden können. So lassen sich lange Bezierkurven mithilfe von vordefinierten Wegpunkten einfach und schnell erzeugen. Des Weiteren bietet MegaShapes noch weitere Funktionen, wie zum Beispiel das Generieren eines 3D-Gewebes entlang der zuvor definierten Bezierkurve (siehe Abbildung 2.4). Allerdings erschien uns bereits der vorgezeigte Arbeitsablauf in Unity3D für das Erstellen einer Bezierkurve als praktisch genug, um einen geeigneten Pfad für eine virtuelle Kamera erzeugen zu können. Daher entschlossen wir uns dazu, dieses System in Unity3D

nachzubilden und für unsere Simulationen zu nutzen - mit der Ausnahme, dass wir keine Bezierkurven, sondern Catmull-Rom-Splines verwenden. Weitere Informationen zu diesem Thema werden im Abschnitt über 'Bezier-/Spline-Kurven-Algorithmen' erläutert.

Abbildung 2.3: Mit den umfangreichen Werkzeugen von MegaFiers können Kurven und Animationen einfach erstellt werden. [Meg15]

Abbildung 2.4: Aus einer vordefinierten Bezierkurve lässt sich zum Beispiel eine Rennbahn erzeugen. [Meg15]

Nachdem nun eine Lösung für eine effektive Pfaderstellung gefunden war, stand noch die Frage im Raum, ob und wie eine Interaktion mit den Informationsobjekten, zu denen der Benutzer in der virtuellen Welt kommen kann, möglich sein sollte. Aufgrund der Tatsache, dass wir den Benutzer unsere Simulationen voraussichtlich ohne Tastatur und Maus durchleben lassen wollen, suchten wir nach alternativen Möglichkeiten, die Informationsobjekte interessant zu gestalten. Es sollte ein Lösungsansatz gefunden werden, der für möglichst viele unterschiedliche Simulationen von Nutzen sein kann.

Eva Pietroni und Andrea Adami [EP14] entwickelten ein System, mit denen man in einem Museum mit den dortigen Artefakten virtuell interagieren kann. Auch sie wollten es vermeiden, Maus, Keyboard, Joystick, Konsolen-Kontroller oder Ähnliches zu verwenden und setzten eher auf eine intuitivere Interaktionsart. Verwendet wurde schließlich ein Kinect Sensor[4], der die Bewegungen des Benutzers aufzeichnen und interpretieren kann und den Benutzer so mit den Objekten in der virtuellen Welt interagieren lässt (zu sehen in den Abbildungen 2.5 und 2.6). Erstellt wurde die Applikation in Unity3D. Mithilfe von Gestiken konnten sich die Museumsbesucher in der virtuellen Welt bewegen und mit den Objekten interagieren.

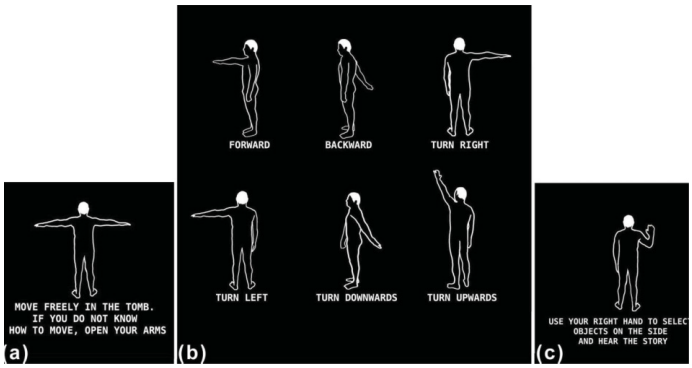

Abbildung 2.5: Den Museumsbesuchern wird über Grafiken angezeigt, welche Gestiken sie zur Interaktion mit der virtuellen Welt verwenden können. [EP14]

Abbildung 2.6: Über eine grafische Benutzerschnittstelle können die Museumsbesucher mit dem virtuellen Inhalt interagieren. [EP14]

Für eine andere Art der Visualisierung von und Interaktion mit dreidimensionalen Objekten und Daten entschieden sich A. L. Fuhrmann et al. [ea08a]. Mithilfe von HMDs und einem Projektor können erweiterte Erfahrungen bei der Präsentation von Diashows und anderen Anwendungen gemacht werden. Dies bietet einige Vorteile, da die Benutzer

[4] `https://www.microsoft.com/en-us/kinectforwindows/` (Zugriff am 06.07.2015)

mit den dreidimensionalen Objekten interagieren und diese von allen Seiten betrachten können, indem sie das Objekt einfach rotieren. Durch die Verwendung von mehreren HMDs können verschiedene Teilnehmer ein Objekt zur selben Zeit von anderen Seiten betrachten, was in bestimmten Situationen Vorteile bringt (siehe Abbildung 2.7). Für die Interaktion selbst wurde auf ein 'Personal Interaction Panel' (PIP) [ea08a] zurückgegriffen. Allerdings wollen wir es in unserer Simulation vermeiden, die Benutzerin zu zwingen, ihre Interaktion auf direktem Weg über Eingabegeräte zu machen. Deshalb ist dieses System leider nicht für unsere geplanten Simulationen geeignet.

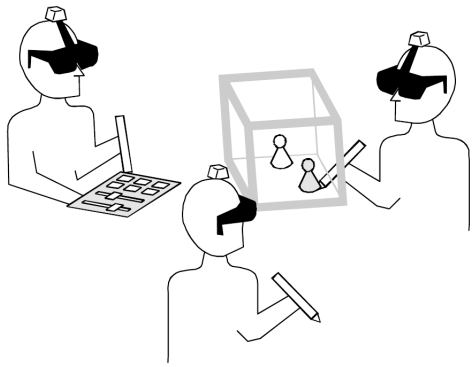

Abbildung 2.7: Mehrere Benutzer können gleichzeitig mit dem virtuellen Inhalt interagieren und sehen die Objekte aus unterschiedlichen Blickwinkeln. [ea08a]

Ein weiteres interessantes Thema, auf das wir während unserer Recherche gestoßen sind, ist der Zusammenhang von Sound und Distanzwahrnehmung im dreidimensionalen Raum. Brian Cullen et al. [ea12a] konnten in ihren Experimenten feststellen, dass ihre Versuchspersonen die Distanz zu einem sich bewegenden Objekt wesentlich besser einschätzen konnten, wenn das visuelle System auditiv verstärkt wurde. Dazu wurde eine Szene in Unity3D erstellt, in der sich ein Objekt auf die virtuelle Kamera zu bewegte. Die Testpersonen sollten nun erkennen, wann das Objekt einen bestimmten Punkt in der virtuellen Szene erreicht hatte. Mithilfe von Stereo Sound, sowie durch die Verwendung von Frequenzabfall konnte der Zeitpunkt, an dem das virtuelle Objekt einen Punkt erreicht hatte signifikant besser bestimmt werden.

Zusammenfassend half uns unsere Literaturrecherche bezüglich verwandter Arbeiten dahingehend weiter, als wir einige gute Lösungsansätze für das Pfaderstellungsproblem finden konnten. Besonders MegaFiers [Meg15] mit ihrem in Unity3D entwickelten Werkzeug MegaShapes wies in eine vielversprechende Richtung. Aber auch Jean-Baptiste Barreau et al. und ihre visuelle Repräsentation des Pfades mithilfe einer gelben Linie konnte schließlich zu unserem entwickelten Werkzeug beitragen. Durch die Forschungen über die Effekte von aktiver und passiver Bewegung [ea08c][ea08b] haben wir gelernt, dass aktive Interaktion für den Merkprozess einen signifikanten Vorteil bringt. Ross Ptacek und John K. Johnstone [RP06] illustrierten einen möglichen Algorithmus, um Kollisionen von Kamera und Umgebung zu vermeiden. Leider erschien uns dieser Algorithmus

für unsere Simulationen unpassend. Dennoch konnte uns der Ansatz, den Kamerapfad über mehrere Wegpunkte zu beschreiben in die richtige Richtung lenken. Des Weiteren konnten wir einige Einblicke in unterschiedliche Interaktionsarten - abseits von Tastatur und Maus - finden, in denen zum Beispiel Museumsbesucher mittels einer Kinect mit Objekten interagieren konnten [EP14]. Schließlich beschrieben Cullen et al. [ea12a], dass die Verwendung von Stereo in einer virtuellen Simulation den Benutzern helfen kann, Distanzen besser einzuschätzen.

2.2 3D Game Engines

Bevor ein Projekt in den Entwicklungsstatus übergehen kann, stellt sich immer die Frage, mit welchen Entwicklungswerkzeugen gearbeitet werden soll. Es ist sehr wichtig, diese Frage im Vorhinein zu klären, da ein Wechsel im späteren Stadium des Entwicklungsprozesses einen sehr großen Zeitverlust bedeutet. Auch muss man die Einarbeitungszeit in neue Werkzeuge mit einkalkulieren. Da es sich bei unserer Anwendung um ein Werkzeug handelt, mit dem 3D Simulationen schneller erstellt werden sollen und auch mit einer 3D Brille verknüpft werden soll, war schnell klar, dass wir das System am Besten in einer 3D-Spiele-Engine entwickeln sollten. Auf dem Markt befindet sich allerdings eine sehr große Auswahl, an ausgereiften Spiele-Engines [Wik15] und so mussten wir unsere Suche etwas eingrenzen.

Die Recherche und brachte nach kurzer Zeit zwei Favoriten zum Vorschein. Die wichtigsten Kriterien, die für unsere Bewertung von Bedeutung waren, liegen im Bereich der Finanzierung und der Benutzerfreundlichkeit. Es waren vor allem die Punkte Einstiegsfreundlichkeit, Support, Features und existierende Plugins und verfügbare Programmiersprachen entscheidend. Sehr wichtig war auch, dass es möglich sein sollte, auf einfache Weise eine 3D Brille, sowie die Kinect in ein Projekt integrieren zu können. So standen wir schließlich vor der Wahl zwischen den beiden Spiele-Engines Unity3D und Unreal Engine. Ihre Gemeinsamkeiten und Unterschiede werden in diesem Kapitel erläutert.

Beide Engines stehen mittlerweile mit all ihren wichtigsten Features zum freien Download zur Verfügung und können frei genutzt werden. Kosten entstehen lediglich für sehr erfolgreiche Entwickler. [May15] So verlangt Unity zum Beispiel, dass Entwickler, die im vergangenen Geschäftsjahr mehr als 100.000 Dollar eingenommen haben, sich die Pro Version der iOS- und/oder Android-Erweiterungen kaufen, um auf diesen Plattformen ihre Anwendungen veröffentlichen zu dürfen. Der Preis für beide beträgt jeweils 1.500 Dollar. Der Geschäftsplan von Unreal Engine sieht vor, dass Entwickler mit einem Umsatz von mehr als 3.000 Dollar fünf Prozent ihres Umsatzes an Epic Games abliefern. Da unser Projekt nicht auf Gewinnerbringung ausgelegt ist, betrifft uns dieser Punkt allerdings nicht.

Auch in anderen Punkten sind sich Unity und Unreal Engine sehr ähnlich. [Bar15] Um den logischen Ablauf der Applikationen zu verändern, verwendet Unity ein komponentenorientiertes System, bei dem auf C#- oder Java- basierende Skripte verwendet werden, um die Objekte im Spiel zu beeinflussen. Unreal Engine setzt hier auf die Programmiersprache

C++, sowie auf ein so genanntes Blueprint-System, mit dem man visuelle Logik in das Spiel bringen kann. Hierzu sind wenige bis keine Programmierkenntnisse erforderlich. Auch was den Umgang mit unterschiedlichen Spiel-Szenen betrifft funktionieren Unity und Unreal Engine auf ähnliche Weise. In Unity ist die Standardszene jene Szene mit dem Index 0, Unreal Engine legt die Standardszene in den Projekteinstellungen fest. Für jene, die von einer Engine zu der anderen wechseln ist wichtig, zu wissen, dass die Koordinaten-Achsen anders aufgebaut sind. So verwendet Unity die X-Achse für links/rechts, die Y-Achse für oben/unten und die Z-Achse für vorwärts/rückwärts. In der Unreal Engine steht die X-Achse für vorwärts/rückwärts, die Y-Achse für links/rechts und die Z-Achse für oben/unten. Die Verwaltung von Objekten in den einzelnen Szenen ist auch in beiden Engines in etwa gleich. Unity verwaltet Spiele-Objekte und weist ihnen Komponenten zu, um ihre Funktionen zu steuern. Unreal Engine besitzt so genannte Actors, welche ebenfalls über Komponenten ihre Funktionalität erhalten. Des Weiteren funktionieren die Standard-Datentypen, die Funktionen und Variablen, sowie die zur Verfügung stehenden Komponenten weitgehend gleich. Sie besitzen nur unterschiedliche Namen.

Weitere Aspekte, welche die beiden Pakete voneinander unterscheiden sind zum Beispiel die Grafik-Engines. Hier hatte lange Zeit Unreal Engine die Nase vorne. Allerdings mit der neuesten Veröffentlichung von Unity3D 5.0 am 3.März 2015 und dem damit neuen auf Physik basierenden Grafikshader konnte Unity einen großen Sprung nach vorne machen. Ein Pluspunkt für Unity ist auf jeden Fall, dass der existierende 'Asset-Store', in dem man Funktionalitäten erwerben kann, die andere Entwickler zum Kauf anbieten, wesentlich größer ist, als der 'Asset-Store' von Unreal Engine. Dies liegt vor allem daran, dass Unity mit etwa 45% einen weitaus größeren Marktanteil besitzt [Uni15a].

Es lässt sich feststellen, dass beide Spiele-Engines sehr gut dazu geeignet sind, 2D- und 3D Spiele, sowie Simulationen zu entwickeln. Letztendlich läuft es darauf hinaus, welche Entwicklungsumgebung von den Entwicklern bevorzugt wird. Auch gibt es Plugins für beide Engines, um 3D Brillen und die Windows Kinect integrieren zu können. Schließlich haben wir uns für Unity3D entschieden, da wir mit dieser Entwicklungsumgebung bereits Erfahrung hatten und aus unserer Sicht der Support seitens der Benutzer von Unity3D sehr gut war. Des Weiteren gibt es bereits sehr viele hilfreiche Tutorials und eine ausgereifte Dokumentation im Web zu finden. Unity3D war außerdem aus unserer Sicht Einsteiger-freundlicher.

2.3 Unity

Unity (oder Unity3D) ist eine Entwicklungssoftware, mit der es möglich ist, auf schnelle Art und Weise Spiele zu entwickeln. Das Hauptaugenmerk liegt dabei zwar auf Spielen, dennoch kann die Software auch dazu genutzt werden, andere Medien und Applikationen zu erstellen, wie in etwa Animationen, Simulationen, Produktvisualisierungen, Visualisierungen von Architektur, Hilfswerkzeuge für den Alltag und Vieles mehr. Entwickelt wurde die erste Version von Unity (die im Juni 2005 veröffentlicht wurde [Hel15]) von

drei Personen: David Helgason, Joachim Ante und Nicholas Francis [Bro15]. Die aktuelle Version (5.1.2) wird von Windows und Mac OS X unterstützt. Seit Unity 5.0 steht der Unity-Editor auch als 64-bit Version zur Verfügung und unterstützt so auch die Entwickler von größeren Projekten.

Im Hinblick auf die Zielplattformen hat sich seit dem Release der Erstversion sehr viel bei Unity getan. So konnte man mit der Version 1.0 im Juni 2005 seine Projekte lediglich auf Mac OS X exportieren. Mittlerweile ist es möglich, auf eine ganze Reihe an Plattformen [Uni15b] zu veröffentlichen, wie zum Beispiel Windows, Linux, Android, iOS, sowie auf Webplayer und Konsolen (PlayStation, XBox, Wii). Des Weiteren wird nun auch von offizieller Seite bereits Hardware im Bereich der virtuellen Realität, wie zum Beispiel Oculus Rift, unterstützt [Tec15]. Damit ist es relativ leicht möglich, VR Anwendungen zu erstellen, wie wir es auch in unseren Simulationen vorhaben.

In den nachfolgenden Kapiteln gehen wir auf die Benutzerschnittstelle des Unity-Editors, sowie auf die Grundkonzepte und Schemata in Unity ein. Auch werden wir erläutern, wie mit welchen Geräten man mit Unity ein virtuelles Erlebnis erzeugen kann.

2.3.1 Benutzerschnittstelle

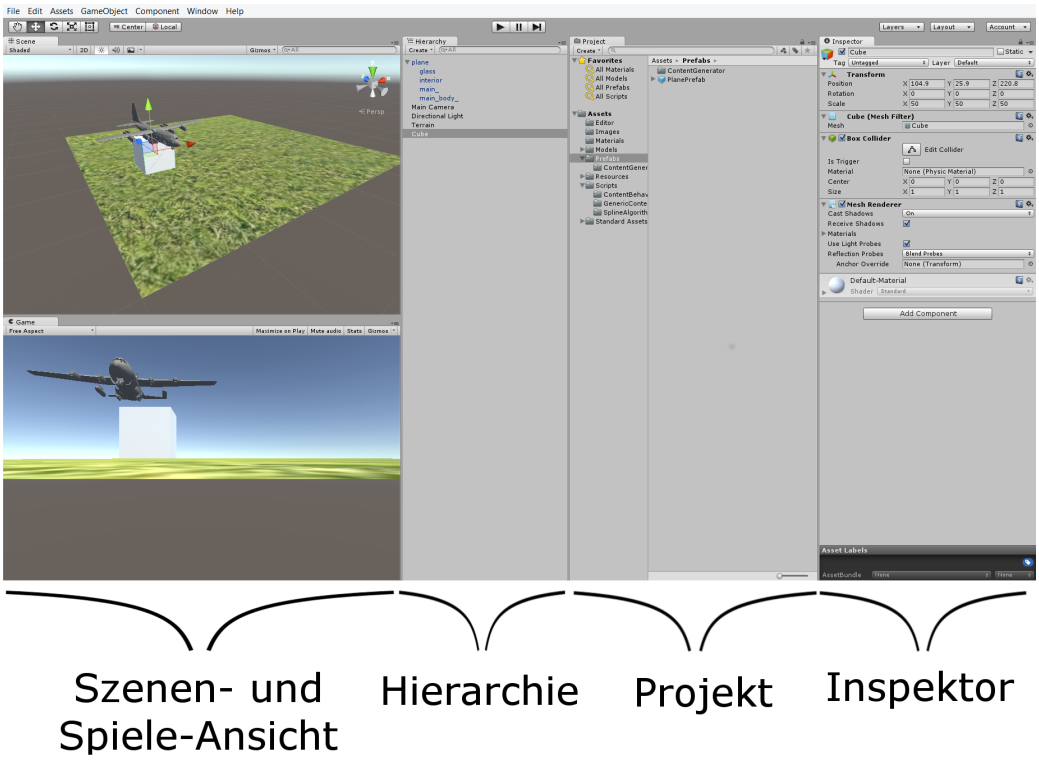

Abbildung 2.8: Die Benutzerschnittstelle der Entwicklungssoftware 'Unity3D'.

Die Benutzerschnittstelle des Unity-Editors ermöglicht es dem Entwickler, umfangreiche virtuelle Welten (Szenen) zu erstellen und diese mit Leben und Logik zu füllen. Das Layout der Schnittstelle lässt sich vom Benutzer sehr gut nach seinen Bedürfnissen anpassen und kann so umgestaltet werden, wie es für den Arbeitsablauf am Besten geeignet ist. Grundsätzlich lässt sich das Interface in vier Bereiche einteilen (siehe Abbildung 2.8).

- Die Szenen- und Spiele-Ansicht
- Die Hierarchie
- Das Projektfenster
- Der Inspektor

Die Szenen- und Spiele-Ansicht sind die beiden Fenster, auf die beim Designen der Umgebung und der Objekte im Spiel das Hauptaugenmerk gelegt wird. In der Szenen-Ansicht kann man mit der Maus frei navigieren, sowie Objekte erzeugen und bearbeiten. Mit den vorgegebenen Werkzeugen lassen sich die Objekte, die sich in der Szene befinden verschieben, rotieren und skalieren. Es besteht auch die Möglichkeit, in einen 2D-Modus zu wechseln, um auf einfache Weise 2D-Anwendungen erzeugen zu können. Dabei wechselt die Kamera von einer perspektivischen Ansicht in eine orthographische.

Die Spiele-Ansicht repräsentiert die in der Szenen-Ansicht erstellte Umgebung aus der Perspektive, welche die virtuelle Kamera in der Szene einnimmt (siehe Abbildung 2.9). Das heißt, der Entwickler erhält zu jedem Zeitpunkt direktes Feedback dazu, wie das Spiel/die Simulation im fertigen Produkt aussehen würde.

Im Hierarchie-Fenster (Abbildung 2.10) werden alle Objekte angezeigt, die sich aktuell in der Unity-Szene befinden. Dabei können die Objekte in der Hierarchie auch untereinander verschachtelt werden, was neben einer überschaubaren Auflistung auch weitere Effekte ermöglicht. Transformationen, die auf das Elternobjekt angewandt werden, haben zum Beispiel auch Auswirkungen auf die Kinderobjekte dieses Objektes. Wird daher beispielsweise ein Auto nach rechts bewegt und das Auto hat als Kinderobjekte Karosserie und Reifen definiert, werden diese gleich mit dem Auto in dieselbe Richtung bewegt. Ebenso verhält es sich mit Rotationen, und Skalierungen. Aus diesem Grund sollte ein Designer und Entwickler, der Unity benutzt, viel Wert auf diese Hierarchie legen und sie gut überlegt ordnen.

Im Projektfenster neben dem Hierarchie-Fenster sind alle Ordner und Dateien aufgelistet, die sich im Projektordner befinden und für die Spielumgebung von Bedeutung sind. Das heißt, neben C# und JavaScript Dokumenten finden sich hier unter Anderem auch Texturen und Bilder, sowie Audiodateien und 3D-Objekte, die für die Szenen verwendet werden können.

Das wichtigste Fenster neben der Szenen-Ansicht ist allerdings der Inspektor (Abbildung 2.11). Wie bereits im Abschnitt 2.2 erwähnt, ist Unity3D ein auf Komponenten basierendes

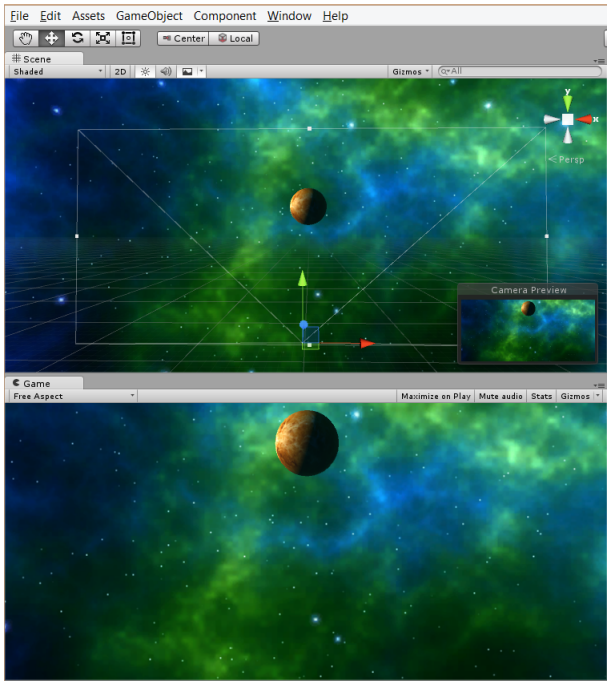

Abbildung 2.9: Die Szenen- und Spiele-Ansicht einer Szene in Unity.

System. Das heißt, Objekte in Unity können mit Komponenten beliebig erweitert werden und erhalten so ihre Funktionalität. Im Inspektor-Fenster werden diese Komponenten aufgelistet und können so auch bearbeitet werden. Ein leeres Spielobjekt besitzt lediglich die Transform-Komponente, mit der sich die Position, Rotation und Skalierung verändern lassen. Zusätzlich gibt es viele weitere Komponenten, die wir im nächsten Abschnitt 2.3.2 näher beleuchten werden. Wie wir in Abbildung 2.11 sehen können, besitzt das ausgewählte Objekt zusätzlich zu der Transform-Komponente auch noch drei weitere Komponenten (ein C# Skript, einen Rigidbody, sowie einen Box-Collider) auf die wir hier nicht weiter eingehen werden.

Die freie Version von Unity 5.1.2 bietet ebenfalls einige weitere hilfreiche Werkzeuge, wie zum Beispiel den Audio-Mixer, mit dessen Hilfe die Musik und die Soundeffekte innerhalb einer Szene auf professionelle Art und Weise geregelt werden können. Auch das Animations-Fenster ist ein äußerst wichtiger Bestandteil von Unity3D. Damit können 2D- und 3D-Objekte mit Animationen versehen werden. Sowohl Transformationen auf die Transform-Komponente, als auch Veränderungen an der grafischen Darstellung (z.B. verändern der Farbe und Deckkraft) und Vieles mehr sind dabei möglich. Mithilfe des 'Lighting'-Fensters kann die Beleuchtung der Szene verändert werden. Mögliche Parameter hier sind beispielsweise die Farbe der Umgebungsbeleuchtung, die Stärke der Lichtreflexionen und die Intensität des Dunstes[5] in der virtuellen Atmosphäre.

[5] https://de.wikipedia.org/wiki/Dunst_%28Atmosph%C3%A4re%29 (Zugriff am: 24.08.2015)

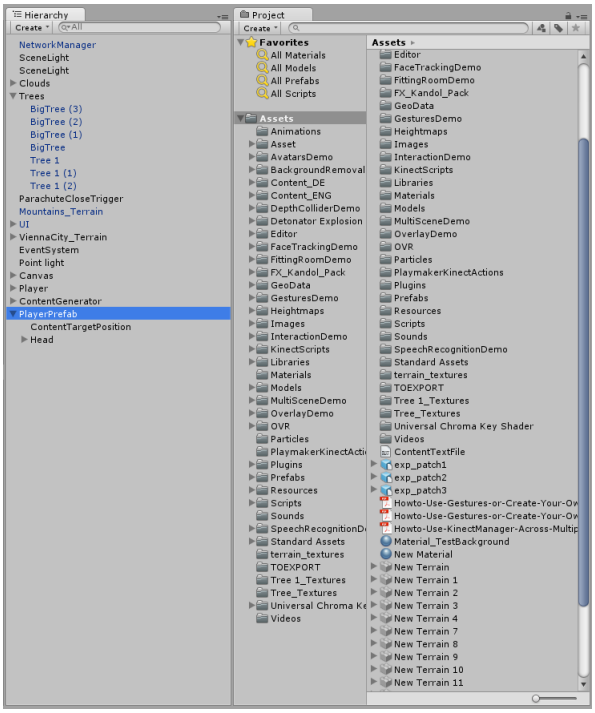

Abbildung 2.10: In der Hierarchie sind alle Objekte, die sich in der Szene befinden, abgebildet. Rechts daneben werden im Projektfenster alle Ordner und Dokumente innerhalb des Projektordners aufgelistet.

2.3.2 Konzepte und Schemata

In diesem Kapitel beleuchten wir genauer die Struktur einer typischen Unity-Anwendung, sowie die Konzepte, die eine Unity-Szene ausmachen. Dazu wird genauer auf die speziellen Merkmale eingegangen, die jeder Unity-Entwickler verstehen muss, um umfangreiche Applikationen erstellen zu können. Einige davon sind beispielsweise so genannte GameObjects, Prefabs, Komponenten, Materialien und Shader, Partikelsysteme, sowie Terrain, Beleuchtung und die Benutzerschnittstelle. Aber auch weitere Aspekte, wie die Physik Engine, das Layer-System und der Asset Store spielen eine große Rolle. Diese Punkte werden im Folgenden genauer erklärt.

Als GameObjects werden in Unity alle Objekte beschrieben, die sich in einer Szene befinden können und mit denen interagiert werden kann. Typische dreidimensionale GameObjects, die man erstellen kann, sind zum Beispiel Würfel, Spähren und Zylinder. Allerdings zählen hierzu auch Terrains und Bäume, sowie alle anderen dreidimensionalen Objekte, die durch ein so genanntes Mesh dargestellt und von Unity interpretiert werden können. Es ist möglich, die GameObjects hierarchisch zu verknüpfen und sie so zu ordnen. Außerdem ermöglicht dies, wie im Kapitel 2.3.1 erwähnt, die Anwendung weiterer Effekte. Alle Transformationen, die auf ein in der Hierarchie höher liegendes Objekt angewandt werden, wirken auch auf die darunterliegenden Objekte. Um dies genauer zu beschreiben

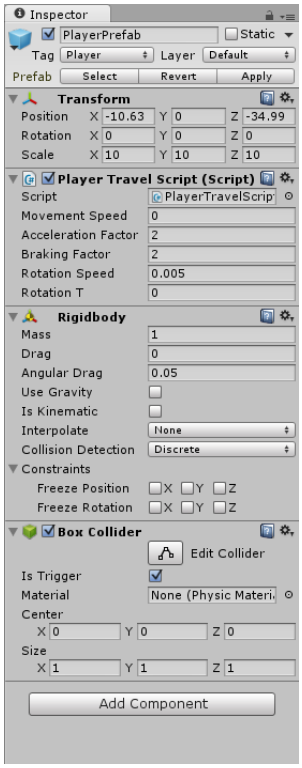

Abbildung 2.11: Im Inspektor können Komponenten zu Objekten hinzugefügt und bearbeitet werden.

dient uns die Hierarchie einer virtuellen Person, die wie in Abbildung 2.12 aufgebaut ist. Wird beispielsweise eine Translation auf das Spielobjekt 'Person' angewandt, verschiebt sich die gesamte Struktur. Eine Rotation des Oberkörpers hätte zur Folge, dass sich nicht nur der Oberkörper, sondern auch der Kopf, sowie die beiden Arme mitsamt ihren Kinder-Objekten mit dem Oberkörper mit rotieren würden. Das ist häufig genau das, was auch gewollt ist, weshalb dieses System in vielerlei Hinsicht sehr nützlich ist.

Es besteht die Möglichkeit, diese Datenstruktur in einem Projektordner als so genanntes Prefab abzuspeichern, wodurch es sehr einfach wird, eine zweite Person in einer Szene zu erzeugen, ohne diese komplett neu erstellen zu müssen. Zieht man das Prefab in die Szenen-Ansicht, oder in das Hierarchiefenster, wird sofort eine neue Instanz einer Person in der Szene erstellt und man kann diese bearbeiten. Dabei weist die Person zu Beginn dieselben Merkmale auf wie das Prefab, verliert jedoch bei Veränderungen seine Beziehung zum Prefab nicht. Es ist ganz wichtig, dass man das im Auge behält, da es sonst zu Problemen kommen kann. Wird eine Person beispielsweise verändert und man überschreibt das Prefab mit dieser neuen Person, so können auch alle anderen bereits erstellten Personen davon beeinflusst werden. Daher macht es häufig Sinn, für die veränderte Person ein eigenes, neues Prefab zu erzeugen.

Abbildung 2.12: Eine Objekt-Hierarchie der Körperteile einer Person in Unity.

GameObjects können mit Komponenten versehen werden, um sie mit Funktionalität auszustatten. Typische Komponenten sind beispielsweise Physik-Komponenten, wie Collider, die reagieren, wenn Objekte mit anderen Objekten in Berührung kommen. Aber auch Rigidbodies zählen zu den Physik-Komponenten. Sie ermöglichen es, dass auf ein Objekt Kräfte einwirken können, wie zum Beispiel die Schwerkraft. Weitere Komponenten dienen dazu, dem Objekt eine Farbe und Textur zuzuweisen, oder Animationen und die Wiedergabe von Sound möglich zu machen. Mit diesen Komponenten kann man zwar bereits einige interessante Szenen bauen, auf denen beispielsweise mehrere Objekte gegeneinanderprallen. Allerdings erhält der Benutzer durch sie alleine noch keine Interaktionsmöglichkeiten. Dazu ist es nötig, geeignete Skripte in C# oder in JavaScript zu verfassen und diese den GameObjects in der jeweiligen Szene zuzuweisen. Auch solche Skripte sind Komponenten in Unity und werden im Inspektor-Fenster aufgelistet. Wird ein Objekt als Prefab abgespeichert, werden ebenso alle Komponenten mit ihren aktuellen Werten mit gespeichert. Dadurch wird sichergestellt, dass jedes Objekt, das aus einem Prefab erzeugt wird, dieselben Funktionen besitzt.

Um einem Objekt Farben und Texturen zuweisen zu können, verwendet Unity Materialien. Diese können im Projektordner erstellt und schließlich Objekten als Komponenten zugewiesen werden. Vor der Veröffentlichung von Unity 5.0 gab es eine Reihe von unterschiedlichen Shadern, die auf ein Material angewandt werden konnten. Damit war es möglich, ein Objekt zum Beispiel rau oder glänzend aussehen zu lassen. Zusätzlich gab es Shader, die es ermöglichten, die Deckkraft eines Objektes zu verringern und es so durchsichtig oder unsichtbar zu machen. Mit der Einführung von Unity 5.0 wurden zusätzlich zu den bisherigen Shadern auf Physik basierende Shader veröffentlicht. [Uni15c] Das ließ die graphische Qualität von Unity-Applikationen einen großen Sprung nach vorne machen, wie man in Demonstrationsvideos[6] von Unity sehen kann. Die besondere Neuerung, die das neue Shader-System bringt ist, dass sich die Materialien an die Umgebungsbeleuchtung anpassen. Ändert sich zum Beispiel die Farbe des Himmels von

[6] `https://www.youtube.com/watch?v=pXWAsayTFTo` (Zugriff am 25.08.2015)

blau auf rot, werden auch die Farben aller Objekte in der Szene in einem rötlichen Ton erscheinen. Vor allem für Szenen, in denen häufig die Beleuchtung wechselt, bringt das enorme Vorteile. Nach der Einschätzung der Unity-Entwickler kann es für viele Projekte möglich sein, dass der neue Shader ausreicht, um alle Objekte in einer Applikation zu beschreiben [Uni15d]. Das ist durchaus realistisch, da in diesem einen Shader viele Features enthalten sind, wie beispielsweise die Möglichkeit, neben der Textur auch eine Normal Map, eine Height Map und eine Occlusion Map angeben zu können. Da dies sehr viel Rechenleistung erfordert, stehen auch spezielle Shader für mobile Geräte bereit.

Um natürliche Phänomene, wie Wasser, Feuer, Wolken, Sternschnuppen und Vieles mehr erzeugen zu können, stellt Unity ein ausgereiftes Partikelsystem zur Verfügung (siehe Abbildung 2.13). Dieses System erzeugt in einem vordefinierten Bereich standardmäßig kleine weiße Partikel. Allerdings können auch alle möglichen anderen Objekte erzeugt werden, die sich durch eine grafische Textur beschreiben lassen. Für Wolken oder einen Wasserfall etwa könnte man ein halb-transparentes weißes Gebilde verwenden. Um die Umgebung in einer Unity-Szene erstellen zu können, kann man auf ein Terrain-System, sowie Skyboxes zurückgreifen, wie es auch Jiang Jie et al. [ea11] in ihrer kurzen Studie gemacht haben. Ein Terrain erzeugt zunächst eine ebene Fläche, die man entweder manuell, oder mithilfe einer Heightmap verformen kann. In einem früheren Projekt [Eid15a] (Virtual Jump Simulator) konnten wir ein Terrain und eine Heightmap dazu verwenden, die Stadt Wien in Unity3D nachzubilden (siehe Abbildung 2.14).

Abbildung 2.13: Mithilfe eines Partikelsystems können sehr viele Effekte in Unity erzeugt werden. Den Partikeln können unter Anderem unterschiedliche Texturen zugewiesen werden, sodass beispielsweise ein Wasserfall, oder Wolken simuliert werden können. Oder, wie hier zu sehen, ein System, dass in einem kegelförmigen Bereich TU-Logos erzeugt.

Einen sehr wichtigen und nützlichen Punkt in Unity3D stellt der Asset Store[7] dar. Er ermöglicht es Entwicklern, sich untereinander auszutauschen und sich gegenseitig mit Werkzeugen und Objekten zu unterstützen. Zu diesem Zwecke bietet der Unity-Editor die Funktion, einzelne Features in einem Projekt zu exportieren und in einem Unity-Paket zusammenzufassen. Dieses Paket kann in den Asset Store hochgeladen und für andere Entwickler zum Kauf angeboten werden.

Wenn man die Konzepte und die Schemata betrachtet, die in Unity vorhanden sind und wie einfach diese für Einsteiger zu erlernen sind, ist es nicht besonders überraschend, dass

[7] `https://www.assetstore.unity3d.com/en/` (Zugriff am 25.08.2015)

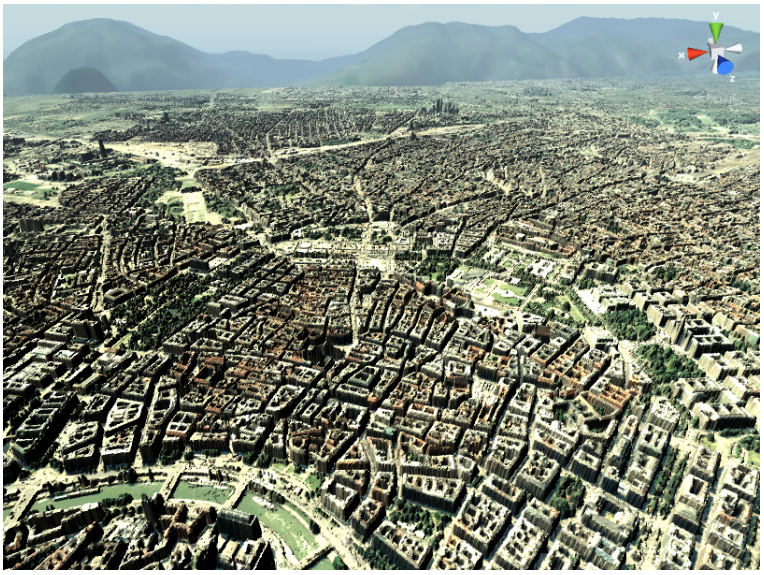

Abbildung 2.14: Mit dem Terrain-Werkzeug konnten wir die Stadt Wien in Kombination mit einer Heightmap modellieren.

Unity seit seiner Veröffentlichung im Juni 2005 [Hel15] immer mehr Entwickler anlockt. Im Jahr 2014 wurde die Zahl der Unity-Entwickler in etwa verdoppelt [Uni15a] und man darf davon ausgehen, dass dieser Trend in bestehen bleibt. Auch der Unity Asset Store trägt hier sicherlich einen großen Teil dazu bei.

2.3.3 Unity3D und Virtual Reality

Virtual Reality liegt seit einigen Jahren sehr stark im Trend. Immer mehr Firmen forschen an Brillen, die für Virtual Reality-Anwendungen verwendet werden können. Einige Beispiele sind Oculus Rift[8], Project Morpheus[9] von Sony, HTC Vive[10], Samsung Gear VR[11], Microsoft HoloLens[12] und FOVE VR[13]. Trendforscher vermuten, dass sich der Verkauf von Virtual-Reality-Brillen in den nächsten Jahren jährlich beinahe verdoppeln wird. [Ins15]

Auch in unseren Simulationen wollen wir die Benutzer mithilfe einer 3D-Brille in eine virtuelle Welt eintauchen lassen. Deshalb stand die Frage im Raum, welche Brille wir verwenden sollten. Unsere Anforderungen waren unter Anderem die Auflösung des

[8] https://developer.oculus.com/ (Zugriff am 26.08.2015)
[9] https://www.playstation.com/de-at/explore/ps4/features/project-morpheus/ (Zugriff am 26.08.2015)
[10] http://www.htcvr.com/ (Zugriff am 26.08.2015)
[11] http://www.samsung.com/global/microsite/gearvr/index.html (Zugriff am 26.08.2015)
[12] https://www.microsoft.com/microsoft-hololens/en-us (Zugriff am 26.08.2015)
[13] http://www.getfove.com/ (Zugriff am 26.08.2015)

Displays, wie ausgereift die Technologie bereits war, der Preis und die Einfachheit der Integration in unser System. Schon nach kurzer Überlegung fiel unser Hauptaugenmerk auf Oculus Rift (Developement Kit 2), da dafür bereits zum Zeitpunkt unseres Projektstarts im Oktober 2014 ein geeignetes Unity-Plugin [Ocu15] zur Verfügung stand. Außerdem war der Preis von 350 Dollar realistisch und auch die Qualität war überzeugend. Mithilfe des Plugins konnte die Oculus Rift schließlich auch relativ schnell integriert werden. Dadurch erhält man in Unity ein Prefab, das die Funktionen der 3D-Brille (die Bewegungen des Benutzers) interpretieren kann und so eine Interaktion mit dem System ermöglicht. Mittlerweile ist es noch einfacher, die Oculus Rift in Unity zu integrieren, da Unity seit der Version 5.1 VR-Geräte intern unterstützt [Tec15] und so keine Plugins mehr von Nöten sind.

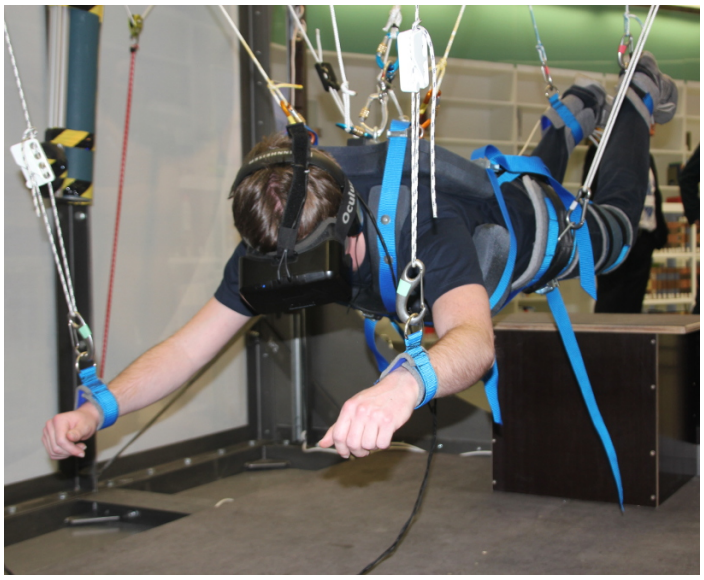

Abbildung 2.15: In unserem Projekt 'TU Jump into the Future' kann ein virtueller Fallschirmsprung durchgeführt werden. Der Springer trägt dazu eine Oculus Rift und wird zusätzlich von einer Kinect beobachtet, um mehr Interaktionsspielraum zu haben.

Neben der Immersion über eine 3D-Brille in eine virtuelle Welt gibt es noch weitere Möglichkeiten, um das Erlebnis aufregend zu gestalten. Ein Beispiel dafür ist Cyberith[14]. Dieses Projekt versucht, dem Spieler zusätzlich zur Oculus Rift noch weitere Interaktionen über Körperbewegungen zu ermöglichen. Das System besteht aus einer Oculus Rift für die Kamerasteuerung, einer Treadmill, um den Benutzer durch die virtuelle Welt laufen zu lassen und einer Wii Remote, um zusätzliche Interaktionen mit den Objekten in der Welt möglich zu machen. [Whi15] Auf ähnliche Weise haben wir ebenfalls versucht, dem Spieler weitere Interaktion in unserem Projekt 'TU Jump into the Future' [Eid15a] anzubieten, indem wir in unser Unity-Projekt eine Kinect integriert haben.

[14] `http://cyberith.com/` (Zugriff am 26.08.2015)

Virtual Reality wird sich in Zukunft mit Sicherheit noch weiterentwickeln und man darf gespannt sein, was sich in dieser Richtung noch tut. Unity3D ist definitiv ein geeignetes Werkzeug, um solche Anwendungen erstellen und den Benutzern so einige berauschende Erlebnisse zu ermöglichen.

2.4 Cybersickness

Unter Cybersickness versteht man im Allgemeinen den Umstand, dass während des Erlebens von Virtueller Realität bestimmte Symptome, wie zum Beispiel Schwindelgefühl, auftreten können. Da unsere Simulationen, wie bereits öfter angemerkt, die Benutzer in eine virtuelle Welt eintauchen lassen, waren wir gezwungen, dieses Thema genauer zu erforschen und Möglichkeiten zu finden, die Symptome möglichst gut zu bekämpfen. Informiert haben wir uns im Speziellen über verschiedene Theorien zu diesem Thema, als auch über die Symptome und Faktoren, die dafür verantwortlich sind, dass bestimmte Personen schlechter auf virtuelle Erfahrungen reagieren. Im Weiteren gehen wir auf die Konsequenzen ein und versuchen, mögliche Lösungs- und Verbesserungsvorschläge zu geben, um den Effekt von Cybersickness zu reduzieren.

2.4.1 Theorien

Auf der Suche nach den Ursachen für Cybersickness stießen wir auf die Arbeit von Joseph J. LaViola Jr. [JJL00]. Darin behandelt er unter Anderem den Aufbau des Auges und des Ohres, da diese beiden Sinnesorgane maßgeblich an den Ursachen beteiligt sind. Das ist gut nachvollziehbar, wenn man bedenkt, dass in etwa 90% unserer Sinneseindrücke über unsere Augen und Ohren in unser Gehirn gelangen, wobei die Augen hier etwa 80% ausmachen. [Goh15] LaViola [JJL00], Simon Davis et al. [ea14b] sowie Hoffmann et al. [SH15] führen in ihren Arbeiten folgende drei Theorien an, auf die wir hier kurz eingehen wollen:

- Die sensorische Konflikt Theorie
- Die Gifttheorie
- Die Theorie zur Haltungsinstabilität

Die sensorische Konflikt-Theorie wurde ursprünglich dazu entworfen, Motion Sickness zu beschreiben. Später wurde sie auch auf Simulator Sickness erweitert. Sie ist die älteste und anerkannteste Theorie im Bezug auf Cybersickness. Man geht dabei davon aus, dass die Symptome daher führen, dass das Gehirn einen Konflikt erkennt zwischen der räumlichen Lage des Körpers und den Sinneseindrücken, die das Gehirn über Augen und Ohren bzw. über den Orientierungssinn erhält. Motion Sickness tritt auf, wenn die Erwartungen, welche die Person hat, nicht mit dem erlebten Ereignissen zusammenpassen. Als Beispiel führt LaViola [JJL00] einen Fahrzeugsimulator an. Wenn sich der Benutzer mit dem

Auto durch die Straßen bewegt, sieht er, wie sich die Gebäude und die anderen Fahrzeuge um ihn herum an ihm vorbei bewegen. Tritt er auf das Gaspedal, wird das Fahrzeug beschleunigen. Der Fahrer kann dies alles mit seinen Augen und Ohren wahrnehmen. Allerdings widerspricht ihm sein Gefühlssinn, denn dieser meldet ihm, dass er keine Beschleunigung wahrnimmt. Das steht im Gegensatz zu dem, was die Person aus ihrem bisherigen Leben gewohnt ist. Aus diesem Grund kann der Person übel werden. Die sensorische Konflikt-Theorie kann jedoch nicht erklären, wieso manchen Personen in bestimmten Situationen schlecht wird und anderen nicht. Auch kann sie nicht vorhersagen, welche Situationen Cybersickness auslösen können und mit welcher Stärke.

Die Gifttheorie sucht die Ursachen für die auftretende Übelkeit in der Evolution des Menschen. Ähnlich wie in virtuellen Simulationen verhalten sich die Sinnesorgane - aus dem Blickwinkel des Gehirns - wenn ein Mensch etwa Gift eingenommen hat: das Wahrgenommene und die Realität stimmen nicht mehr überein und der Körper versucht, sich gegen diesen Umstand zu wehren. Über die Evolution hat sich wohl bewährt, dass Menschen besser überlebt haben, wenn sie in so einem Zustand versucht haben, das geschluckte Gift zu erbrechen. Die Gifttheorie will damit erklären, warum Menschen in einer virtuellen Umgebung übel wird. [JJL00][SH15] Allerdings kann diese Theorie nicht erklären, wieso einigen Personen in einer Situation schlecht wird und anderen wiederum nicht.

Die Theorie zur Haltungsinstabilität geht davon aus, dass der Mensch ständig darauf bedacht ist, eine aufrechte Haltung einzunehmen und diese zu halten. *Dabei is Haltungsstabilität definiert als Zustand, in dem unkontrollierte Bewegungen der Wahrnehmungs- und Handlungssysteme auf ein Minimum reduziert sind.* [SH15] Da man nun ständig diesen Zustand bzw. diese Kontrolle beibehalten will, kann es zu Irritationen führen, wenn man in eine Situation gelangt, in der es zu einem Kontrollverlust kommt, ohne dass man auf vorhandene Gegenstrategien zurückgreifen kann. So ein Kontrollverlust wäre zum Beispiel, wenn man auf einem nassen Boden ausrutscht. Die Folge ist eine sehr schnelle unkontrollierte Körperbewegung, der man nur schwer entgegenwirken kann. Nach einiger Zeit wird man jedoch eine Gegenstrategie entwickeln (z.B. langsam über den nassen Boden gehen), womit der Zustand der Haltungsstabilität wiedererlangt wird. Bei schnellen Bewegungen und Rotationen in einer virtuellen Umgebung ist das allerdings schwer bis gar nicht möglich. Auch Norman G. Vinson et al. [ea12b] konnten in ihrer Studie feststellen, dass die Art der Bewegung und die Kontrolle über diese, sowie die Rotationsgeschwindigkeit einen Einfluss auf Motion Sickness haben. Je länger nun diese Haltungsinstabilität anhält, umso schwerer werden die Symptome von Cybersickness.

Jede dieser drei Theorien versucht auf ihre Art, die Ursachen von Cybersickness zu finden. Wie wir sehen können, haben allerdings alle Theorien Schwierigkeiten, zu erklären, wieso Cybersickness bei bestimmten Personen unter gewissen Situationen auftritt und bei anderen wiederum nicht.

2.4.2 Symptome

Die Symptome von Cybersickness können sehr vielfältig ausfallen. Aus unserer persönlichen Erfahrung können die meisten Personen über mehrere Minuten hinweg in eine virtuelle Welt eintauchen, ohne jegliche Anzeichen von Cybersickness aufzuweisen. Einige wenige jedoch können keine 3D Brille aufsetzen, ohne dass ihnen sofort schlecht wird. Das sind nach unserer Erfahrung allerdings weniger als 1%. Es stellte sich für uns die Frage, womit wir rechnen müssen, wenn wir unsere Simulationen für die Allgemeinheit zur Verfügung stellen wollen. Im Folgenden werden wir einige Symptome anführen, die auftreten können.

Da es sehr viele Symptome geben kann, wollen wir uns hier nur auf diese beschränken, die am Häufigsten auftreten. Simon Davis et al. [ea14b] haben jene Symptome vermerkt, die bei mindestens 20% ihrer Testpersonen aufgetreten sind. Im Augenbereich konnten diese Personen feststellen, dass sie teilweise unter müden, oder geröteten Augen, sowie unter Sehbeschwerden litten. Auch hatten sie Schwierigkeiten, Objekte zu fokussieren, oder hatten eine verschwommene Sicht. Sogar schmerzende Augen kamen mit einer Wahrscheinlichkeit von über 20% vor. Ansonsten fühlten sich einige Personen sehr müde und schläfrig und konnten generelles Unbehagen feststellen. Auch Kopfschmerzen und Probleme beim Konzentrieren wurden vermerkt. Einigen wurde schwindelig, wohingegen sich manche gelangweilt fühlten.

Wie wir sehen, können sich dieselben Erfahrungen bei unterschiedlichen Personen sehr verschieden auswirken. Aus diesem Grund ist es schwer vorherzusagen, wie eine bestimmte Person auf eine virtuelle Welt reagieren wird. Grundsätzlich lassen sich die Auswirkungen in drei Kategorien einteilen. [ea14b]

- Somatische Auswirkungen
- Auswirkungen auf Magen und Darm
- Emotionale Auswirkungen

Somatische Auswirkungen sind in etwa Müdigkeit, körperliche Schwäche, Hitze und Schweißausbrüche, sowie Benommenheit und Unsicherheiten in der Stabilität. Auswirkungen auf Magen und Darm können beispielsweise ein Krankheitsgefühl oder Übelkeit, bis hin zu Brechreiz sein. Im emotionalen Bereich können Nervosität, Verängstigung, Besorgnis, sowie Aufregung oder Panik, bis hin zur Verzweiflung auftreten.

Es ist sehr wichtig, dass Personen darauf hingewiesen werden, dass sie sich in eine virtuelle Umgebung begeben und dass sie wissen, welche Symptome eventuell auftreten können, damit sie sich dessen bewusst sind und während den Simulationen darauf achten können. Glücklicherweise sind die Auswirkungen der virtuellen Erlebnisse aus unserer Erfahrung kaum bis gar nicht erkennbar. Dennoch ist es gut, dafür zu sorgen, dass es in der Nähe eine Sitzmöglichkeit zum Ausruhen gibt und genügend Wasser zur Beruhigung zur Verfügung

steht. Bei längeren Aufenthalten in der virtuellen Welt, werden die Symptome auch stärker ausfallen, als wir das bei unseren eher kurzen Simulationen gewohnt sind.

2.4.3 Praktische Faktoren

Wir konnten nun sehen, welche Theorien hinter Cybersickness stehen und wie sich das Erleben einer virtuellen Welt auf Personen auswirken kann. Um diesen Symptomen entgegenwirken zu können, ist es wichtig zu verstehen, aus welchen praktischen Gründen nun diese unerwünschten Effekte entstehen und wie wir ihnen entgegenwirken können. Es gibt eine Reihe von Kriterien neben den drei erwähnten Theorien, die für das Entstehen von Cybersickness verantwortlich sind. In diesem Kapitel werden wir einige dieser Faktoren aufzählen und kurz auf sie eingehen.

Zum einen spielt die verwendete Technologie eine entscheidende Rolle. LaViola [JJL00] weist in diesem Zusammenhang speziell auf das verwendete Display hin. In den vergangenen Jahren konnte glücklicherweise hier ein großer Sprung nach vorne gemacht werden. Viele VR-Brillen, wie in etwa die Oculus Rift, konnten die Auflösung und das Nachverfolgen der Kopfbewegungen wesentlich verbessern. Dennoch gibt es hier noch sehr viel Spielraum für Verbesserungen. Wenn man eine Oculus Rift aufsetzt erkennt man immer noch deutlich, dass das Bild aus einzelnen Pixeln besteht, die man durchaus mit dem Auge erkennen kann. Des Weiteren lässt es sich schwer vermeiden, dass das erzeugte Bild stabil ist. So gibt es durch die verwendete Technologie immer ein kleines Flackern, was auch als Jitter bezeichnet wird. Das kommt daher, dass die Position der Brille nicht immer exakt bestimmt werden kann und es zu Rundungsfehlern bei der Messung kommt. Ein weiteres Problem besteht in der Latenz. Das ist die Zeit, die zwischen der Bewegung des Kopfes und der tatsächlichen Darstellung des neuen Bildes auf dem Display vergeht. Wird das mit einer aufwändigen Szene und einer niedrigen Bildwiederholungsfrequenz kombiniert, kann das zu sehr schlechter Performance führen, bei der das Erleben der virtuellen Realität zu einer Qual wird. Des Weiteren ist eine falsche Kalibrierung der Brille sehr schlecht. Auch Brillenträger haben Nachteile, weil sie das dargestellte Bild nicht scharf erkennen können. Hier schafft Oculus Rift beispielsweise mit einem zweiten Linsensatz für die Brille Abhilfe.

Zusätzlich zur Technologie führen LaViola [JJL00], sowie Simon Davis et al. [ea14b] individuelle Faktoren an. Sie verweisen hier beispielsweise auf das Alter der Person. Man würde zwar vermuten, dass vor allem alte Personen mehr Probleme mit Cybersickness hätten, dem ist jedoch nicht so. Die größten Schwierigkeiten haben Kinder im Alter zwischen 2 und 12 Jahren. Danach sinkt die Anfälligkeit sehr stark und ab einem Alter von etwa 50 Jahren gebe es anscheinend keine Anzeichen mehr für Cybersickness. Auch das Geschlecht ist ein Faktor in diesem Bereich. So haben Frauen ein größeres Sichtfeld, wodurch sie eher ein unerwünschtes Flackern in den Augenwinkeln wahrnehmen als Männer. Des Weiteren spielt der Gesundheitszustand eine Rolle. Beispielsweise sind Ermüdung, Krankheit, sowie Alkoholeinfluss entscheidende Faktoren. Das erklärt auch, warum manche Personen an gewissen Tagen keine Anzeichen von Cybersickness aufweisen, jedoch an anderen Tagen deutlich beeinträchtigt wirken. Auch die Position und die

Körperhaltung während der Simulation ist ein großes Thema. So konnte festgestellt werden, dass eine sitzende Körperhaltung am Besten zur Prävention von Cybersickness geeignet ist. Dies steht in Übereinstimmung mit der Theorie zur Haltungsinstabilität, da es dem Benutzer leichter fällt, seine aktuelle Position zu halten.

Weitere Faktoren laut Simon Davis et al. [ea14b] finden sich im Bereich der Aufgabenstellung und der Kontrolle. Kann der Benutzer sehr viel selbst steuern, wie beispielsweise die Bewegungsrichtung, Rotationen und das Blickfeld, ist es eher unwahrscheinlich, dass Cybersickness auftritt. Können zukünftige Ereignisse gut eingeschätzt werden, sodass sich die Person darauf vorbereiten kann, trägt das ebenso positiv zum Erlebnis bei. Problematisch hingegen sind Simulationen, in denen wenig bis gar keine Kontrolle besteht und sehr schnelle Bewegungen und Richtungswechsel stattfinden. [ea12b][SH15] Dieses Phänomen ist vergleichbar zum Autofahren in der Realität. Personen, die als Beifahrer im Fahrzeug sitzen, wird eher schlecht, als wenn sie selber das Fahrzeug steuern können.

Es ist wichtig, dass wir uns bewusst sind, welche Faktoren uns beeinflussen, wenn wir eine virtuelle Simulation erstellen und Personen in diese eintauchen lassen. Einige Faktoren, wie zum Beispiel die Technologie, lassen sich nur schwer beseitigen. Allerdings können wir in dieser Hinsicht davon ausgehen, dass sich in Zukunft hier viel verbessern wird und wir mehr Möglichkeiten haben, erstaunliche Erlebnisse erstellen zu können. Individuelle Faktoren können wir großteils leider kaum beeinflussen, wir können jedoch auf unseren Gesundheitszustand achten und virtuelle Umgebungen meiden, wenn wir uns nicht wohl fühlen. In unseren Simulationen können wir dafür Sorge tragen, dass keine raschen Bewegungen und Rotationen vorkommen, um die Benutzer zu schonen.

2.4.4 Konsequenzen

Mit unserem Wissen um die drei Theorien um Cybersickness und die Faktoren, die diese unerwünschten Symptome auslösen können, ist es uns nun möglich, nach Gegenmaßnahmen zu suchen und den Benutzern ein möglichst positives Erlebnis zu schaffen. LaViola [JJL00] führt vier mögliche Herangehensweisen an:

- Bewegliche Plattformen
- Direkte Stimulation des Gleichgewichtsorgans
- Ruhesysteme
- Anpassung

Bewegliche Plattformen sollen entsprechend der Sensorischen-Konflikt-Theorie das Gleichgewichtsorgan beeinflussen, damit das Gefühlte mit dem Gesehenen übereinstimmt. Das heißt, fährt der Benutzer mit einem virtuellen Fahrzeug in eine Kurve, so muss sich auch der Sessel, auf dem er sitzt so verhalten, dass sein Körper tatsächlich das Gefühl hat,

in eine Kurve zu fahren. Die Firma CKAS[15] hat sich beispielsweise auf die Erzeugung solcher Plattformen spezialisiert.

Eine ähnliche Herangehensweise bietet die direkte Stimulation des Gleichgewichtsorgans. Hier wird versucht, den Gleichgewichtssinn über elektrische Impulse zu stimulieren und so eine räumliche Bewegung zu simulieren. In diese Richtung wird allerdings im Bereich von Cybersickness noch sehr wenig geforscht. Es ist auch sehr schwierig, den Gleichgewichtssinn genau so zu stimulieren, dass das Wahrgenommene mit der virtuellen Szene übereinstimmt.

Das Konzept von Ruhesystemen kommt daher, dass der Mensch alle paar Momente nach Punkten in seiner Umgebung sucht, die in Ruhe sind, um sich neu orientieren zu können. Fällt es den Personen schwer, solche Punkte in der virtuellen Umgebung zu finden, kann es leicht passieren, dass es zu Symptomen von Cybersickness kommt.

Die wohl geeignetste Methode, um gegen Cybersickness vorzugehen besteht darin, die Benutzer langsam an die virtuelle Umgebung gewöhnen zu lassen. So sollte es vermieden werden, direkt zu Beginn einer Simulation schnelle Bewegungen und Rotationen durchzuführen. Auch die Dauer des Umgangs mit der virtuellen Welt sollte anfangs eher kurz gehalten und mit häufigerem Benutzen des Systems langsam erhöht werden. LaViola [JJL00] schreibt, dass dadurch zwar trotzdem noch Probleme nach dem Benutzen von VR-Anwendungen auftreten können. Allerdings sei die Methode der Anpassung die Beste, sofern die Benutzer die Zeit haben, langsam durch diesen Prozess zu gehen.

Bewegliche Plattformen, wie sie LaViola [JJL00] beschreibt, sind in unseren Simulationen nicht vorhanden, allerdings wird das Gleichgewichtsorgan des Benutzers richtig beeinflusst, da die Bewegungen des Nutzers in der Realität mit den virtuellen Bewegungen nahezu übereinstimmen. Um einen stärkeren Effekt zu erzielen, würde sich anbieten, das Gleichgewichtsorgan direkt zu über eine Diode zu stimulieren. Das kommt jedoch aus praktischen Gründen nicht in Frage, da das Kabel der Diode den Benutzer irritieren könnte. Ruhesysteme hingegen lassen sich leichter anlegen und wir müssen in unseren Simulationen darauf achten, genügend solche Ruhepunkte einzufügen, um dem Benutzer zu jeder Zeit einige Anhaltspunkte geben zu können. Auch das Konzept der Anpassung scheint logisch und sinnvoll zu sein. Jedoch können wir uns voraussichtlich nicht immer die Zeit nehmen, Benutzer unserer Simulationen an das System heranzuführen. Wir können allerdings darauf acht geben, keine hohen Geschwindigkeiten und schnelle Rotationen in unseren Szenen einzubauen.

2.5 Bezier-/Spline-Kurven-Algorithmen

Für die Erzeugung des Kamerapfades in unserer Simulation wollten wir keine geraden Linien verwenden, da dadurch an den Wegpunkten sehr harte Kanten auftreten können, was die Personen, die unsere Simulationen erleben, irritieren könnte. Deshalb kamen wir

[15] `http://www.ckas.com.au/Home_1.html` (Zugriff am 26.08.2015)

bald auf die Idee, für diesen Teil unseres Systems auf Bezierkurven (oder bezierkurven-ähnliche Funktionen) zurückzugreifen. Kurvenfunktionen bringen den Vorteil, dass der Kamerapfad geglättet verläuft und auch schnelle Rotationen vermieden werden, was wiederum zur Vorbeugung von Cybersickness dienlich ist. Ebenso können so die Benutzer leichter vorhersagen, was auf sie zukommen wird, da sie sich bereits frühzeitig auf den nächsten Wegpunkt einstellen können. Aus diesem Grunde sahen wir uns dazu gezwungen, einige Nachforschungen über verschiedene Kurven-Algorithmen zu machen. Die entscheidenden Teile unserer Recherche werden wir im Folgenden präsentieren.

Um die unterschiedlichen Algorithmen verstehen zu können, ist es zunächst wichtig, zu wissen, wie Bezierkurven gebildet werden. Mike Kamermans [Kam15] erklärt dies in seiner Ausarbeitung auf einfache Weise: *Bezier curves are the result of linear interpolations.* [Kam15] Lineare Interpolation bedeutet, dass man einen Punkt zwischen zwei Punkten auswählt. Im Allgemeinen kann dieser Punkt noch zusätzliche Eigenschaften haben. Wenn man zum Beispiel den beiden äußeren Punkten einen Farbwert zuweist, würde ein bestimmter Punkt dazwischen einen bestimmten Farbwert zwischen den beiden anderen Punkten besitzen. Häufig wird ein Punkt, der auf der Strecke von A nach B liegt als Prozentwert angegeben. So könnte man einen Punkt D wählen, sodass dieser bei 20% auf der Strecke AB liegt. Er hätte 20% der Strecke Abstand zu Punkt A und 80% Abstand zu Punkt B. Bezierkurven werden grundsätzlich über eine größere Anzahl an Punkten beschrieben. Fügen wir zwischen A und B einen Punkt C hinzu, können wir dadurch eine Bezierkurve interpolieren. Dies funktioniert so, indem wir zunächst beispielsweise die 20% Punkte von AC und CB suchen und eine Linie zwischen diesen Punkten bilden. Auf dieser Linie suchen wir wiederum den 20% Wert und erhalten so einen Punkt, der auf der Bezierkurve liegt, wie die Skizze in Abbildung 2.16 verdeutlichen soll. Durch Wiederholung dieser Methode erhalten wir alle Punkte auf der Bezierkurve.

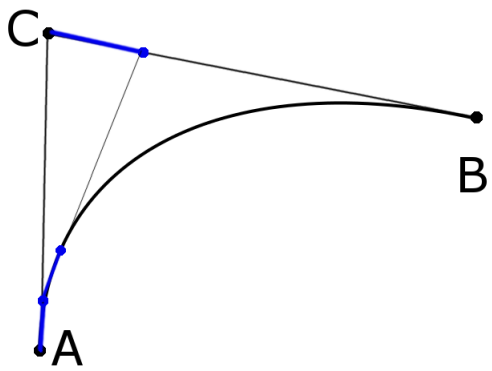

Abbildung 2.16: Über lineare Interpolation können alle Punkte auf einer Bezierkurve bestimmt werden. Diese Skizze stellt die Berechnung eines Punktes der Bezierkurve dar.

Am Computer werden Bezierkurven jedoch selten über Interpolationen ausgerechnet, sondern eher über Polynomfunktionen definiert. Hier wird anstatt eines Prozentwertes

für eine Interpolation meist eine Variable t verwendet, die zwischen 0 und 1 liegen kann. Mit dem Wert 0 erhält man den Anfangspunkt der Kurve, mit 1 den Endpunkt. Des Weiteren lassen sich Bezierkurven über Matrixoperationen berechnen. Wir wollen hier jedoch nicht näher darauf eingehen, da dieses Thema den Umfang dieser Diplomarbeit sprengen würde.

Wichtig zu wissen ist, dass Bezierkurven über so genannte Kontrollpunkte definiert werden (wie oben erwähnt beispielsweise mit den Punkten A, B und C). Diese Punkte wirken auf den Kurvenverlauf der Bezierkurve wie ein Schwerkraftsystem, das auf ein Objekt wirkt. Man kann sich das in etwa so vorstellen, als ob die Punkte Planeten sind, die auf einen vorbeiziehenden Kometen einwirken und so seine Richtung beeinflussen. Allerdings ist bei Bezierkurven nicht garantiert, dass der Kurvenverlauf durch die Kontrollpunkte läuft. Lediglich der Anfangs- und Endpunkt liegt immer auf der Kurve. Außerdem haben Bezierkurven den Nachteil, dass sich die Veränderung eines Kontrollpunktes auf die gesamte Kurve drastisch auswirken kann. Wenn man bedenkt, dass Kurven gerne für die Darstellung von Schriftzeichen, Autoformen und Animationspfaden [Tul15] verwendet werden, ist es verständlicherweise sehr unpraktisch, wenn sich die ganze Kurve ändert, wenn man nur einen Punkt verändern will. Aus diesem Grund suchte man nach Methoden, eine Kurve als Teilstücke (so genannte Splines) zu beschreiben. Eine Kurve wird somit auf Fragmente aufgeteilt, die kontrollierbar verändert werden können, sodass sich die Veränderung nur lokal auswirkt und nicht die gesamte Kurve beeinflusst. Mittels Casteljau's Algorithmus [Kam15] kann dies beispielsweise bewerkstelligt werden.

John Lowther und Ching-Kuang Shene [JL03] beschreiben in ihrer Arbeit, wie ihre Herangehensweise ist, um ihren Studierenden Bezierkurven und Bezier-Splines näher zu bringen. Auch zeigen sie, wie eine Bezierkurve in B-Splines umgewandelt werden kann und welche Auswirkungen das auf die Kontrollpunkte hat. In Abbildung 2.17 können wir sehen, dass für die Bezier-Splines wesentlich mehr Kontrollpunkte benötigt werden. Der große Vorteil davon - wie bereits erwähnt - ist allerdings, dass Veränderungen von Kontrollpunkten nur noch lokal, das heißt, auf die beiden angrenzenden Linienfragmente, wirken. Dies können wir in Abbildung 2.18 sehen.

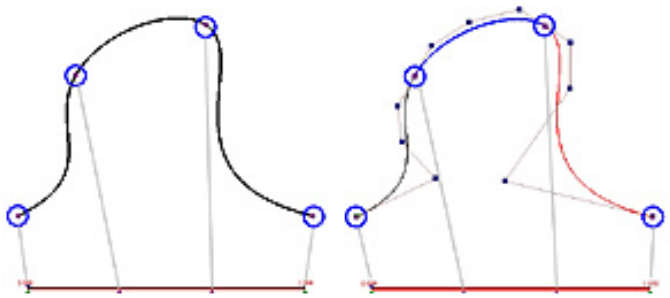

Abbildung 2.17: Wir sehen links eine Bezier-Kurve und rechts die dazugehörigen Bezier-Splines, die dieselbe Kurve aufspannen. [JL03]

Abbildung 2.18: Positionsveränderungen eines Kontrollpunktes wirken bei Bezier-Splines nur lokal und haben keinen Einfluss auf weiter entfernte Liniensegmente.[JL03]

Abgesehen von Bezier-Splines gibt es noch weitere Spline-Algorithmen, welche Ähnlichkeiten mit B-Splines aufweisen. Der Algorithmus von Edwin Catmull und Raphael Rom basiert ebenfalls auf Splines, jedoch mit dem Unterschied, dass die Kurve durch alle Kontrollpunkte verläuft (mit Ausnahme des Start- und Endpunktes). [Kam15] Daher ist der Catmull-Rom-Algorithmus sehr gut dafür geeignet, Pfade und Formen zu designen, wenn man vorher bereits genau weiß, durch welche Punkte die Kurve verlaufen muss. Zusätzlich ist es möglich, die Parameter der Catmull-Rom-Splines zu verändern und so mehr Kontrolle über die Kurve zu erhalten. Cem Yuksel et al. haben in diese Richtung geforscht und ihre Ergebnisse veröffentlicht. [ea09] Auch sie geben als Vorteile von Catmull-Rom-Kurven an, dass die Kurve durch die Kontrollpunkte verläuft und sich Änderungen an den Kontrollpunkten nur lokal auswirken. Des Weiteren können die Kurventeile über Polynomfunktionen repräsentiert werden, was für eine schnelle Verarbeitung auch bei geringer Rechenleistung sorgt. Wählt man eine geeignete Parametrisierung für den Kurvenalgorithmus, erhält man eine Kurve, die keine Schlaufen oder Überschneidungen mit sich selbst bildet (zu sehen auf Abbildung 2.19).

Die Tatsache, dass Catmull-Rom-Kurven durch ihre vordefinierten Kontrollpunkte verlaufen und Änderungen nur lokal wirken, hat uns dazu veranlasst, nach bereits bestehende Lösungen in Unity3D zu suchen. Erik Nordeus beschreibt in seinem kurzen Tutorial [Nor15] eine Möglichkeit, wie ein solches System im Unity-Editor verwirklicht werden kann. Auf ähnliche Art und Weise konnten wir unser System in Unity verwirklichen, wie wir im Kapitel Systembeschreibung genauer erläutern.

In diesem Kapitel konnten wir sehen, wie Bezierkurven aufgebaut sind und wie sie berechnet werden können. Wir haben erklärt, wie das Konzept der Kontrollpunkte funktioniert und dass Bezierkurven in Bezier-Splines umgewandelt werden können. Dadurch wird eine Kurve in kleinere Fragmente zerlegt, die durch eine größere Anzahl an Kontrollpunkten definiert wird. Das hat den Vorteil, dass sich Änderungen an einem Kontrollpunkt nicht global, sondern lokal auswirken und man daher als Designer mehr Kontrolle erhält.

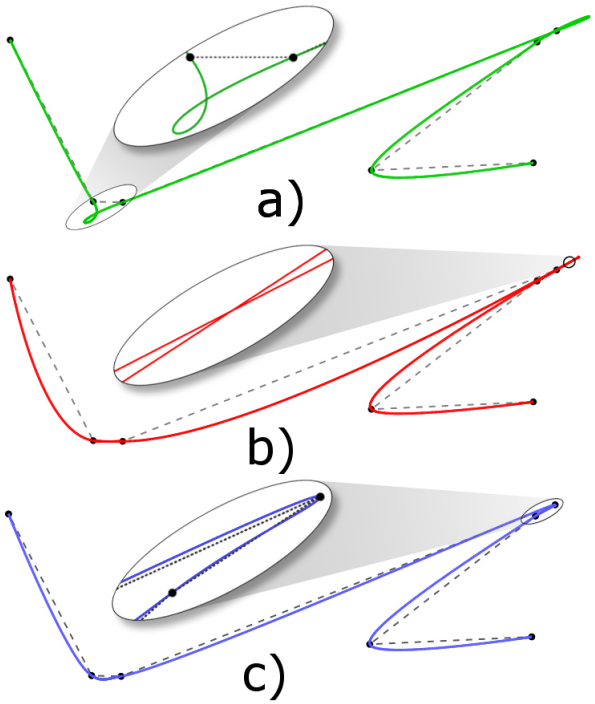

Abbildung 2.19: Die Uniform-Parametrisierung (a) neigt dazu, Schlaufen zu bilden. Chordal-Parametrisierung (b) kann Überschneidungen der Kurve mit sich selbst verursachen. Durch die Centripetal-Parametrisierung (c) entstehen keine Schlaufen oder Überschneidungen. [ea09]

Schließlich konnten wir über die Funktionsweise von Catmull-Rom-Kurven diskutieren und haben gemerkt, dass diese Kurven durch ihre Kontrollpunkte verlaufen. Das macht sie sehr nützlich, wenn man vorher bereits weiß, durch welche Punkte die Kurve auf jeden Fall durchlaufen muss. Auch gibt es die Möglichkeit, eine bestimmte Parametrisierung zu wählen, sodass der Catmull-Rom-Algorithmus keine Schlaufen und Überschneidungen bildet.

KAPITEL 3

Projektdefinition

In diesem Kapitel diskutieren wir den Entwicklungsprozess des Content-Generators. Zunächst beleuchten wir die Gründe und zeigen, dass die Entstehung des Content-Generators aus einem vorherigen Projekt [Eid15a] hervorging. In diesem Zusammenhang werden unsere bisherigen Erfahrungen mit der Entwicklungsumgebung Unity3D aufgezeigt und unsere Fehler und Erkenntnisse im Umgang mit dem Unity-Editor beschrieben. Zusätzlich wird darauf eingegangen, welche Probleme uns dazu bewegten, ein eigenes Werkzeug zu entwickeln, um zukünftig effektiver an unseren Simulationen arbeiten zu können. Im Unterkapitel 3.2 wird beschrieben, welche Funktionen der Content-Generator besitzen muss, um einen effizienten Arbeitsablauf ermöglichen zu können und um möglichst viele Anwendungsfälle abdecken zu können. Schließlich wird im Kapitel 3.3 die Vorgehensweise und der zeitliche Ablauf bei der Entwicklung des Werkzeugs beschrieben und auf welchem Wissen unsere verwendeten Methoden beruhen.

3.1 Hintergrund

Um zu verstehen, wie es zu der Entwicklung des Content-Generator-Werkzeugs gekommen ist, muss zunächst erklärt werden, woher die Notwendigkeit dafür stammt. Im September 2014 wurde uns ein Projekt [Eid15a] zur Bearbeitung angeboten. Das Ziel dieses Projekts war es, einen Fallschirmsprung-Simulator zu erstellen, der es ermöglichen sollte, Benutzer in eine virtuelle Umgebung eintauchen zu lassen, sodass sie über der Stadt Wien aus einem virtuellen Flugzeug springen können. Da diese Simulation im Rahmen von TU200[1] entstand, musste dafür gesorgt werden, dass neben dem virtuellen Sprung und der Umgebung dem Springer zusätzlich Informationen über die vergangenen 200 Jahre der Technischen Universität Wien zugänglich gemacht werden. Dies sollte während des mehrminütigen Sprunges über so genannte Objektträger/Informationsobjekte geschehen. Das heißt, die Kernziele des Sprungsimulators waren unter Anderem:

[1] `http://www.tu200.at/` (Zugriff am 29.08.2015)

- Die virtuelle Umgebung erzeugen (Stadtmodell, Flugzeuge, Vögel, Wolken, usw.)
- Logik
  - Erzeugung eines Bewegungspfades des Springers
  - Informationsträger erstellen, welche Bilder, Text und Ton speichern und wiedergeben können
  - Spezialeffekte in der virtuellen Umgebung
- Verknüpfung des Projektes mit zusätzlichen Ein- und Ausgabegeräten, wie etwa Oculus Rift, Kinect, sowie einem Raspberry[2]

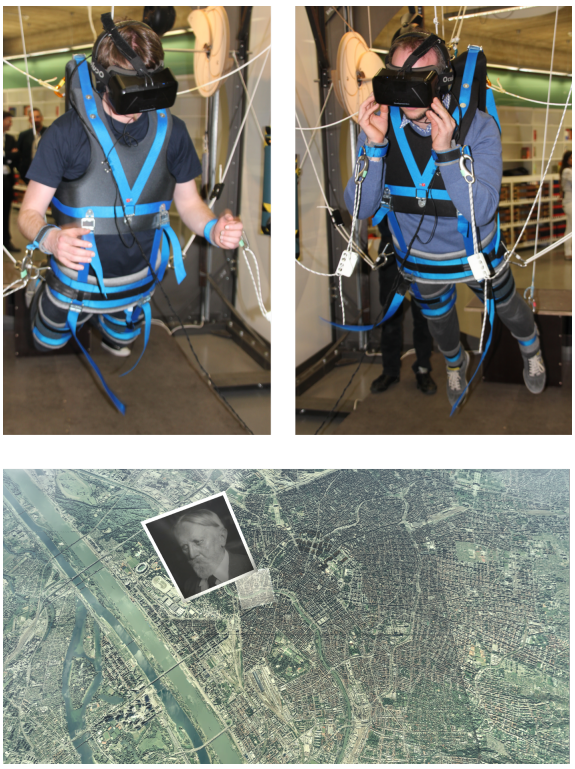

Abbildung 3.1: Während die Benutzer des Fallschirmsprung-Simulators aus etwa 80 cm Höhe nach vorne fallen, springen sie in der Simulation aus einem Flugzeug über Wien heraus und absolvieren so einen virtuellen Fallschirmsprung. Während des Sprunges erhalten die Benutzer über Objekte in der virtuellen Umgebung Informationen über die vergangenen 200 Jahre der Technischen Universität Wien.

Um das Spielerobjekt vom Flugzeug aus auf die Stadt fallen zu lassen bietet Unity eine Physik-Engine an, mit der Schwerkraft simuliert werden kann. Allerdings haben wir

[2] https://www.raspberrypi.org/

darauf verzichtet, da uns ein freier Fall zu viel Kontrolle über die Simulation nehmen würde. Aus diesem Grund haben wir in unserem Prototypen ein einfaches Wegpunktsystem integriert, das den Spieler auf geraden Linien durch die dreidimensionale virtuelle Welt führt. Der erste Wegpunkt lag außerhalb des Flugzeuges, sodass sich der Spieler zunächst aus dem Flugzeug heraus bewegt. Der nächste Punkt lag bereits ein paar hundert Meter über der Stadt, wo sich auch der Fallschirm öffnet, um den Spieler später über die Stadt gleiten zu lassen. Zwischen diesen beiden Wegpunkten sollte nun der multimediale Informationsinhalt in Form von Bilder, Texten und Sprache wiedergegeben werden. Zu diesem Zweck wurden Objekte erstellt, die sich nahe dem Bewegungspfad des Spielers befinden und sich aktivieren, sobald der Spieler nahe genug an sie heran kommt. Zu Beginn waren das sieben Objekte, die jeweils etwa 30 Sekunden lang den Spieler begleiteten und ihm währenddessen unterschiedliche Bilder präsentierten, während im Hintergrund Musik gespielt und Texte gesprochen wurden. Nachdem sich der Spieler in einem dreidimensionalen Raum bewegte, war es sehr schwierig, diese Informationsobjekte so in der virtuellen Welt zu platzieren, dass sie erstens vom Spieler aktiviert wurden und zweitens sich nicht mit den anderen Informationsobjekten bei der Wiedergabe überschnitten. Unter der Annahme, dass wir diese Objekte einmalig platzieren müssen und sie für den Rest des Entwicklungsprozesses an diesen Punkten bleiben können, machten wir die Platzierung vorerst manuell. Dies kostete allerdings viel Zeit, da hier mehrere Faktoren eine Rolle spielen: Die Geschwindigkeit des Spielers, die Länge der einzelnen Informationsinhalte, sowie der Bewegungspfad. Unsere Annahme, dass die Informationsobjekte an ihren Plätzen bis zum Schluss des Projektes bleiben würden, war leider falsch und dadurch verloren wir jedes Mal, wenn wir Änderungen machten, abermals viel Zeit. Zu einem späteren Zeitpunkt wollten wir aus Gründen eines besseren Simulations-Erlebnisses aus den 7 Informationsobjekten (welche jeweils mehrere Bilder anzeigten) 42 Objekte (die nur ein Bild anzeigen) machen. Zu diesem Zeitpunkt war klar, dass wir diese Objekte nicht mehr manuell platzieren konnten, ohne einen erheblichen Zeitverlust hinnehmen zu müssen. Da wir zu diesem Zeitpunkt bereits Ideen für weitere, ähnliche Simulationen hatten, wurde uns bewusst, dass wir für die Erstellung von solchen Simulationen ein besseres Werkzeug in Unity3D benötigen würden. Mit einigen zusätzlichen Gedanken und Verbesserungsvorschlägen kamen wir auf eine Anforderungsliste für einen Content Generator, die wir im folgenden Abschnitt diskutieren.

3.2 Anforderungen

Die Anforderungen für den Content Generator ergaben sich wie bereits beschrieben zu einem großen Teil aus dem vorherigen Projekt. [Eid15a] Des Weiteren hatten andere uns bekannte Simulationen, wie etwa Museumssimulationen, Einfluss auf unsere Anforderungsliste. Es sollte ein Werkzeug erstellt werden, mit dem es möglich sein muss, in einer Game Engine zumindest folgende Tätigkeiten effektiv verrichten zu können:

- Mithilfe eines Wegpunktsystems muss ein Pfad erstellt werden können, der auf Bezier- oder Spline-Kurven basiert. Da ein geglätteter Pfad vor allem den Vorteil

hat, dass sich der Benutzer frühzeitig darauf einstellen kann, in welche Richtung er sich als nächstes bewegen wird. Dadurch können die Effekte von Cybersickness reduziert und das Benutzererlebnis verbessert werden.

- Dreidimensionale Objekte müssen als Informationsträger dienen können. Dazu ist es erforderlich, dass diese Objekte mit Bildern, Texten und Ton ausgestattet werden, die sie zu einem geeigneten Zeitpunkt von sich geben können. Die Präsentationsfläche der Bilder und der Texte muss dazu eigens definiert werden können, damit das Werkzeug für eine Reihe von Anwendungen sinnvoll ist.
- Es muss die Möglichkeit vorhanden sein, die Gesamtdauer der Simulation und des multimedialen Inhalts zeitlich zu begrenzen. Das dient vor allem dem Zweck, dass Personen nicht unnötig lange darauf warten müssen, dass vorherige Benutzer die Simulation abschließen.
- Die Informationsträger müssen so auf dem vordefinierten Pfad platziert werden können, dass es zu keinen Überschneidungen während der Simulation kommt. Das heißt, solange ein Informationsträger noch aktiv ist, soll ihn kein anderer unterbrechen.
- Für die Informationsträger soll eine Reihe von Verhaltensweisen zur Verfügung stehen, um sie individuell an die Erfordernisse der jeweiligen Simulation anpassen zu können. Dies muss für jeden einzelnen Informationsträger innerhalb einer Simulation möglich sein und die Verhaltensweisen müssen sich zwischen den Informationsträgern unterscheiden können.
- Das System muss die Möglichkeit bieten, ein HMD auf einfache Art und Weise integrieren zu können, um dem Benutzer ein dreidimensionales Erlebnis zu bieten.

Wir gehen davon aus, dass mit einem Werkzeug, das diese Anforderungen erfüllt, eine große Anzahl an verschiedensten Simulationen einfach und schnell erstellt werden kann. Neben dem bereits erstellten Fallschirmsprung-Simulator, sollten auch weitere Anwendungen, wie etwa Flüge im Weltall, Tauchgänge in Ozeanen, Achterbahnfahrten, Flugzeugflüge und Heißluftballonfahrten, sowie Museumsrundgänge und Besichtigungen von Gebäuden und vieles mehr auf einfache Art und Weise realisiert werden können. Nach der Literaturrecherche, und mit diesen verschiedenen Simulationsarten im Hinterkopf, starteten wir schließlich den Entwicklungsprozess, über den wir im Folgenden berichten.

3.3 Entstehungsprozess

In diesem Kapitel wird darauf eingegangen, wie die Herangehensweise an die Entwicklung des Content-Generators war. Es wird diskutiert, was die ursprünglichen Probleme in unserem Prototypen waren, und welche Lösungsansätze dafür gefunden wurden und woher diese stammen. Des Weiteren geben wir einen Überblick über die verschiedenen Features, die im Laufe der Entwicklung implementiert wurden. Genauer wird hier darauf

eingegangen, in welcher Reihenfolge wir die jeweiligen Punkte der Anforderungsliste integriert haben und wo Schwierigkeiten aufgetaucht sind. Des Weiteren wird über unsere Erfahrungen sowie unseren Lernprozess berichtet und auf Punkte hingewiesen, die verbessert werden können.

Bevor mit der Entwicklung gestartet werden konnte, war es sehr wichtig, den Prototypen im Virtual Jump Simulator [Eid15a] zu evaluieren und zu erkennen, wo genau die größten Probleme lagen (das HMD war bereits im Unity-Prototypen vollständig integriert). Informationsobjekte wurden im ursprünglichen Prototypen noch manuell platziert. Das sorgte für einige Probleme, die wir unbedingt lösen mussten, um nicht bei kleinen Änderungen viel Zeit zu verlieren. Sowohl Veränderungen am Pfad, als auch an den Informationsinhalten oder der Spieler-Geschwindigkeit haben Einfluss auf die Position der Objekte. Deshalb hatte die Positionierung der Informationsobjekte für uns eine sehr hohe Priorität. Des Weiteren musste eine effiziente Möglichkeit gefunden werden, um die vielen Informationsobjekte mit multimedialem Inhalt zu füllen, damit die Fallschirmsprung-Simulation effektiver verwaltet werden konnte. Da die neuen Funktionen des Content-Generators oftmals sofort in den Sprung-Simulator integriert worden waren, stand als nächster Punkt die Implementierung eines Systems an der Reihe, mit dem die Informationsobjekte mit Verhaltensmuster bestückt werden konnten. Die Erzeugung eines abgerundeten Pfades war zu diesem Zeitpunkt von geringerer Bedeutung, da diese Simulation nur wenige Wegpunkte für den Spieler beinhaltete. Wir werden den gesamten Entwicklungsprozess im Folgenden genauer beschreiben.

Das ursprüngliche Projekt startete im Oktober 2014 mit dem Ziel, auf einem dreidimensionalem Pfad Objekte zu platzieren, die an den Benutzer Informationen abgeben können. Zunächst waren dies 7 Objekte, die manuell in der Umgebung platziert worden waren. Es wurde viel Zeit dafür verwendet, diese Objekte passend zu positionieren, sodass es zu keinen Überschneidungen kommt, wenn sich der Benutzer daran vorbei bewegt. Im Februar 2015 wurde die Anzahl der Informationsträger in der virtuellen Szene des Prototypen von 7 auf 42 erhöht. Damit war klar, dass diese nicht mehr manuell platziert werden konnten, ohne einen großen Zeitverlust in Kauf zu nehmen. Daher musste ein so genanntes Manager-Objekt entwickelt werden, das die Aufgabe hatte, die Informationsobjekte passend in der Szene zu platzieren. Dazu wurde zu Beginn ein Algorithmus entwickelt, der diese Objekte entlang einer geraden Linie in der dreidimensionalen Umgebung mit einer konstanten Entfernung voneinander getrennt positioniert. Für unsere ursprünglichen Zwecke war dies bereits nahezu ausreichend. Auch der multimediale Inhalt konnte mithilfe des Manager-Objektes in den Objekten gespeichert werden, welche auch schon mit einer einfachen Verhaltensweise bestückt waren (zu Beginn der Simulation waren sie unsichtbar und wurden langsam sichtbar, wenn sich der Spieler annäherte). Allerdings unterschieden sich die Informationsobjekte dahingehend, dass einige etwas mehr Zeit verbrauchten als andere. Das kommt daher, dass der dazugehörige Informationsinhalt in Form von gesprochener Sprache länger dauerte. Dadurch kam es zu Überschneidungen des multimedialen Inhalts. Daher wurde der Algorithmus so umgeformt, dass er diesen Zeitfaktor auch mit einbezieht. Später wurde auch noch die Geschwindigkeit des Spielers

als Faktor mitberechnet, da diese noch nicht final im Simulator festgelegt war. Nach einigen weiteren Änderungen und im Hinblick darauf, dass viele ähnliche Simulationen noch erstellt werden sollten, wurde klar, dass dieser einfache Algorithmus, der die Informationsobjekte auf einer geraden Linie positioniert, nicht ausreichen würde. Daher wurde der Algorithmus und das Manager-Objekt ab Ende April 2015 nochmal komplett überarbeitet und sollte von nun an den Namen Content-Generator tragen.

Anfang Mai 2015 wurde der Positionierungs-Algorithmus soweit verbessert, dass er die Objekte entlang eines vordefinierten Wegpunktsystems platziert. Dabei wird aus der Wegstrecke und der Gesamtzeit des multimedialen Inhaltes die maximale Geschwindigkeit des Spielers berechnet, welche wiederum dazu verwendet wird, die Informationsobjekte in einem ausreichenden Abstand voneinander zu platzieren. Es war nun erstmals möglich, die Objekte auch auf einem Zick-Zack-Kurs zu platzieren, ohne dass es beim Durchlaufen der Simulation zu Überschneidungen des multimedialen Inhaltes kam. Um einen besseren Überblick über den Pfad zu haben, wurde im Unity-Editor auf das Gizmo-Werkzeug zurückgegriffen, mit dessen Hilfe in der Szenen-Ansicht grafische Objekte, wie etwa Linien und Sphären eingezeichnet werden können (siehe Abbildung 3.2)

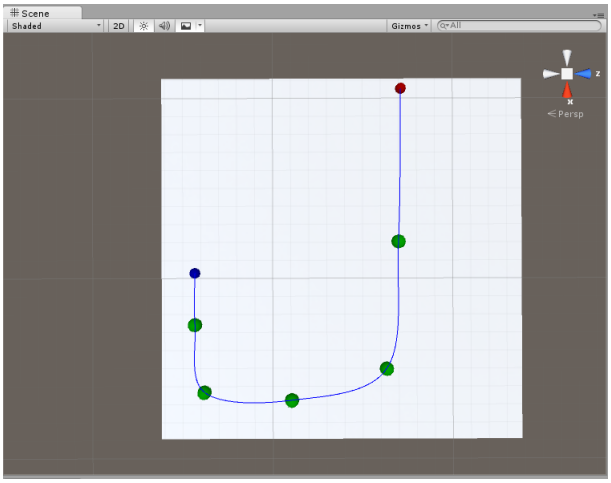

Abbildung 3.2: Der Pfad des Wegpunktsystems wird mittels Linien und Sphären visualisiert.

Ende Mai wurde das Pfadsystem insofern erweitert, als dass nun anstatt gerader Linien Catmull-Rom-Splines verwendet wurden. Diese haben - wie im Abschnitt 2.5 erwähnt - den Vorteil, dass sie einen abgerundeten Verlauf haben, Veränderungen an Kontrollpunkten nur lokalen Einfluss haben und der erstellte Pfad immer durch die Kontrollpunkte verläuft. Der Algorithmus der Objektpositionierung konnte relativ einfach an das neue Pfadsystem angepasst werden. Danach wurde versucht, den Arbeitsablauf zu vereinfachen, indem der Entwickler den Pfad rein über die Szenen-Ansicht verwalten kann, ohne das Hierarchie-Fenster beachten zu müssen. Dies sollte realisiert werden, indem der Pfad durch die Manipulation der Gizmo-Sphären verändert wird. Leider war dies nicht von Erfolg gekrönt

und so ist es weiterhin nötig, die Wegpunkte zuerst im Hierarchiefenster auszuwählen.

Schließlich wurden Verhaltensmuster für die Informationsobjekte erstellt, sowie der Content-Generator als Werkzeug für den Entwickler und den Simulations-Designer verbessert. Neben Informationstexten im Unity-Editor und weiteren Funktionen für die Pfaderstellung und die Positionierung der Informationsobjekte wurde auch eine Funktion bereitgestellt, mit deren Hilfe die Bilder, Texte und Audio-Dateien nicht mehr einzeln in den Content-Generator eingegeben werden müssen, sondern über eine Text-Datei definiert werden können. Das bietet einerseits den Vorteil, dass sich der Entwickler nicht damit beschäftigen muss und ein externer Designer diese Aufgabe übernehmen kann. Andererseits können bereits getätigte Eingaben nicht so leicht verloren gehen. Des Weiteren wurde während der Entwicklung der einzelnen Funktionen ständig auf die Merkmale der anderen Anforderungspunkte geachtet und diese häufig verbessert und aktualisiert. Abbildung 3.3 zeigt den gesamten Entwicklungsprozess seit dem Start des Projektes TU Jump into the Future. [Eid15a]

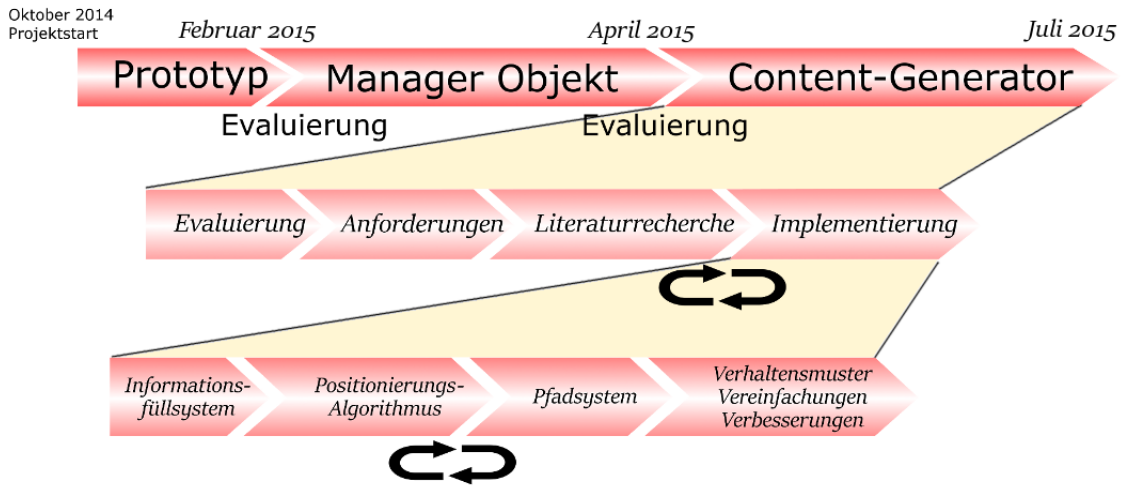

Abbildung 3.3: Diese Grafik visualisiert den Entstehungsprozess des Content-Generators und seinen Ursprung im Prototypen des Projekts TU Jump into the Future, das im Oktober 2014 begonnen hat.

KAPITEL 4

Systembeschreibung

In diesem Kapitel werden die einzelnen Komponenten des Content-Generators aufgezählt und detailliert beschrieben. Es wird genauer darauf eingegangen, wie diese Komponenten zusammenarbeiten und welchem Zweck sie dienen. Des Weiteren wird erläutert, welche Vorteile die Entwicklerin dadurch erhält und wie die verschiedenen Elemente im Unity-Editor verwendet werden können.

Zunächst geben wir einen Überblick über das gesamte Werkzeug und beschreiben, wie die einzelnen Komponenten im Zusammenhang stehen. Hierzu wird anhand einer Beispielsimulation erklärt, aus welchen wichtigen Bestandteilen unsere Simulationen bestehen. Des Weiteren wird beschrieben, wie ein Spieler-Objekt definiert sein kann und wie der Content-Generator mit diesem umgeht. Wir gehen auch darauf ein, mit welchen weiteren Daten der Content-Generator gefüllt werden kann, um eine Simulation erzeugen zu können. Hier wird kurz das Wegpunktsystem beleuchtet, sowie der Aufbau der Informationsobjekte.

Im Unterkapitel 4.2 erläutern wir anhand von Grafiken den genauen Aufbau des Systems, das zur Erzeugung von geeigneten Pfaden in unseren Simulationen dient. Wir gehen darauf ein, wie die ursprüngliche Pfaderzeugung war und was uns dazu veranlasste, dieses System zu verändern. Hier werden wir sehen, dass dieses Pfadsystem auch in vielen anderen Anwendungen Nutzen finden kann. Des Weiteren wird auch darauf eingegangen, welchen Sinn Catmull-Rom-Splines für virtuelle Simulationen haben und weshalb dieses System für die Pfaderzeugung gewählt wurde.

Im Abschnitt 4.3 wird beschrieben, wie die Struktur der Informationsobjekte aufgebaut ist und wie diese Objekte dazu genutzt werden können, Informationen zu speichern und wiederzugeben, sobald sie vom Benutzer aktiviert werden. Auch wird hier kurz erläutert, wie die Informationsobjekte mit einem speziellen Verhalten versehen werden können - was im Abschnitt 4.5 genauer beschrieben wird.

Im Abschnitt 4.4 wird darauf eingegangen, weshalb wir das spezielle Objekt mit dem Namen Content-Generator benötigen und welche Funktionen uns dieses bietet. Neben

dem Zweck des Generators beschreiben wir die genaue Funktionsweise und erklären, wie dieses Objekt auf die vorher beschriebenen Systeme des Pfades und der Informationsobjekte zugreift und diese verwaltet. So wird gezeigt, dass der Content-Generator als Manager hinter allen anderen bereits beschriebenen Funktionen steht und auf diese Weise erst die ganze Anwendung möglich macht. Genauer wird darauf eingegangen, wie die Positionierung der Informationsobjekte auf dem Pfad funktioniert. Des Weiteren wird erklärt, wie diese Informationsobjekte mit dem multimedialen Inhalt gefüllt werden können und welche unterschiedlichen Methoden dafür zur Verfügung stehen.

Abschließend können wir im Abschnitt 4.5 erfahren, welche unterschiedlichen Funktionen bereit stehen, um den Informationsobjekten zusätzliche Funktionalität zu verleihen und diese dynamischer gestalten zu können. Dazu werden die verschiedenen Verhaltensweisen aufgelistet und genauer beschrieben. Wir gehen dabei darauf ein, welchem Zweck die unterschiedlichen Funktionen dienen und wozu diese verwendet werden können. Schließlich werden wir erkennen, dass viele dieser Verhaltensweisen in sehr unterschiedlichen Simulationen Anwendung finden können.

4.1 Basissystem

Im Folgenden wird beschrieben, wie der Content-Generator aufgebaut ist und wie dieses System funktioniert. Dazu werden die einzelnen Komponenten kurz beleuchtet und unter der Zuhilfenahme von Bildern beschrieben. Speziell wollen wir hier einen Überblick über die notwendigen Objekte in unseren Simulationen vermitteln, um für die nachfolgenden Abschnitte bereits ein grundlegendes Wissen zu schaffen. Hierzu veranschaulichen wir kurz, wie mit dem Content-Generator-Werkzeug eine Simulation erstellt werden kann, in der sich eine Person durch eine Stadt bewegt und währenddessen über vordefinierte Informationsobjekte Informationen sammeln kann. Als Stadtmodell dient uns die Unity-Szene, die wir in Abbildung 4.1 sehen können.

Um eine Simulation erstellen zu können wird zunächst ein Spieler-Objekt benötigt, das sich durch die virtuelle Umgebung bewegen kann. Dieses Objekt kann sehr unterschiedlich dargestellt werden. Eine mögliche Variante ist beispielsweise ein menschlicher Avatar. Allerdings könnte das Spieler-Objekt auch in einem Fahrzeug, in einem Achterbahnwaggon oder in einem Flugzeug positioniert werden. Für unsere Simulation reicht in diesem Fall eine einfache Kamera, die keinen Körper besitzt - wie man es aus unterschiedlichen Spielen gewohnt ist.

Des Weiteren muss ein Pfad definiert werden, der angibt, wohin wir uns in unserer Simulation begeben wollen. Das heißt, es müssen Wegpunkte angegeben werden, zu denen sich unsere Kamera in sequentieller Reihenfolge bewegen soll. Glücklicherweise beinhaltet unser in Unity erstelltes Prefab des Content-Generators bereits einige Wegpunkte, die durch den Unity-Editor in der Szenen-Ansicht angezeigt werden können. Nachdem wir die Wegpunkte in unserer virtuellen Szene auf die gewünschten Positionen verschoben haben, befinden sich in unserem Hierarchiefenster zusätzlich zu der Stadtszene auch unser Spieler-Objekt, das mit einer Kamera ausgestattet ist, sowie der Content-Generator mit

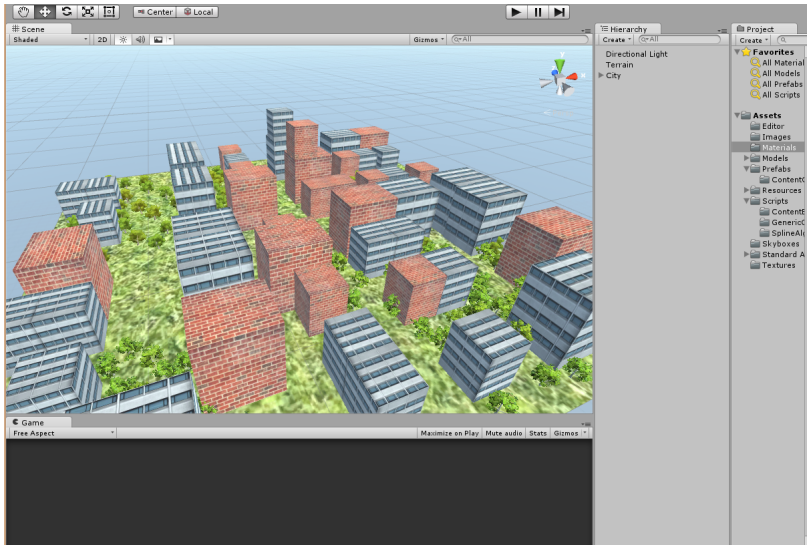

Abbildung 4.1: Eine einfach modellierte Stadtszene in Unity3D.

insgesamt zehn Wegpunkten (siehe Abbildung 4.2). Die Wegpunkte sind dabei durch farbige Sphären markiert und der Pfad als blaue Linie eingezeichnet. Die blaue Sphäre stellt den Anfangspunkt der Simulation dar, die rote Sphäre den Endpunkt. Der gesamte Informationsinhalt, dem die Benutzerin später in der Simulation begegnen wird, spielt sich zwischen den grünen Sphären ab.

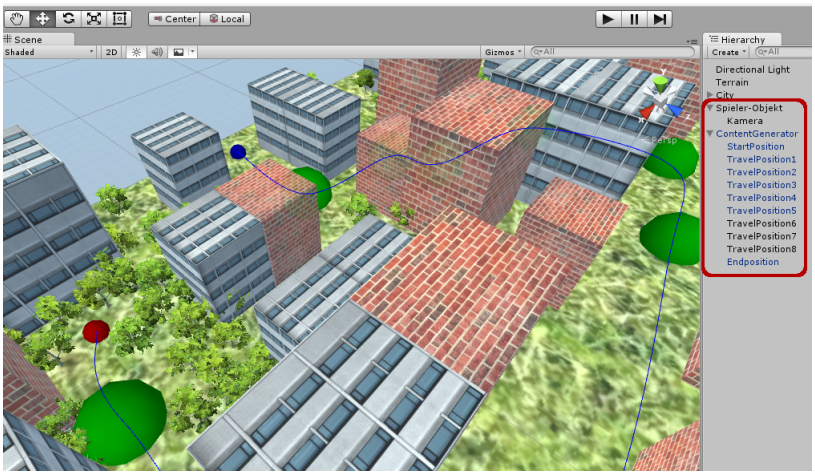

Abbildung 4.2: Zusätzlich zur Stadt befindet sich nun auch das Spieler-Objekt, sowie der Content-Generator und die Wegpunkte in der Szene. Der Pfad, dem das Spieler-Objekt folgen wird, ist in der Szenen-Ansicht des Unity-Editors als blaue Linie eingezeichnet. Die grünen Sphären stellen die vordefinierten Wegpunkte des Pfades dar.

Da nun das Spieler-Objekt und der Pfad definiert sind, ist es nötig, die Informations-

objekte zu erstellen, die schließlich verwendet werden, um dem Benutzer Informationen präsentieren zu können. Die Art der Objekte spielt hierbei keine Rolle. Es kann sich dabei beispielsweise um Gebäudewände oder Leinwände handeln, jedoch können auch Fahrzeuge oder Flugzeuge dazu verwendet werden. Grundsätzlich kann jedes Spiel-Objekt dafür genutzt werden, solange es als Komponente einen Mesh-Renderer besitzt, auf dem Texturen dargestellt werden können. Für unsere Zwecke reichen vier rechteckige Objekte, auf denen später die Bilder visualisiert werden. Für den dazugehörigen Text wurde ein Objekt verwendet, das normalerweise in Unity für die Darstellung der graphischen Benutzeroberfläche (GUI) verwendet wird. Dieses Objekt besitzt eine Textkomponente, weshalb sie für diesen Zweck sehr nützlich ist. Wie in Abbildung 4.3 zu sehen, wurden die Objekte auch mit einer weiteren Komponente ausgestattet, die es den Objekten ermöglicht, Audiodateien wiederzugeben. Dies wird in Unity mittels weißer Lautsprecher-Symbole angezeigt.

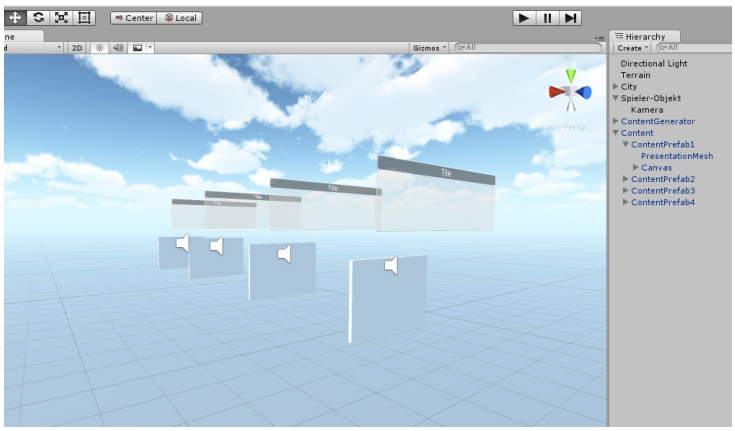

Abbildung 4.3: Vier Informationsobjekte wurden in der Szene erstellt, die dazu dienen, der Benutzerin während der Simulation mittels Bild, Text und Ton Informationen zu vermitteln.

Zusätzlich zum Spieler-Objekt, den Wegpunkten und den Informationsobjekten muss der Content-Generator auch noch alle Informationen über die Bilder, die Texte und die Audiodateien - sowie deren Dauer - erhalten. All diese Informationen können im Unity-Editor auf einfache Weise via Drag-And-Drop an den Content-Generator weitergegeben werden. Sobald dies geschehen ist, kann die Anwendung gestartet werden und der Content-Generator erzeugt mit seinen Informationen die Simulation, indem er das Spieler-Objekt an seine Startposition setzt. Anschließend wird die maximale Spieler-Geschwindigkeit aus der Wegstrecke und der Abspielzeit des multimedialen Inhaltes berechnet. Ist dies geschehen, werden die vier Informationsobjekte mit dem multimedialen Inhalt befüllt und auf dem Pfad in passendem Abstand voneinander platziert, sodass die Simulation gestartet werden kann. Abbildung 4.4 visualisiert die Funktion des Content-Generators als Manager in diesem Zusammenspiel.

In diesem Abschnitt gaben wir nun einen kurzen Einblick darüber, aus welchen wich-

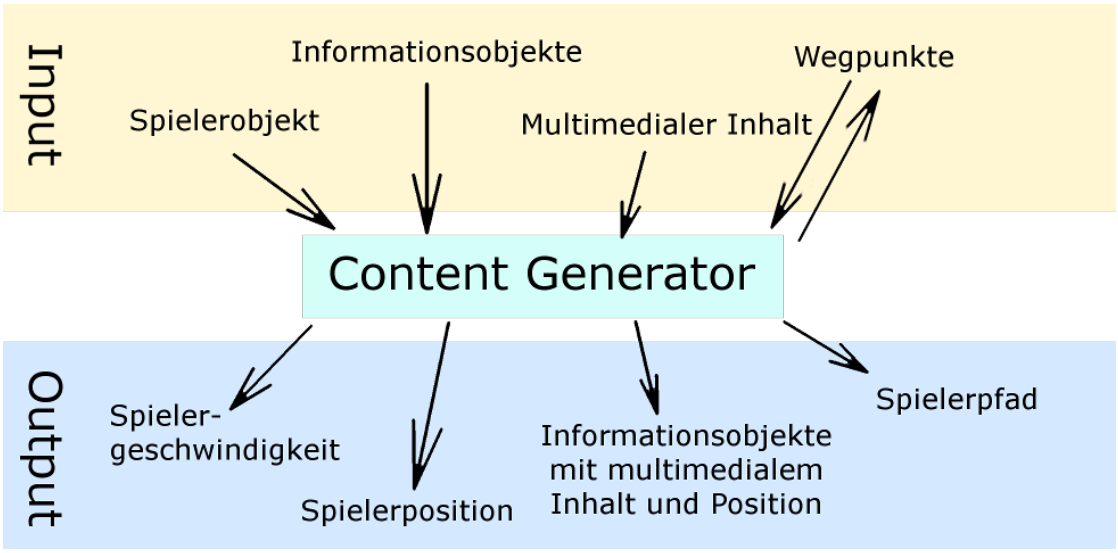

Abbildung 4.4: Diese Grafik spiegelt die Funktionsweise des Content-Generators wider. Aus den eingegebenen Daten berechnet das System die Spielergeschwindigkeit und den Spielerpfad, sowie die anfängliche Spielerposition. Außerdem wird die Position der Informationsobjekte kalkuliert, welche mit dem multimedialen Inhalt gefüllt werden.

tigen Bestandteilen unsere Simulationen bestehen und wie uns der Content-Generator als Werkzeug dient, um mit wenigen einfachen Schritten eine Simulation erstellen zu können. In den nachfolgenden Abschnitten wird genauer auf die einzelnen Komponenten dieses Systems eingegangen. Unter anderem berichten wir im Kapitel 4.5 darüber, wie die Informationsobjekte mit weiteren Funktionen ausgestattet werden können, um sie dynamischer zu gestalten.

4.2 Wegpunkt- und Pfadsystem

Um die Bewegung des Spieler-Objektes in der Szene zu ermöglichen wurde bereits in der Prototyp-Version unseres Projekts [Eid15a] ein Wegpunktsystem verwendet. Als Wegpunkte dienten spezielle Objekte, die in der dreidimensionalen Szene positioniert werden konnten und zu denen sich der Spieler im Laufe der Simulation hin bewegte. Allerdings bewegte sich der Spieler auf geraden Linien zwischen diesen Wegpunkten, was zwar für diese Simulation ausreichend war, uns jedoch für spätere, komplexere Simulationen als unpassend erschien. Auch wurde die Strecke zwischen den Wegpunkten dem Entwickler nicht visualisiert. So mussten viele Testdurchläufe gestartet werden, sobald sich am Pfad etwas veränderte. In unserem neuen Pfadsystem wurden diese Aspekte überarbeitet.

Zunächst wurde versucht, den Pfad zu visualisieren, um dem Entwickler einen besseren Überblick darüber zu verschaffen, auf welcher Strecke sich der Spieler bewegen wird.

Auf diese Weise sollte es leichter werden, spezielle Objekte, die der Spieler durch seine Bewegung auslösen muss, zu platzieren. Auch würde es leichter werden, die Umgebung so zu formen, dass sich die Kamera nicht durch unpassierbares Gelände hindurch bewegt. Unser Manager-Objekt, das den Vorgänger zum Content-Generator darstellt (siehe Abschnitt 3.3), versuchte dieses Problem mithilfe eines Line-Renderers im Unity-Editor zu lösen. Dieser wird jedoch nur visualisiert, wenn die Simulation gestartet wird. Nachdem allerdings Änderungen in einem Testlauf nicht für die eigentliche Szene übernommen werden (wie am Ende des Abschnitts 2.3.1 erwähnt), ist das nicht die zweckdienlichste Art, um den Pfad zu visualisieren. Im Unity-Editor ist stattdessen das Gizmo-Feature dafür vorgesehen, das im Content-Generator nun für die Visualisierung dient. In Abbildung 4.5 können wir sehen, wie sich dieses Visualisierungssystem auf unseren Pfad auswirkt. So wurde der Pfad zu Beginn noch rein durch gerade Linien dargestellt. Um einen besseren Überblick zu erhalten, wurden später Sphären hinzugefügt, die unsere Wegpunkte, sowie den Start- und Endpunkt visualisieren. Vor allem, als schließlich der gerade Pfad in Catmull-Rom-Splines umgewandelt wurde, war das Anzeigen der Kontrollpunkte sehr nützlich, um den Pfad besser kontrollieren zu können.

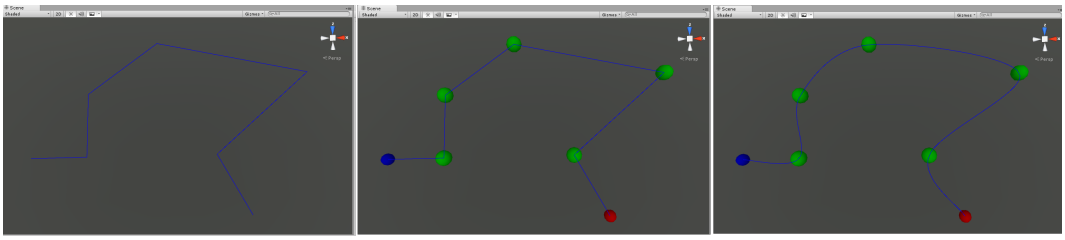

Abbildung 4.5: Diese Abbildung zeigt die Veränderung der Visualisierung des Spieler-Pfades im Laufe der Entwicklung. So wurde zu Beginn noch ein gerader Pfad a) erstellt. Später wurden die Wegpunkte des Pfades mit Sphären b) markiert. In der finalen Version c) beschreibt ein Catmull-Rom-Algorithmus den Verlauf des Pfades und bildet so einen geglätteten Pfad, ohne scharfe Wendepunkte.

Die Umstellung von geraden Linien zu Catmull-Rom-Splines geschah aus mehreren Gründen. Uns war wichtig, dass wir in unseren Pfaden keine scharfen Kanten haben, da wir davon ausgingen, diese kaum bis gar nicht zu benötigen. Auch in unserem bereits erstellten Fallschirmsprung-Simulator werden keine scharfen Kanten benötigt. Beispielsweise wird die Flugbahn eines Springers beim Öffnen des Fallschirms auch eher abgerundet verlaufen und langsam abflachen, als dass der Springer von einem Moment auf den anderen seine Richtung komplett verändert. Auch für weitere Simulationen, wie zum Beispiel Fahrzeug- oder Flugzeugsimulationen, sowie Tauchgänge oder Reisen im Weltall, machen abgerundete Pfade mehr Sinn. Des Weiteren förderte die Literaturrecherche unser Bestreben in diese Richtung. Vor allem der Bereich Cybersickness hatte großen Einfluss darauf. Um den Effekten von Cybersickness entgegen zu wirken, macht es Sinn, den Verlauf des Spieler-Pfades geglättet zu designen, um den Spieler auf die bevorstehenden Ereignisse besser vorbereiten zu können und schnelle Rotationen zu vermeiden. Durch

die Literaturrecherche haben wir auch gelernt, dass es besser ist, den Benutzer langsam an die virtuelle Umgebung zu gewöhnen. Deshalb sind wir bemüht, nur solche Pfade zu erzeugen, die zu Beginn eher schwache Rotationen aufweisen und generell keine scharfen Kurven beinhalten. Auch sollte es vermieden werden, zu schnelle Bewegungen in die Simulation zu bringen, um die Sinne des Benutzers nicht zu überfordern. Wie dies in unseren Simulationen bewerkstelligt wird, beschreiben wir in den folgenden Abschnitten.

4.3 Contentstruktur

Die Contentstruktur steht hinter den Informationsobjekten. Da wir für unsere Simulationen möglichst viele unterschiedliche Objekte zu Informationsobjekten machen wollen, ist es notwendig, diese Struktur möglichst allgemein zu beschreiben, wobei der Content-Generator weiterhin mit diesen Objekten umgehen können muss. Grundsätzlich kann jedes Objekt in einer Unity-Szene zu einem Informationsobjekt gemacht werden. Wichtig dafür ist, dass dieses Objekt als *Tag* das Stichwort Content erhält und es als Komponente das *ContentScript* besitzt. Des Weiteren benötigt es einen Collider bei dem das Trigger-Attribut gesetzt ist, sodass das Objekt aktiviert werden kann, sobald das Spieler-Objekt mit diesem kollidiert. In Abbildung 4.6 sind diese Komponenten im Inspektor des Informationsobjektes hervorgehoben.

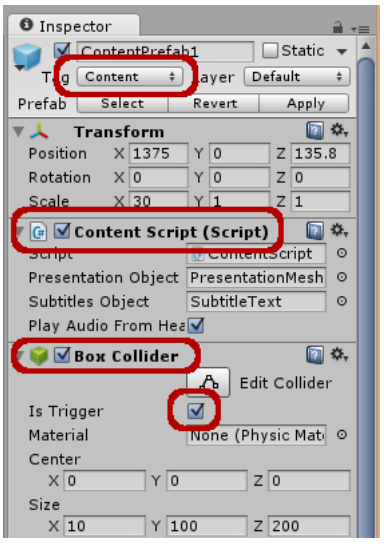

Abbildung 4.6: Entscheidend für ein Informationsobjekt ist, dass es den *Tag* Content, sowie die Komponenten *ContentScript* und *Collider* (mit gesetztem Trigger-Attribut) erhält.

Sobald die erforderlichen Komponenten vorhanden sind, können weitere Objekte im Inspektor des Informationsobjektes definiert werden, die schließlich für die tatsächliche Wiedergabe der multimedialen Inhalte verwendet werden. Zum einen ist dies das Präsentations-Objekt, welches die vorgesehenen Bilder bzw. das vorgesehene Bild anzeigen

wird, zum anderen das Objekt, das den dazugehörigen Text visualisiert - hierbei ist es wichtig, dass dieses Objekt eine Textkomponente besitzt. Wir haben hierfür, wie im Abschnitt 4.1 beschrieben, auf ein Unity-GUI-Objekt zurückgegriffen. Schließlich besteht noch die Möglichkeit, zu wählen, ob die vorgegebene Audiodatei vom Informationsobjekt selbst, oder direkt von den Kopfhörern des Spieler-Objektes abgespielt werden soll. Der zweite Fall hat aus unserer Erfahrung den Vorteil, dass beispielsweise eine Musikhinterlegung lauter und somit dominanter wirkt, wenn sie direkt über die Kopfhörer des Spieler-Objektes abgespielt wird. Das kommt daher, dass die Audio-Quelle in diesem Fall näher am Spieler ist. Der Nachteil besteht darin, dass dafür ein Kopfhörer-Objekt definiert werden muss, das den *Tag* PlayerHeadphones, sowie eine Audio-Source-Komponente besitzt. Beides ist jedoch über den Unity-Editor schnell verwirklicht.

Auf diese Weise können beliebig viele Informationsobjekte für die Simulation erstellt werden. Damit die Übersicht der Hierarchie durch die Verwendung von sehr vielen Informationsobjekten nicht leidet, wurde die Contentstruktur so gewählt, dass alle Informationsobjekte Kinderobjekte von einem gesonderten Spiel-Objekt sein müssen. Dadurch kann im Hierarchie-Fenster die gesamte Struktur geschlossen werden, sodass nur ein Objekt in der Hierarchie angezeigt wird. Abbildung 4.7 zeigt, wie eine solche Objekt-Hierarchie in Unity aussehen kann.

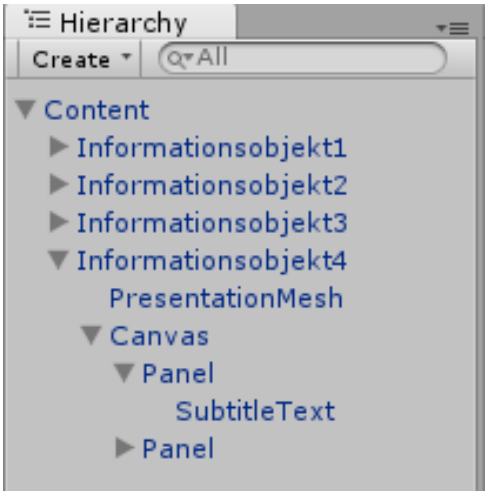

Abbildung 4.7: Die Struktur der Informationsobjekte wird im Unity-Editor so gewählt, dass die Übersicht in großen Unity-Szenen gewahrt wird. Durch hierarchische Anordnung können alle Informationsobjekte geschlossen werden, wodurch zuletzt nur ein Objekt im Hierarchie-Fenster angezeigt wird.

Wie wir im nächsten Kapitel sehen werden, übernimmt der Content-Generator die Befüllung der Informationsobjekte mit Bildern, Texten und Audiodateien. Wird während der Simulation ein Informationsobjekt aktiviert, indem der Spieler mit dessen Kollisionskomponente in Berührung kommt, werden sequentiell die vordefinierten Bilder und die Texte angezeigt, sowie die Audiodatei abgespielt. Wie und wann dies geschieht, hängt

dabei davon ab, wie das jeweilige Informationsobjekt aufgebaut ist und welche Position die Kollisionskomponente besitzt.

Als Informationsobjekte können selbstverständlich sehr viele verschiedene Objekte in Frage kommen. In unserer Fallschirmsprung-Simulation wurden ganz einfache rechteckige Objekte verwendet, die jeweils nur ein Bild präsentieren und ihre Position nicht verändern. Allerdings hatten sie über zusätzliche Komponenten die Eigenschaft, zu Beginn der Simulation unsichtbar zu sein und sich immer dem Spieler zuzuwenden. Weitere solche Verhaltensweisen werden im Abschnitt 4.5 beschrieben. Nachdem rechteckige Objekte als Informationsträger für bestimmte Simulationen nicht sehr spannend sein mögen, zählen wir in Tabelle 4.1 einige mögliche Informationsobjekte auf, die beispielsweise verwendet werden können.

Informationsobjekt	Präsentations-Obj.	Text-Objekt	Soundquelle
Bus	Busseite	Teil der Leinwand	Bus
Fernseher	Display	Teil des Displays	Fernseher
Klassenzimmer	Leinwand	Teil des Displays	Lautsprecher
Informationspult im Museum	Display	Teil des Displays	Lautsprecher
Ein Fischschwarm im Ozean	Keines	Keines	Kopfhörer
Ein Planet im Weltall	Keines	Keines	Kopfhörer
Ein animierter Engel	Keines	Keines	Engel

Tabelle 4.1: Eine beispielhafte Auflistung von möglichen Informationsobjekten und deren Funktionen.

4.4 Content-Generator

Wie bereits im Abschnitt 3.3 beschrieben wurde, leitet sich der Content-Generator von seinem Vorgänger, dem Manager-Objekt, ab und hat primär die Aufgabe, die Informationsobjekte zu verwalten. Das Manager-Objekt wurde zunächst dazu verwendet, die Informationsobjekte auf einer geraden Linie in der dreidimensionalen Szene zu platzieren. Später wurde aus Gründen der Übersichtlichkeit dieses Objekt so erweitert, dass es auch für die Verwaltung der Bilder, Texte und Audiodateien verantwortlich war, die während der Simulation durch die Informationsobjekte angezeigt werden sollten. Die Evaluierung ergab jedoch, dass das Manager-Objekt an zu wenig Übersichtlichkeit litt und so stellte sich uns die Frage, ob die Eingabe der Daten über den Inspektor nicht übersichtlicher gestaltet werden könnte, als es zu diesem Zeitpunkt der Fall war. Auch war es bis dahin nicht möglich, über das Manager-Objekt die Informationsobjekte mit mehr als einem Bild zu befüllen, was dazu führte, dass nicht verschiedene Bilder von einem Informationsträger sequentiell angezeigt werden konnten. Dies war für den Fallschirmsprung-Simulator zwar kein Problem, für zukünftige Projekte stellt dies jedoch ein Hindernis dar. Neben dieser Funktion sollte auch die Positionierung der Informationsobjekte sowie der Prozess der

Pfaderstellung verbessert werden. Auch sollten weitere Aspekte angeboten werden, wie etwa die Befüllung der Informationsobjekte durch eine externe Informationsdatei. Auf all diese Punkte gehen wir in diesem Abschnitt ein.

Die erste und wichtigste Verbesserung, die der Content-Generator mit sich bringt, stellt der neue Positionierungs-Algorithmus der Informationsobjekte auf der Catmull-Rom-Kurve dar. Auf die Funktionsweise dieses Algorithmus werden wir hier genauer eingehen. Damit die exakten Positionen berechnet werden können, ist es erforderlich, dass alle Parameter, die mit den Informationsobjekten, sowie mit dem Pfad in der virtuellen Welt in Zusammenhang stehen, bekannt sind. Es sind dies die vordefinierten Kontrollpunkte und die daraus resultierende Catmull-Rom-Kurve, sowie die Dauer der einzelnen Informationsbilder und Informationstexte, die während der Simulation wiedergegeben werden. Dadurch, dass wir wissen, wie lange die Informationsobjekte benötigen, um ihren Inhalt wiederzugeben und mit dem Wissen um die Gesamtlänge des Pfades, können wir die maximale Geschwindigkeit des Spieler-Objektes kalkulieren. Da nun auch bekannt ist, wie weit sich das Spieler-Objekt pro Sekunde bewegen wird, können wir berechnen, in welchen Entfernungen zueinander sich die Informationsobjekte schließlich befinden müssen. Das bedeutet, dass wir uns in unserem Algorithmus von jedem Punkt auf dem Pfad zum nächsten Punkt bewegen und uns dabei die zurückgelegte Strecke merken. Sobald wir an eine Position kommen, die weit genug von dem vorherigen Informationsobjekt entfernt liegt, können wir die Position des neuen Informationsobjektes berechnen, da diese zwischen den letzten beiden Punkten liegen muss. In Abbildung 4.8 können wir sehen, dass die Objekte mit ausreichend Abstand entlang des Pfades platziert werden. Für einige Simulationen (wie unseren bereits beschriebenen Fallschirmsprung-Simulator) macht es Sinn, die Informationsobjekte nicht direkt auf dem Pfad zu platzieren, sondern etwas versetzt an der Seite oder oberhalb. Der Content-Generator bietet die Möglichkeit, jeden Durchlauf der Simulation anders zu gestalten, indem die Objekte über einen Zufallsfaktor (der vom Designer bestimmt werden kann), im dreidimensionalen Raum etwas verschoben werden. Möchte man nur bestimmte Informationsobjekte verschieben, gibt es einen weiteren Lösungsansatz: Ein Testlauf muss gestartet werden, damit die Objekte richtig platziert werden. Der Content mit den richtigen Positionen wird kopiert und der Testlauf beendet. Nun kann der kopierte Content in die Szene eingefügt werden, wodurch er bereits passend positioniert ist. Wenn man an diesem Content Änderungen macht und beim Content-Generator die Funktion des automatischen Platzierens der Informationsobjekte deaktiviert, werden beim nächsten Testlauf (und in der fertigen Applikation) die aktuellen Positionen beibehalten.

Damit unsere Simulationen reibungslos funktionieren, ist es wichtig, dass von außen keine Kraft auf die Spieler-Geschwindigkeit wirkt, sodass diese die maximale Geschwindigkeit nicht überschreitet. Die Spieler-Geschwindigkeit kann jedoch zum Beispiel an bestimmten Wegpunkten verringert werden, was beispielsweise am Ende der Simulation Sinn macht, um die Benutzerin sanft am Zielpunkt ankommen zu lassen und nicht ruckartig zu stoppen. Bei Sprung-, Fahrzeug-, oder Flugsimulationen wird die Benutzerin genau das erwarten.

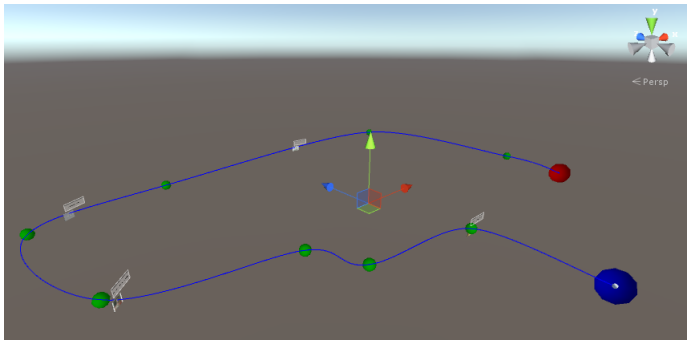

Abbildung 4.8: Der Content-Generator platziert die Informationsobjekte so auf der Catmull-Rom-Kurve, dass es während der Simulation zu keinen Überschneidungen des multimedialen Inhaltes kommt.

Die Befüllung der Informationsobjekte mit Bildern, Texten und Audiodateien verläuft aus Gründen der Übersichtlichkeit ebenfalls über den Content-Generator. Hier bieten sich zwei Ansätze. Zum einen kann im Inspektor-Fenster des Content-Generators manuell eine Liste erstellt und angepasst werden, zum anderen ist es den Entwicklern möglich, eine Informationsdatei anzugeben, in der alle Bilder, Texte und Audiodateien definiert sind, sodass sie der Content-Generator im Projektordner finden und verwalten kann. Zunächst gehen wir auf den manuellen Befüllungsvorgang ein. Wie wir in Abbildung 4.9 sehen können, wird zu Beginn im Inspektor definiert, um wie viele Informationsobjekte es sich in der Simulation handelt. In diesem Fall sind dies vier Objekte. Dadurch erhält man eine Liste, die aus vier Elementen besteht und die der Entwickler oder der Designer mit Werten füllen kann. Dazu wurde für den ersten Eintrag der Liste folgendes gewählt: als Informationsobjekt dient das Objekt *ContentPrefab1*; angezeigt werden sollen zwei Bilder, wobei das erste 5 Sekunden und das zweite Bild 7 Sekunden angezeigt werden soll. Das erste Bild trägt den Namen *01_doppler*, das zweite Bild *02_dopplereffekt*. Danach können noch Untertitel angegeben werden, die passend zu den Bildern angezeigt werden, sowie eine Audiodatei, die während der Dauer der Präsentation vom Informationsobjekt abgespielt wird. Derselbe Vorgang lässt sich für die nächsten drei Einträge in dieser Liste wiederholen, sodass man schließlich eine fertige Simulation mit vier Informationsobjekten erhält. Der zweite Ansatz, um die Informationsobjekte mit Inhalten zu befüllen, funktioniert - wie bereits erwähnt - über die Angabe eines Informationsdokumentes. In diesem Dokument müssen, ähnlich wie bei der manuellen Befüllung, alle wichtigen Daten angegeben werden. Allerdings mit dem Unterschied, dass für die Bilder und Audiodateien die Dateipfade angegeben werden müssen. Ist das geschehen, kann die Datei für beliebig viele Simulationen genutzt werden, ohne dass ein zusätzlicher Aufwand entsteht. Außerdem kann das auch durch einen Designer gemacht werden, ohne dass dieser sich in die Unity-Entwicklungsumgebung einarbeiten muss. Beide Varianten haben Vor- und Nachteile. Für welche Variante man sich entscheidet, hängt wohl davon ab, wie sehr man mit dem Unity-Editor vertraut ist.

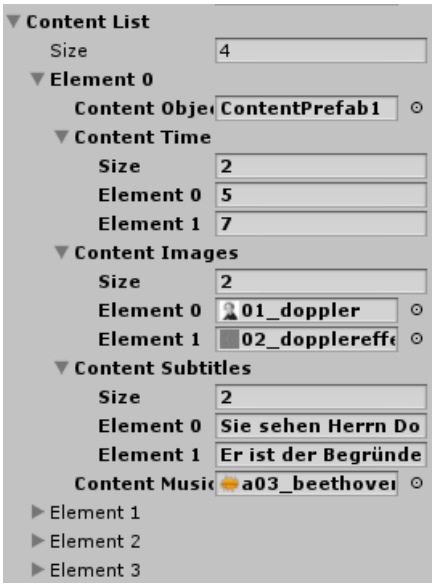

Abbildung 4.9: Über das Inspektor-Fenster des Content-Generators können die Informationsobjekte mit Bildern, Texten und Audiodateien befüllt werden, um diese während der Simulation der Benutzerin präsentieren zu können.

Neben den Aufgaben der Objektpositionierung und der Verwaltung der multimedialen Informationen dient der Content-Manager auch der Erzeugung von zusätzlichen Wegpunkten. So wird im Inspektor-Fenster eine Liste mit Wegpunkten verwaltet, die beliebig erweitert werden kann, um den Pfad des Spieler-Objektes zu gestalten. Wie diese Liste aussehen kann, zeigt Abbildung 4.10. Um auf schnelle Weise neue Wegpunkte hinzufügen zu können, dient ein eigens definierter Button, der automatisch einen neuen Wegpunkt in der virtuellen Szene hinzufügt und so den Spieler-Pfad erweitert.

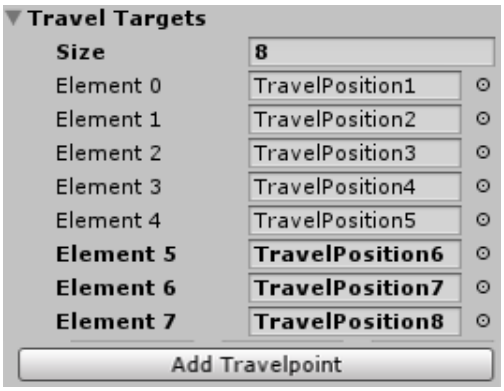

Abbildung 4.10: Der Content-Generator verwaltet die Wegpunktliste. Mit einem Mausklick können weitere Wegpunkte hinzugefügt und der Pfad auf diese Weise schnell erweitert werden.

Wie wir in diesem Abschnitt erfahren konnten, wurde der Content-Generator zu jenem Zweck entwickelt, Informationsobjekte zu verwalten und mit multimedialem Inhalt zu füllen. Dies funktioniert entweder manuell, oder mithilfe einer vordefinierten Informationsdatei. Des Weiteren besteht seine Aufgabe darin, das Wegpunkt- und Pfadsystem zu erzeugen und zu visualisieren, sowie das Spieler-Objekt und die Informationsobjekte passend auf der Catmull-Rom-Kurve zu platzieren. Dabei stehen dem Entwickler einige weitere Funktionen zur Verfügung, wie etwa jene, mit der die Informationsobjekte auf zufällige Weise etwas von ihrer eigentlichen Position versetzt werden. Auch bietet sich die Möglichkeit, die Informationsobjekte nachdem sie platziert wurden noch händisch zu verschieben, sollte dies den Wünschen der Entwickler entsprechen. Somit stellt der Content-Generator per se ein relativ umfangreiches Werkzeug dar, mit dem viele unterschiedliche Simulationen auf schnelle Art und Weise erstellt werden können.

4.5 Contentverhalten

Einen weiteren wichtigen Punkt in den Simulationen, die mithilfe des Content-Generators erstellt werden sollen, stellt das Verhalten der Informationsobjekte dar. Es war uns wichtig, dass die Informationsträger nicht starr in der virtuellen Welt verharren, sondern dass es möglich sein soll, ihnen eine gewisse Dynamik verleihen zu können, sodass die Simulationen interessanter werden. So wurden einige Funktionen erstellt, die den Informationsträgern als zusätzliche Komponenten hinzugefügt werden können und ihnen so ein zusätzliches Verhalten neben der Präsentation der multimedialen Inhalte ermöglichen. Wir werden im Folgenden auf die einzelnen Funktionen genauer eingehen.

Bereits im Projekt [Eid15a] wurden die Informationsobjekte mit dem Verhalten ausgestattet, das verursachte, dass sie zu Beginn der Simulation unsichtbar waren. Erst wenn sich der Spieler nahe genug an das Objekt heran bewegt, wird es langsam sichtbar. Das bringt vor allem Vorteile, wenn sich in der virtuellen Szene sehr viele Informationsobjekte auf einmal im Blickfeld befinden. Zum einen bleibt der Rest der Szene frei und wird nicht von den Informationsobjekten verdeckt und zum anderen kann der Benutzer nicht den ganzen Ablauf im Vorhinein erkennen, wodurch die Simulation spannend bleibt.

Ein weiteres Verhalten, das bereits im Fallschirmsprung-Simulator enthalten war verursacht, dass sich die Informationsobjekte stets in die Richtung des Spieler-Objektes drehen, wodurch der Spieler ihren Informationsinhalt zu jedem Zeitpunkt sehen kann. Sehr gut lässt sich dieses Verhalten mit einer weiteren Funktion kombinieren, wie beispielsweise dem Merkmal, welches die Informationsobjekte dazu veranlasst, den Spieler auf seinem Weg zu begleiten. Hierzu muss ein Objekt angegeben werden, zu dem sich das Informationsobjekt nach seiner Aktivierung bewegen soll. In unseren Simulationen wurden dafür Objekte angegeben, die in der Hierarchie des Spieler-Objektes verankert waren und sich so immer in der Nähe des Spielers befanden. Selbstverständlich kann dafür auch ein anderes Objekt angegeben werden. Beispielsweise könnte das Informationsobjekt ein Kranhaken sein, der sich bei Aktivierung zu einem bestimmten Punkt bewegt und ein schweres Objekt aufhebt. Die Anwendungsarten dieses Verhaltens können daher sehr

unterschiedlich sein.

Für eine verbesserte Kontrolle über die Bewegungen der Informationsobjekte wurde ein Verhalten entwickelt, das ähnlich dem System von Spieler und Spieler-Pfad funktioniert. Es kann hierfür ein Wegpunktsystem für das Informationsobjekt definiert werden, sodass das Objekt dem erzeugten Pfad folgt (der wiederum eine Catmull-Rom-Kurve ist). Dabei kann ausgewählt werden, ob das Objekt erst nach seiner Aktivierung diesem Pfad folgen soll, oder sofort nach dem Beginn der Simulation. Dieses System eröffnet dem Designer sehr viele verschiedene Möglichkeiten, die Simulation zu beeinflussen. Es könnten beispielsweise Fische, Busse, Flugzeuge oder Meteoriten definiert werden, die einem solchen Pfad in der dreidimensionalen Umgebung folgen. Abbildung 4.11 zeigt einen solchen Pfad, der in einer Unity-Szene dafür verwendet wurde, ein Informationsobjekt zuerst durch ein Tal und später über eine Gebirgskette zu leiten, während sich das Spieler-Objekt weiterhin durch das Tal bewegt.

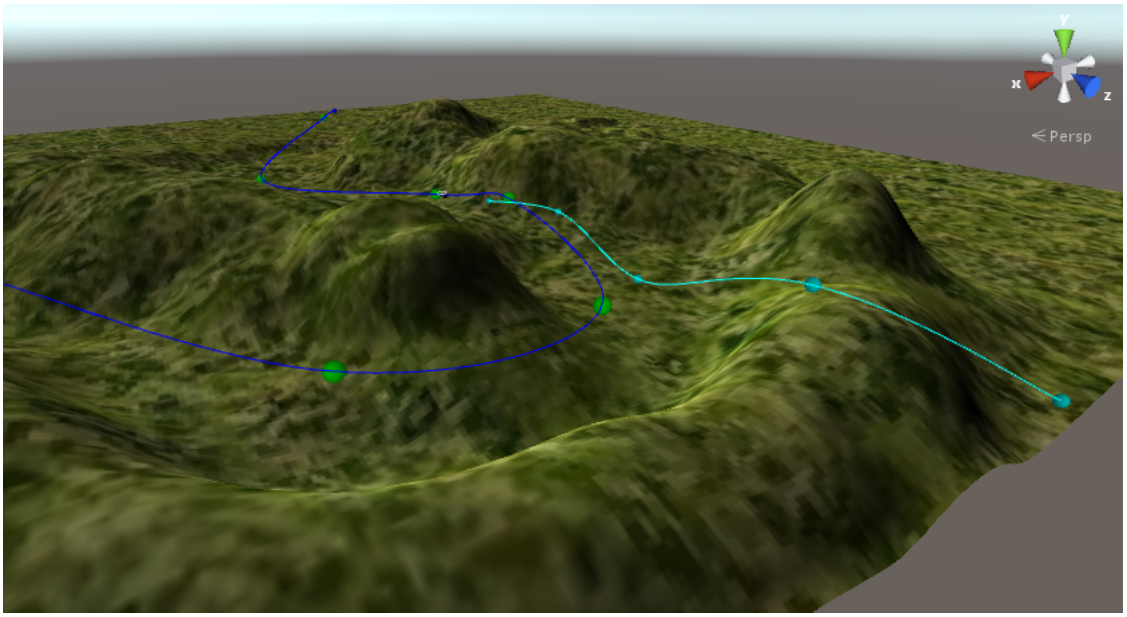

Abbildung 4.11: Für die Informationsobjekte kann ein Pfad definiert werden, dem diese Objekte nach ihrer Aktivierung folgen. In der Szenen-Ansicht wird ein solcher Pfad türkis visualisiert.

Schließlich wurde im Hinblick auf geplante Simulationen, in denen sich der Benutzer im Weltall bewegt ein zusätzliches Verhalten entworfen, welches das Informationsobjekt dazu veranlasst, um seine eigene Achse zu rotieren. Dieses Verhalten sollte speziell dazu dienen, Planeten rotieren zu lassen. Allerdings kann es auch für andere Objekte, wie beispielsweise Rotoren, oder Windräder verwendet werden.

Durch die unterschiedlichen Verhaltensweisen, die in Unity erstellt wurden, gehen wir davon aus, eine Vielzahl an Simulationen erstellen zu können, die alle auf diesem Werkzeug

basieren. Da Unity ein komponentenorientiertes System ist, sind diese Verhaltensweisen selbstverständlich nicht auf unsere Informationsobjekte begrenzt und können auch dazu verwendet werden, die Umwelt in den Simulationen lebendiger zu gestalten. Auch lassen sich viele der beschriebenen Verhaltenarten miteinander kombinieren. Beispielsweise macht es Sinn, die Informationsobjekte mit dem Spieler mit bewegen zu lassen, während sie sich ständig dem Spieler zuwenden, um ihm über ihre gesamte Dauer den Informationsinhalt präsentieren zu können. Für Planeten oder Kometen würde sich anbieten, einen Pfad zu definieren, dem diese folgen können, während sie um ihre eigene Achse rotieren. Diese und viele weitere Kombinationen sind denkbar.

KAPITEL 5

Zusammenfassung und Ausblick

In diesem Kapitel werden die Funktionen des beschriebenen Systems zunächst nochmals zusammengefasst und erläutert. Auch wird kurz auf das Projekt *TU Jump into the Future* eingegangen und erklärt, welche Funktion der Content-Generator in dieser Simulation erfüllt. Ebenfalls werden unsere Testläufe erläutert, in denen Entwickler, welche mit Unity vertraut waren, an das bestehende System herangeführt wurden und die Aufgabe hatten, eine Simulation in einer bestehenden Szene mithilfe des Content-Generators zu erstellen. Abschließend geben wir einen Ausblick auf zukünftige Simulationen und mögliche Erweiterungen des Content-Generators.

Der generische Content-Generator, der in Unity entwickelt wurde, dient dazu, Simulationen zu erstellen und diese zu Anwendungen zu erweitern, die den Benutzer mithilfe einer 3D-Brille in eine virtuelle Welt eintauchen lassen. Während sich der Benutzer durch die dreidimensionale Umgebung bewegt, können an ihn über so genannte Informationsobjekte Inhalte mittels Bilder, Texten und Audiodateien vermittelt werden. Der generische Content-Generator dient in der Entwicklung dieser Simulationen als Verwaltungsobjekt, das sowohl das Spieler-Objekt, als auch die Informationsobjekte, die multimedialen Inhalte und den Spieler-Pfad verwaltet. Der Pfad, dem der Spieler in der Simulation folgt, wird bereits in der Entwicklung der Anwendung durch Wegpunkte vordefiniert. Ein Catmull-Rom-Algorithmus berechnet aus den vorgegebenen Wegpunkten einen abgerundeten Pfad, der durch die Wegpunkte hindurch geht. Ein abgerundeter Pfad hat den Vorteil, dass sich der Benutzer bereits frühzeitig darauf einstellen kann, wohin er sich in naher Zukunft bewegen wird. Dadurch wirkt sich Cybersickness weniger stark auf den Zustand des Benutzers aus. Die Objekte, die dem Benutzer Informationen präsentieren sollen, werden durch einen speziellen Algorithmus auf dem vorher definierten Pfad auf solche Weise positioniert, dass keine Überschneidungen der multimedialen Inhalte auftreten. Um die Positionen ausrechnen zu können, muss die Gesamtdauer der Informationsinhalte, sowie die Länge des Pfades bekannt sein. Mithilfe der daraus errechneten maximalen Spieler-Geschwindigkeit können die Positionen der Informationsobjekte exakt bestimmt

werden. Damit das Werkzeug für möglichst viele unterschiedliche Simulationen effektiv genutzt werden kann, wurden Verhaltensweisen für die Informationsobjekte erstellt. Diese beeinflussen die Eigenschaften der Objekte, sodass sich diese beispielsweise mit dem Benutzer mit bewegen, sobald er nahe genug an sie heran kommt. Teile des Content-Generators wurden bereits in das bestehende Projekt *TU Jump into the Future* integriert und ermöglichen es uns, einen vier-minütigen virtuellen Fallschirmsprung über der Stadt Wien zu absolvieren. Die 42 Informationsobjekte, die sich in dieser Simulation befinden, besitzen einerseits das Verhalten, unsichtbar in der Szene zu starten und sich bei der Annäherung des Spielers zu zeigen und andererseits das Verhalten, sich stets dem Spieler zuzuwenden, um ihm ständig ihre Informationen präsentieren zu können.

In unseren Evaluierungen mit anderen Entwicklern, die bereits Unity verwendet haben, konnten wir feststellen, dass die Einarbeitungszeit in das Werkzeug sehr gering ist. So war es den Entwicklern möglich, bereits innerhalb von 30 Minuten eine Simulation in einer vorgegebenen Umgebung zu erstellen. Die Prefabs, die wir in unserem Content-Generator-Paket mitliefern, haben hierbei einen entscheidenden Einfluss darauf gehabt, da es damit möglich ist, mit nur wenigen Aktionen das Spieler-Objekt, einen Pfad mit sieben Kontrollpunkten, sowie vier beispielhafte Informationsobjekte und den Content-Generator in die Unity-Szene einzufügen. Auch eine Informationsdatei für die Befüllung der Informationsobjekte mit multimedialen Inhalten war bereits im Content-Generator-Paket enthalten. Somit entstand auch hier kein zusätzlicher Aufwand und für zukünftige Projekte ist eine Datei vorhanden, die nur noch an die Simulationen angepasst werden muss. Diese Tatsache wurde von den Entwicklern sehr geschätzt. Es stellte sich heraus, dass die Erzeugung des Pfades intuitiv war. Speziell wurde die Schaltfläche im Inspektor zum Hinzufügen von zusätzlichen Wegpunkten häufig verwendet, um nicht manuell neue Spiel-Objekte erzeugen zu müssen, die als Wegpunkte fungieren. Allerdings konnten wir feststellen, dass die Interaktion mit den Kontrollpunkten des Pfades nicht ganz intuitiv erscheint. Die Testpersonen haben hier anfangs häufig versucht, die grünen Sphären direkt in der Szene zu verschieben, ohne sie vorher im Hierarchie-Fenster zu markieren. Für zukünftige Verbesserungen sollte hier eine Methode gefunden werden, um zu erkennen, wann ein Benutzer auf eine Sphäre klickt, um somit das Objekt im Hierarchie-Fenster automatisch zu markieren und auf diese Weise Interaktionen mit den Kontrollpunkten zu ermöglichen. Auch wurde eine Erweiterung des Pfadsystemes in Betracht bezogen. Für einige Simulationen könnte es besser sein, wenn die Möglichkeit besteht, zwischen dem Startpunkt des Spieler-Objektes und der ersten grünen Sphäre (an der das erste Informationsobjekt platziert wird) noch weitere Kontrollpunkte einfügen zu können, um nicht direkt mit den Informationsinhalten zu beginnen. Ähnlich verhält es sich am Ende des Pfades, wo weitere 'leere' Kontrollpunkte von Nutzen sein können. Unsere Evaluierungsergebnisse deuten jedoch darauf hin, dass das System ein geeignetes Werkzeug darstellt, um neue Simulationen effizient erstellen zu können. Die Testpersonen waren großteils begeistert, dass sie mit diesem Werkzeug in so kurzer Zeit eine Simulation erstellen und sich mithilfe einer 3D-Brille durch die virtuelle Szene bewegen konnten. Ebenfalls wurde in diesem Zusammenhang der abgerundete Pfad als gut empfunden, da er stark zur Cybersickness-Prävention beiträgt. Auch die unterschiedlichen Methoden, um

die Informationsobjekte mit multimedialen Inhalten zu füllen, wurde von den Entwicklern sehr geschätzt. Dennoch sehen wir Verbesserungsmöglichkeiten in diesem Werkzeug. Ein verändertes Kontrollpunktsystem könnte es uns ermöglichen, den Bereich, in dem sich die Informationsträger befinden, stärker einzugrenzen und somit zu Beginn und am Ende des Pfades mehr freien Raum zu besitzen.

Da wir mit dem Content-Generator ein Werkzeug besitzen mit dessen Hilfe sehr schnell eine Simulation mit Informationsobjekten erstellt werden kann, lässt sich davon eine sehr große Anzahl an Anwendungsmöglichkeiten ableiten. Sofern die jeweilige virtuelle Umgebung bereits erstellt wurde, könnten beispielsweise Tauchgänge in einigen hundert Metern Tiefe, oder Reisen zu fernen Planeten gemacht werden. Während den Simulationen können die Benutzer mehr über die Objekte in ihrer Umgebung erfahren, indem sie von Informationsobjekten begleitet werden. Auch weitere Simulationen, wie Gleitschirmfliegen, Base-Jumping, Fliegen mit einem Flügelanzug, sowie Museumsbesuche, Besichtigungen von Kulturerbe-Stätten und Ähnliches können mithilfe dieses Werkzeugs erstellt werden. Im Hinblick auf zukünftige Arbeiten könnte es von Nutzen sein, das Spieler-Objekt vom Pfad lösen zu können, oder den Pfad während der Simulation verändern zu können, um dem Benutzer mehr Freiraum zu bieten. Es würde sich anbieten, die Bewegungen des Benutzer mithilfe eines Bewegungs-Registrierungs-Geräts zu verfolgen und ihm so Interaktionen mit dem System zu ermöglichen (wie es bereits im Projekt *TU Fly into the Future* [Eid15b] geplant ist).

Abbildungsverzeichnis

Literaturverzeichnis

[Bar15] Oliver Barraza. Unity3d developer's guide to unreal engine 4, Letzter Zugriff: 31.07.2015. https://wiki.unrealengine.com/Unity3D_Developers_Guide_to_Unreal_Engine_4

[Bro15] Jon Brodkin. How unity3d became a game-development beast, Letzter Zugriff: 24.08.2015. http://insights.dice.com/2013/06/03/how-unity3d-become-a-game-development-beast/

[ea08a] A. L. Fuhrmann et al. Interactive content for presentations in virtual reality. *Proceedings of the ACM symposium on Virtual reality software and technology*, 2008.

[ea08b] Gaën Plancher et al. Virtual reality as a tool for assessing episodic memory. *Proceedings of the 2008 ACM symposium on Virtual reality software and technology*, 2008.

[ea08c] Grégory Wallet et al. Use of virtual reality for spatial knowledge transfer: effects of passive/active exploration mode in simple and complex routes for three different recall tasks. *Proceedings of the 2008 ACM symposium on Virtual reality software and technology*, 2008.

[ea09] Cem Yuksel et al. On the parameterization of catmull-rom curves. *SPM '09 2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling*, 2009.

[ea11] Kuang Yang et al. Study on the virtual natural landscape walkthrough by using unity 3d. *VR Innovation (ISVRI), 2011 IEEE International Symposium*, 2011.

[ea12a] Brian Cullen et al. The effects of audio on depth perception in s3d games. *Proceedings of the 7th Audio Mostly Conference: A Conference on Interaction with Sound*, 2012.

[ea12b] Norman G. Vinson et al. Cybersickness induced by desktop virtual reality. *GI '12 Proceedings of Graphics Interface 2012*, 2012.

[ea14a] Jean-Baptiste Barreau et al. Virtual reality tools for the west digital conservatory of archaelogical heritage. *Proceedings of the 2014 Virtual Reality International Conference*, 2014.

[ea14b] Simon Davis et al. A systematic review of cybersickness. *Proceedings of the 2014 Conference on Interactive Entertainment*, 2014.

[Eid15a] Horst Eidenberger. Virtual jump simulator, Letzter Zugriff: 05.09.2015. https://www.ims.tuwien.ac.at/projects/virtualjumpsimulator

[Eid15b] Horst Eidenberger. TU fly into the future, Letzter Zugriff: 28.09.2015. https://www.ims.tuwien.ac.at/projects/mpltufly

[EP14] Andrea Adami Eva Pietroni. Interacting with virtual reconstructions in museums: The etruscanning project. *Journal on Computing and Cultural Heritage (JOCCH) - Special Issue on Interacting with the Past*, 2014.

[Goh15] Katharina Gohr. *Stand und Entwicklungstendenzen im multisensorischen Marketing zur Inszenierung von Marken - eine kritische Analyse*. Letzter Zugriff: 12.09.2015.

[Hel15] David Helgason. Unity 1.0 is shipping, Letzter Zugriff: 24.08.2015. http://forum.unity3d.com/threads/unity-1-0-is-shipping.56/

[Ins15] Business Insider. The virtual reality report, Letzter Zugriff: 25.08.2015. http://uk.businessinsider.com/virtual-reality-headset-sales-explode-2015-7

[JJL00] Jr. Joseph J. LaViola. A discussion of cybersickness in virtual environments. *ACM SIGCHI Bulletin*, 2000.

[JL03] Ching-Kuang Shene John Lowther. Teaching b-splines is not difficult! *SIGCSE '03 Proceedings of the 34th SIGCSE technical symposium on Computer science education*, 2003.

[Kam15] Mike Kamermans. A primer on bézier curves, Letzter Zugriff: 05.07.2015. http://pomax.github.io/bezierinfo/

[May15] Austin Mayden. Unreal engine 4 vs. unity: Which game engine is best for you?, Letzter Zugriff: 31.07.2015. http://blog.digitaltutors.com/unreal-engine-4-vs-unity-game-engine-best/

[Meg15] MegaFiers. Megashapes, Letzter Zugriff: 30.07.2015. http://www.west-racing.com/mf/?page_id=2208

[Nor15] Erik Nordeus. Tutorial: How to create catmull-rom splines in unity, Letzter Zugriff: 05.07.2015. http://www.habrador.com/labs/catmull-rom-splines/

[Ocu15] Oculus. Oculus - developer center, Letzter Zugriff: 25.08.2015. https://developer.oculus.com/downloads/

[RP06] John K. Johnstone Ross Ptacek. Controlling the entire path of a virtual camera. *Proceedings of the 44th annual Southeast regional conference*, 2006.

[SH15] Susanne Buld Sonja Hoffmann, Hans-Peter Krüger. Vermeidung von simulator sickness anhand eines trainings zur gewöhnung an die fahrsimulation., Letzter Zugriff: 26.08.2015. http://www.psychologie.uni-wuerzburg.de/izvw/publikationen/index.php.de

[Tec15] Unity Technologies. Vr overview, Letzter Zugriff: 30.07.2015. http://docs.unity3d.com/Manual/VROverview.html

[Tul15] Herman Tulleken. Bézier curves for your games: A tutorial, Letzter Zugriff: 05.07.2015. http://devmag.org.za/2011/04/05/bzier-curves-a-tutorial/

[Uni15a] Unity. Unity public relations, Letzter Zugriff: 22.08.2015. https://unity3d.com/public-relations

[Uni15b] Unity. Unity - multiplatform, Letzter Zugriff: 24.08.2015. https://unity3d.com/unity/multiplatform

[Uni15c] Unity. What's new in unity 5.0, Letzter Zugriff: 24.08.2015. https://unity3d.com/unity/whats-new/unity-5.0

[Uni15d] Unity. Unity - the standard shader, Letzter Zugriff: 25.08.2015. https://unity3d.com/learn/tutorials/modules/beginner/5-tutorials/standard-shader

[Whi15] Thomas Whitehead. Weirdness: The cyberith virtualizer combines wii remote controls, oculus rift and a treadmill, Letzter Zugriff: 25.08.2015. http://www.nintendolife.com/news/2014/07/weirdness_the_cyberith _virtualizer_combines_wii_remote_controls_oculus_rift_and_a_treadmill

[Wik15] Wikipedia. Liste von spiel engines, Letzter Zugriff: 22.08.2015. https://de.wikipedia.org/wiki/Liste_von_Spiel-Engines

[YK14] Jie Jiang Yang Kuang. The research of making scenic wandering system based on unity 3d. *Electronics, Computer and Applications, 2014 IEEE Workshop*, 2014.