HOCHSCHULE ANHALT University of Applied Sciences

Bachelorarbeit

Fachbereich Informatik und Sprachen

Studiengang Angewandte Informatik - Digitale Medien und Spieleentwicklung

Vergleich der Umsetzung eines VR-Projektes zwischen Unity und der Unreal Engine 5

Eingereicht von

Jonas Bongartz

Matr. Nr.: 4069916

am 26. Juli 2023

an der Hochschule Anhalt

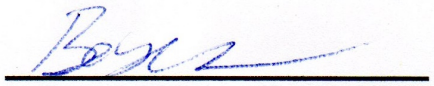
Erstprüfer: Prof. Dr. Stefan Schlechtweg

Zweitprüfer: Prof. Dr. Johannes Tümler

Selbständigkeitserklärung

Diese Arbeit mit dem Titel „Vergleich der Umsetzung eines VR-Projektes zwischen Unity und der Unreal Engine 5“ wurde von mir selbständig verfasst und in gleicher oder ähnlicher Fassung noch nicht in einem anderen Studiengang als Prüfungsleistung vorgelegt. Ich habe keine anderen als die angegebenen Hilfsmittel und Quellen, einschließlich der angegebenen oder beschriebenen Software, verwendet.

Köthen (Anhalt), den 22. Juli 2023



Jonas Bongartz

Gliederung

1. Einleitung

In der heutigen Gesellschaft hat sich Virtual Reality zu einem großen Themenfeld entwickelt und findet vermehrt Anwendung nicht nur in der Unterhaltungsbranche, sondern auch in Medizin, Architektur und anderen Bereichen. Für die Umsetzung von VR-Projekten werden Game Engines genutzt. Dabei wird häufig auf Unity gesetzt, da diese Engine einen umfangreichen VR-Support bietet und bei vielen Anwendern als einsteigerfreundlich bekannt ist. Jedoch existieren auch weitere Engines, die durchaus zu Unity konkurrenzfähig in diesem Feld sind. Eine davon ist die Unreal Engine. Diese hat mit der Veröffentlichung der Unreal Engine 5 ein breites Angebot an neuen Werkzeugen und Funktionen bereitgestellt, mit denen das Umsetzen einer Vielzahl von Anwendungen einfacher gestaltet wird. Im Bereich Virtual Reality bleibt jedoch Unity ein Standardwerkzeug zur Erstellung von VR-Anwendungen [1] und das, obwohl die Unreal Engine 5 auch VR-Entwicklung unterstützt. Dabei stellt sich die Frage, ob Unity tatsächlich besser für die Umsetzung von VR-Projekten geeignet ist oder ob die Unreal Engine 5 eine bessere Option darstellt.

1.1. Ziel

Das Ziel dieser Arbeit ist, mit Hilfe eines repräsentativen Projekts einen Vergleich zwischen den beiden Engines anzustellen. Bei diesem Projekt handelt es sich um eine Anwendung im Stile des bekannten Spiels „Dosenwerfen“. Die Anwendung wird zunächst vom Autor implementiert, anschließend kritisch analysiert und danach im Zuge eines Experimentes mit Studierenden, die zuvor wenig bis gar keine Erfahrung mit Game Engines gesammelt haben, implementiert und bewertet. Dabei richtet sich der analytische Vergleich an eine Zielgruppe von fortgeschrittenen xR-Entwicklern. In dem Experiment sollen Erkenntnisse aus der Sicht von xR-Einsteigern gesammelt werden. Am Ende entsteht daraus eine Erkenntnissammlung, welche Engine wofür besser geeignet ist, sowohl für Einsteiger als auch für Fortgeschrittene.

1.2. Aufbau

Zunächst werden Grundbegriffe zu den Themen „xR“ und „Game Engine“ geklärt, um ein Verständnis für die Thematik aufzubauen. Anschließend geht es um das für den analytischen Vergleich vom Autor implementierte Projekt. Dabei soll dieser Teil erklären, wie bei der Implementierung des Projektes in den jeweiligen Engines vorgegangen wurde. Auf Grundlage dessen wird ein analytischer Vergleich durchgeführt, indem Kriterien erarbeitet und anschließend angewendet werden. Es folgt die Beleuchtung der einzelnen Aspekte des im Rahmen dieser Arbeit durchgeführten Experimentes. Hier wird darauf eingegangen, wie dieses Experiment umgesetzt wurde und welche Ergebnisse es erzielt hat. Im letzten Teil werden die Ergebnisse abschließend zusammengeführt und in einer grafischen Form dargestellt.

2. Grundlagen und Stand der Technik

Um das Verständnis für die in dieser Arbeit behandelten Themen und Begriffe aufzubauen, werden sie in diesem Kapitel aufgeschlüsselt und entsprechend der Relevanz näher erklärt.

2.1. xR

xR, oder „Extended Reality“, ist ein Oberbegriff, der verschiedene Formen von computergenerierter Realität umfasst, darunter Virtual Reality (VR), Augmented Reality (AR) und Mixed Reality (MR) [2]. xR erweitert unsere Wahrnehmung und ermöglicht es uns, digitale Informationen in die physische Welt einzubetten oder uns vollständig in virtuelle Umgebungen eintauchen zu lassen.

Es beschreibt ein Spektrum von Realitätsmodi, die von der vollständigen Virtualität bis zur vollständigen Realität reichen. xR-Technologien bieten den Benutzern die Möglichkeit, mit virtuellen Objekten und Informationen zu interagieren, um neue Anwendungen in Bereichen wie Bildung, Gesundheitswesen, Architektur und Unterhaltung zu schaffen [3].

2.1.1. AR

Augmented Reality (AR) ergänzt die reale Welt um digitale Inhalte, indem sie u. a. über Smartphones, Tablets oder spezielle AR-Brillen visuelle Informationen überlagert. Eine Studie von Billinghurst und Kato (2002) in der Zeitschrift „Communications of the ACM“ hebt hervor, dass AR es den Benutzern ermöglicht, Informationen nahtlos in ihre physische Umgebung zu integrieren. AR findet Anwendung in Bereichen wie der industriellen Fertigung, dem Einzelhandel und der Erweiterung des Lernens [4]. Ein bekanntes Beispiel dafür ist die „IKEA Place App“, mit der Möbel per Smartphone als 3D-Modelle im Raum platziert werden können, um vorab einschätzen zu können, ob das Möbelstück in das Zimmer passt [5].

2.1.2. MR

Mixed Reality (MR) kombiniert Elemente von VR und AR, um eine nahtlose Interaktion zwischen der realen und der virtuellen Welt zu schaffen. Eine Veröffentlichung von Milgram und Kishino (1994) in der Zeitschrift „Presence: Teleoperators and Virtual Environments“ erklärt, dass MR es den Benutzern ermöglicht, digitale Objekte in die reale Welt einzufügen und gleichzeitig eine Interaktion mit ihnen zu ermöglichen. MR wird in verschiedenen Bereichen wie Design, Architektur und kollaborativer Arbeit eingesetzt, letzteres vor allem vertreten mit der Konferenz-Software „Dynamics 365 Remote Assist“ von Microsoft für die HoloLens 2 [6].

2.1.3. VR

Virtual Reality (VR) ist eine Technologie, die es Benutzern ermöglicht, in eine immersive und interaktive virtuelle Umgebung einzutauchen. VR erzeugt eine simulierte Realität, die visuelle, auditive und manchmal haptische Reize nutzt, um eine Erfahrung zu schaffen, die der physischen Realität ähnelt oder sie sogar übertrifft [7]. Da Virtual Reality das Hauptthema dieser Arbeit bildet, wird sie im Folgenden näher erläutert.

Die Grundlage von VR besteht aus drei Grundkonzepten, die erfüllt sein müssen [8]:

- Immersion
- Interaktion
- Presence Flow

Um das zu erreichen, wird spezielle Hard- und Software verwendet. Benutzer tragen in der Regel ein Headset, das aus einer VR-Brille mit integrierten Bildschirmen besteht, die ein immersives visuelles Erlebnis bieten. Die Bewegungen des Benutzers werden dabei von Sensoren erfasst und an die Software weitergeleitet, um die virtuelle Umgebung entsprechend anzupassen. Alternativ können auch andere Hardwaresetups genutzt werden [9].

Die immersiven visuellen Effekte von VR werden durch stereoskopische Displays erzeugt, die zwei leicht unterschiedliche Bilder anzeigen, um einen räumlichen Eindruck zu erzeugen. Die Auflösung der Displays und die Bildwiederholungsrate sind entscheidend für die Qualität und das Eintauchen in die virtuelle Umgebung. Hochauflösende Displays und hohe Bildwiederholungsraten sind erforderlich, um ein flüssiges und realistisches visuelles Erlebnis zu bieten.

Um eine räumliche Wahrnehmung zu erzeugen, verwenden VR-Systeme Tracking-Technologien. Dies erfolgt durch Sensoren, die die Position und Bewegung des Benutzers verfolgen, wie zum Beispiel Infrarotsensoren, Beschleunigungsmesser oder Gyroskope. Durch präzises Tracking wird die virtuelle Umgebung entsprechend den Kopf- und Körperbewegungen des Benutzers angepasst [10, 11, 12].

2.1.4. Anwendungsbereiche von VR

Virtual Reality (VR) bietet eine Vielzahl von Anwendungsmöglichkeiten in verschiedenen Bereichen, von der Unterhaltung bis hin zur Bildung und Medizin. Die immersive und interaktive Natur von VR eröffnet neue Möglichkeiten, um virtuelle Umgebungen zu erkunden und realitätsnahe Erfahrungen zu schaffen. Im Folgenden werden einige Anwendungsgebiete von VR vorgestellt:

1. **Unterhaltung und Gaming:**
   VR hat die Art und Weise, wie wir Spiele spielen und Unterhaltung erleben, revolutioniert. Durch das Eintauchen in virtuelle Welten können Benutzer ein intensives Spielerlebnis genießen. Studien haben gezeigt, dass VR eine erhöhte Immersion und ein gesteigertes Engagement bietet, was zu einer intensiveren Spielerfahrung führt [13].

2. **Training und Simulation:**
   Auch für Training und Simulationen wird VR in verschiedenen Bereichen eingesetzt, wie z. B. in der Medizin, im Militär und in der Luftfahrt. Durch VR können realitätsnahe Szenarien geschaffen werden, um praktische Fähigkeiten zu entwickeln und kritische Situationen zu üben. VR-basiertes Training kann sehr effektiv sein und zu einer verbesserten Leistung und höherem Lernerfolg führen, was Studien belegen [14].

3. **Architektur und Design:**
   VR ermöglicht es Architekten und Designern, virtuelle Modelle von Gebäuden und Produkten zu erstellen und zu visualisieren. Dadurch können sie potenzielle Designs vor der tatsächlichen Umsetzung erkunden und Kunden eine immersive Erfahrung bieten. Architekturvisualisierung durch VR bietet eine verbesserte Raumwahrnehmung und unterstützt laut Studien effektiv bei der Entscheidungsfindung [15].

4. **Gesundheitswesen und Therapie:**
   Ein besonderes Anwendungsgebiet von Virtual Reality ist das Gesundheitswesen. Dort wird es insbesondere in der Schmerztherapie, bei der Behandlung von Angststörungen und der Rehabilitation eingesetzt. Durch das Eintauchen in angenehme virtuelle Umgebungen können Schmerzen gelindert und Angstzustände reduziert werden. Auch hier zeigen Studien auf, dass VR-gestützte Therapien effektiv sind und positive Ergebnisse bei verschiedenen medizinischen Anwendungen erzielen [16].

2.2. Game Engine

Eine Game Engine ist eine Softwareplattform, die zur Entwicklung und Erstellung von multimedialen Anwendungen verwendet wird. Sie stellt Entwicklern Werkzeuge und Funktionen in einer Laufzeitumgebung als Editor zur Verfügung, um Anwendungen zu entwerfen, zu erstellen, zu modellieren und zu programmieren. Sie besteht aus einer Sammlung von Bibliotheken, Modulen und Tools, die die Entwicklung erleichtern.

Die Architektur einer Game Engine setzt sich aus verschiedenen Komponenten zusammen. Eine davon ist die Grafik-Engine, die für die Darstellung von 2D- oder 3D-Grafiken zuständig ist. Diese Grafik-Engine kann hochentwickelte Rendering-Techniken wie Beleuchtung, Schattierung und Effekte bieten, um realistische und beeindruckende visuelle Erfahrungen zu erzeugen. Die Physik-Engine ermöglicht die Simulation von physikalischen Kräften und deren Verhalten in den Anwendungen. Dadurch können Objekte sich realistisch bewegen, kollidieren und reagieren. Die Audio-Engine, die für die Wiedergabe von Soundeffekten, Hintergrundmusik und Dialogen verantwortlich ist, unterstützt die Integration verschiedener Audioformate und bietet Tools zur Klangbearbeitung und -steuerung.

Zusätzlich bieten Game Engines Tools für die KI-Programmierung, Animation, Kollisionserkennung, Benutzerinteraktion, Netzwerkfunktionen und mehr. Sie ermöglichen außerdem Entwicklern, Anwendungen plattformübergreifend zu erstellen, was bedeutet, dass sie auf verschiedenen Betriebssystemen und Geräten laufen können, wie z.B. PCs, Konsolen, Smartphones und Tablets [17]. Game Engines können verschiedenartig konstruiert sein. Sie können speziell für eine ganz bestimmte Anwendung oder für eine Vielfalt von Anwendungen konzipiert sein. Ein Beispiel für eine spezialisierte Game Engine ist die im MMORPG Final Fantasy 14 genutzte „Luminous Engine“. Als „All-Purpose“ Engines sind vor allem Unity, Unreal Engine, CryEngine und Godot zu nennen, wobei folgend auf die beiden bekanntesten Engines eingegangen wird: Unity und Unreal Engine. Diese sind nicht nur Teil der weiteren Arbeit, sondern auch in den Communities und der Industrie die am häufigsten genutzten Third-Party Game Engines. Dabei ist Unity vor allem durch seine Einsteigerfreundlichkeit bekannt geworden, während die Unreal Engine, vor allem mit der Version 5, durch seine fortschrittlichen Features und extrem performante, aber hochqualitative, Grafik überzeugt.

Eine Übersicht über die anderen Engines findet sich in der Arbeit „Identification and evaluation of alternatives to Unity which are not under U.S. Sanction lists“ von Nipun Sharma [18].

2.2.1. Unity

Unity ist eine Game Engine, die von Unity Technologies entwickelt wurde und von zahlreichen Entwicklern weltweit genutzt wird [19]. Ihre Geschichte begann im Jahr 2002, als der dänische Programmierer David Helgason zusammen mit Nicholas Francis und Joachim Ante die Firma gründete. Ursprünglich als Tool für die Entwicklung von Spielen auf Apple-Geräten gedacht, erweiterte sich der Anwendungsbereich von Unity schnell auf andere Plattformen wie Windows, Android und Konsolen [20]. Sie besteht hauptsächlich aus folgenden Komponenten:

- **Editor:** Der Unity Editor ist die Entwicklungsplattform, die Tools und Funktionen für die Szenenerstellung, den Asset-Import, die Skripterstellung und das Debugging bietet. In Abbildung 2.1 kann die Benutzeroberfläche des Unity Editors eingesehen werden.

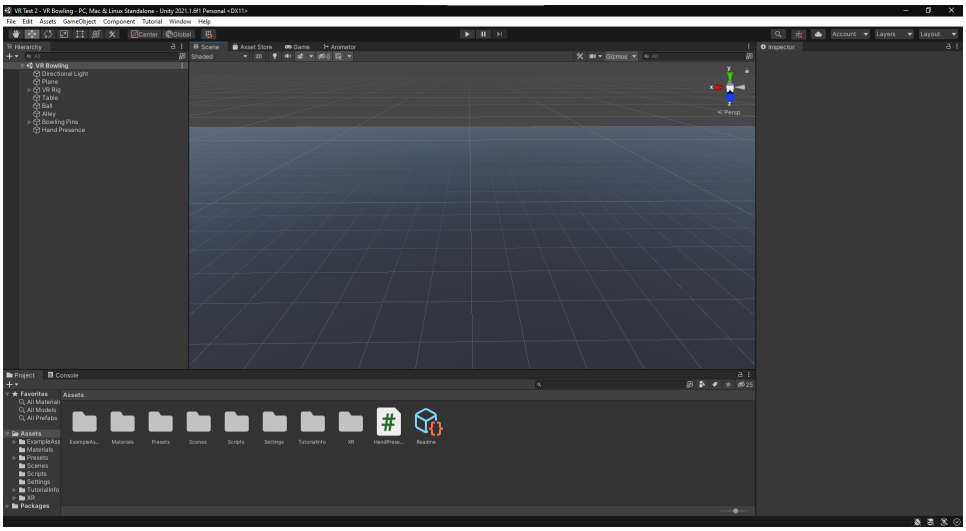


Abbildung 2.1.: Der Unity Editor

- **Skripting-System:** Entwickler können Skripte in C# schreiben, um das Verhalten von Objekten zu steuern und die Spiellogik zu implementieren.

- **Rendering-Engine:** Unity verfügt über eine leistungsstarke Rendering-Engine, die fortschrittliche Grafiktechnologien unterstützt. Zu ihren Funktionen gehören unter anderem physikalisch basierte Rendering-Techniken, beeindruckende Beleuchtungseffekte, Partikelsimulationen und die Integration von hochwertigen Effekten wie Bloom, Tiefenunschärfe und Bewegungsunschärfe.
- **Physik-Engine:** Unity verwendet eine robuste Physik-Engine, die realistische Simulationen von Objekten, Kollisionen und Bewegungen ermöglicht. Das Ergebnis sind realistische Interaktionen zwischen Objekten und Umgebungen.
- **Audio-Engine:** Unity bietet eine leistungsstarke Audio-Engine, mit der hochwertige Soundeffekte, Musik und Sprachausgabe in ihren Spielen integriert werden. Die Engine unterstützt fortschrittliche Audioeffekte wie 3D-Sound und Reverb.

Auch die Integration von Plug-ins wird von der Engine unterstützt, um den Funktionsumfang zu erweitern. Diese Plug-ins können von Drittanbietern entwickelt werden und bieten zusätzliche Funktionen und Dienste wie Analytics, Werbung oder soziale Integration. Die Entwicklung von Projekten kann dabei plattformübergreifend geschehen, was bedeutet, dass Spiele für verschiedene Plattformen wie PC, Konsolen, Mobilgeräte und sogar Virtual Reality (VR) und Augmented Reality (AR) entwickelt werden können. Dadurch haben Entwickler die Möglichkeit, ihre Spiele einem breiten Publikum zugänglich zu machen [21]. Unity ist kostenlos nutzbar, weshalb es ideal für Einsteiger geeignet ist. Mit Anwendungen, die mit der kostenlosen Lizenz entwickelt wurden, dürfen nur „[...] Einzelpersonen, Hobbyisten und kleine Organisationen mit einem Umsatz oder Spenden von weniger als 100.000 US-Dollar in den letzten 12 Monaten [22]“ Umsatz machen. Alle Organisationen, die über dieser Grenze liegen, müssen sich zur Monetarisierung ihrer Anwendung eine kostenpflichtige Lizenz kaufen [23].

2.2.2. Unreal Engine

Die Unreal Engine ist eine Game Engine der Firma Epic Games. Erstmals 1998 von Tim Sweeney, dem Gründer von Epic Games, veröffentlicht, hat sie seitdem eine beeindruckende Entwicklung durchlaufen. Ursprünglich entwickelt für den Ego-Shooter „Unreal“, wurde die Engine kontinuierlich weiterentwickelt und fand Verwendung in vielen erfolgreichen Spielen, darunter „Unreal Tournament“, „Gears of War“ und „Fortnite“. Die Unreal Engine besteht hauptsächlich aus folgenden Komponenten:

- **Unreal Editor:** Der Unreal Editor ist die zentrale Entwicklungsumgebung der Unreal Engine. Hier können Entwickler ihre Anwendungen entwerfen, Szenen erstellen, Assets importieren und bearbeiten, Materialien definieren, Animationen erstellen und vieles mehr. Der Editor bietet eine intuitive Benutzeroberfläche und umfangreiche Tools, um den Entwicklungsprozess zu unterstützen. Die Benutzeroberfläche des Editors ist in Abbildung 2.2 dargestellt.

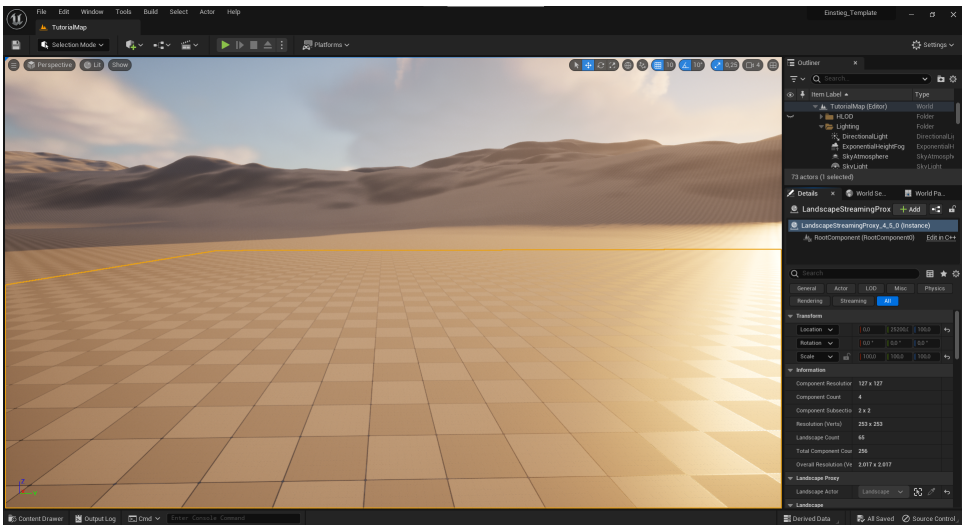


Abbildung 2.2.: Der Unreal Engine 5 Editor

- **Blueprint Visual Scripting:** Die Unreal Engine unterstützt das sogenannte Blueprint Visual Scripting, das es Entwicklern ermöglicht, das Verhalten von Objekten und die Anwendungslogik ohne umfangreiche Programmierkenntnisse zu steuern. Durch das Erstellen von visuellen Skripten in einem Node-basierten System können Entwickler komplexe Interaktionen und Abläufe in ihren Anwendungen erstellen.

- **Rendering-Engine:** Die Rendering-Engine in Unreal Engine 5 ist wie in Unity für die Erzeugung von hochwertigen Grafiken und visuellen Effekten verantwortlich. Sie arbeitet eng mit der Hardware des Computers oder der Spielekonsole zusammen, um eine realistische Darstellung von 3D-Szenen zu ermöglichen. Die Unreal Engine 5 verwendet dabei fortschrittliche Rendering-Techniken wie Raytracing, um Licht und Schatten in Echtzeit zu berechnen und fotorealistische Ergebnisse zu erzielen. Darüber hinaus bietet die Rendering-Engine leistungsstarke Werkzeuge zur Materialbearbeitung, Partikelerzeugung und Post-Processing-Effekten, um die visuelle Qualität und den Stil einer Anwendung anzupassen.
- **Physik-Engine:** Äquivalent zu Unity ist auch in der Unreal Engine 5 die Physik-Engine dafür zuständig, die physikalischen Eigenschaften und Interaktionen von Objekten in der Anwendung zu simulieren. Sie berücksichtigt Kräfte wie Schwerkraft, Reibung, Kollisionen und Bewegungen, um realistisches Verhalten von Charakteren, Fahrzeugen, Objekten und Umgebungen zu erzeugen. Sie unterstützt auch komplexe Physikeffekte wie zerstörbare Umgebungen, Flüssigkeiten und Stoffsimulation.
- **Audio-Engine:** Die Audio-Engine in Unreal Engine 5 ist für die Erzeugung und Wiedergabe von hochwertigem Klang und Audioeffekten in Anwendungen zuständig und erfüllt damit die gleichen Aufgaben wie die Audio-Engine von Unity. Sie ermöglicht es Entwicklern, eine immersivere Erfahrung zu schaffen, indem sie realistischen und räumlichen Klang erzeugt. Die Audio-Engine unterstützt auch fortschrittliche Techniken wie Echtzeit-Audio-Simulation, Umgebungsgeräusche, 3D-Positionierung von Klängen und Doppler-Effekte. Dadurch können Entwickler den Klang von Charakteren, Umgebungen und Ereignissen feinabstimmen, um die Atmosphäre und den Realismus der Anwendung zu verbessern. Darüber hinaus bietet sie Werkzeuge zur Musikkomposition und -integration, um den Soundtrack einer Anwendung zu erstellen und nahtlos in die Szenen zu integrieren.

Dabei können die Funktionen der Engine durch Plugins, sowohl von Epic Games als auch aus der Community, erweitert werden.
Darüber hinaus hat die Unreal Engine ihren Einfluss über die Spieleentwicklung hinaus erweitert und wird vermehrt in der Filmindustrie eingesetzt. Dank ihrer Echtzeit-Rendering-Fähigkeiten und ihrer visuellen Qualität wird sie für die Erstellung von Visual Effects und virtuellen Sets in Filmen und Fernsehproduktionen eingesetzt [24].
Die Unreal Engine ist kostenlos nutzbar mit der Standard Lizenz von Epic Games und somit auch leicht zugänglich für Einsteiger. Die Standard Lizenz erlaubt auch den Vertrieb von entwickelter Software, solange man mit Hilfe eines Formulars Epic Games über die Veröffentlichung der Anwendung informiert. Danach fallen keinerlei Lizenzgebühren an, solange die Anwendung noch nicht mehr als eine Million Dollar Bruttoeinnahmen generiert hat. Hat man diese Schwelle überschritten, müssen auf alle weiteren Einnahmen 5% Lizenzgebühren an Epic Games gezahlt werden [25].

3. Implementierung der Projekte in den Engines

In diesem Kapitel wird die Umsetzung eines repräsentativen Projektes durch den Autor behandelt. Bei dem Projekt handelt es sich um ein Dosenwerfen-Minispiel, das zur Erlangung von Erkenntnissen in der Umsetzung von VR-Projekten in den jeweiligen Engines dient und die Grundlage für die im Kapitel 4 durchgeführte Analyse bildet. Um eine Vergleichbarkeit zwischen den Engines herzustellen, wird in beiden das gleiche Projekt umgesetzt. In diesem Spiel gibt es einen Ball, der für den Spieler auf einem Tisch platziert wurde. Der Spieler kann den Ball aufnehmen und auf eine weiter entfernte Dosenpyramide werfen, die wiederum durch Kollision mit dem Ball mit Hilfe von Physiksimulation umgeworfen werden kann. Nachdem der Spieler einmal geworfen und entweder die Dosen getroffen hat oder nicht, können per Knopfdruck sowohl der Ball als auch die Pyramide zurückgesetzt werden, sodass noch einmal gespielt werden kann. Während des Spiels ist es dem Spieler möglich, sich über das Spielfeld sowohl physisch durch körperliche Bewegungen z. B. der Arme oder der Beine als auch nicht-physisch durch das Betätigen der Joysticks und Buttons der Controller zu bewegen, um sich für den Wurf zu positionieren. Auch ein UI steht zur Verfügung, welches jederzeit per Knopfdruck aufgerufen werden kann und sich mit dem Controller des Spielers bewegt. Dort ist hauptsächlich das Ändern der Bewegungsoptionen möglich. Dabei gibt es für den Spieler keinerlei Score-System, da in dieser Arbeit eher die VR-Funktionalitäten betrachtet werden sollen. Die Inhalte des Projektes wurden gewählt, weil beim Dosenwerfen viele typische Grundelemente von VR-Anwendungen benötigt werden. Dazu zählen z. B. das Greifen von Objekten, das Werfen von Objekten, die Objektphysik und die Bewegung der Spielerfiguren.

3.1. Anforderungen an die Projekte

Bevor die Anforderungen an das Projekt im Einzelnen erläutert werden, müssen einige Grundbedingungen vor der Umsetzung festgelegt werden, um eine Vergleichbarkeit der beiden Projekte zu gewährleisten:

1. Das Projekt muss sowohl in Unity als auch in der Unreal Engine 5 mit den gleichen Projektanforderungen umgesetzt werden.
2. Das Projekt soll in beiden Engines mit dem VR-Template der Engines initialisiert werden.
3. Alle vorimplementierten Features, die durch Template, Engine und SDKs zur Verfügung gestellt werden, dürfen genutzt werden und müssen nicht eigenhändig implementiert werden.

Da das Projekt alle Grundelemente von VR-Anwendungen abdecken soll, werden folgende Bedingungen daran gestellt:

1. Der Spieler muss in der Lage sein, sich zu bewegen. Dies beinhaltet:
   - das nicht-physische Drehen mit Hilfe der Joysticks. Es soll sowohl die Möglichkeit geben, sich per *Snap Turn*[1] als auch per *Continuous Turn*[2] zu drehen.
   - das nicht-physische Fortbewegen mit Hilfe der Joysticks. Es soll sowohl die Möglichkeit geben, sich per *Teleport*[3] fortzubewegen als auch per *Continuous Locomotion*[4].
   - das physische Bewegen der Arme bzw. Controller Tracking.
2. Der Spieler muss in der Lage sein, zu jeder Zeit einen Ball greifen zu können.
3. Der Spieler muss in der Lage sein, diesen Ball auf eine Dosenpyramide zu werfen und diese umwerfen zu können.

[1] Drehungsart, die bei einer Richtungseingabe durch einen Joystick o. ä. eine einmalige Drehung des Spielers vornimmt
[2] Drehungsart, die bei kontinuierlicher Eingabe einer Richtung durch Joystick o. ä. solange den Spieler in die gewünschte Richtung dreht, bis keine Richtungseingabe mehr erfolgt
[3] Bewegungsart, bei der der Spieler direkt an eine von ihm gewählte Stelle transferiert wird
[4] Bewegungsart, bei der der Spieler sich kontinuierlich in eine von ihm gewählte Richtung bewegt

4. Der Spieler muss in der Lage sein, die Dosenpyramide sowie den Ball für eine weitere Spielrunde zurücksetzen zu können.

Dabei wird auf die ästhetischen Aspekte des Spiels nicht geachtet, da nur die technischen Aspekte für die spätere Analyse von Bedeutung sind.

3.2. Umsetzung der Projekte

Die in diesem Kapitel beschriebenen Umsetzungen des Projektes werden vom Autor implementiert und stellen die Sichtweise eines erfahrenen Entwicklers dar, der sowohl sieben Jahre Erfahrungen im Programmieren als auch zwei Jahre Erfahrung im Umgang mit Game Engines besitzt. Diese Projekte dienen als Grundlage für die spätere Analyse und als Vorlage für die Projekte, welche in einem späteren Experiment von Studierenden implementiert werden. Dabei wurden für die Umsetzungen folgende Engine-Versionen verwendet:

- Unity: Version 2020.3.15f2
- Unreal Engine: Version 5.1.1

Diese Versionen wurden ausgewählt, da sie grob den Versionen der Engines auf den im späteren Experiment von den Studierenden genutzten Rechnern entsprechen. Dadurch können die Erfahrungen aus diesen Projekten mit den Erfahrungen der Teilnehmer des Experimentes besser verglichen werden. Als VR-Hardware stand die Oculus Rift S von Meta zur Verfügung.

3.2.1. Umsetzung in Unity

Initialisierung

Zur Initialisierung des Projektes wurde das *VR-Core Template* von Unity ausgewählt. Das erlaubt es, bereits mit folgenden Features zu starten:

- *OpenXR-Plugin*, das generell VR-Entwicklung in Unity ermöglicht
- Headset und Controller Tracking durch *Tracked Pose Driver*.

Diese bieten eine gute Grundlage für den Start in die VR-Entwicklung, müssen aber erweitert werden, um das Spiel vollständig umzusetzen. Dazu fehlen noch folgende Grundfeatures:

- Greifen von Objekten in der virtuellen Welt mit den Controllern
- Locomotion System für das nicht-physische Laufen und Drehen des Spielercharakters.

Sobald der Editor sich öffnet, fordert ein Prompt dazu auf, die Plattformen auszuwählen, für die das Spiel später verfügbar gemacht werden soll. Dabei entscheidet man, welche VR-Hardware mit dem Spiel kompatibel ist. Folgende Optionen stehen zur Auswahl:

- Oculus
- Magic Leap Zero Iteration
- Microsoft Mixed Reality
- OpenXR
- Unity Mock HMD.

In diesem Fall wird sich für Oculus entschieden, da die Zielhardware die für diese Umsetzung verwendete Oculus Rift S ist.

SDK-Setup

Um Interaktionen zu ermöglichen, wird zusätzlich aus dem Package Manager das Plugin „XR Interaction Toolkit“ installiert, das im weiteren Verlauf der Arbeit mit XRTK abgekürzt wird. Dieses SDK erlaubt, komplexe Interaktionen des Spielers mit der virtuellen Umgebung durch vorgefertigte Komponenten zu etablieren. Damit diese Interaktionen erstellt werden können, benötigt das XRTK zwei grundlegende Komponenten:

1. *XR Origin* - Standort des Spielers in der virtuellen Welt, basierend auf den Trackingdaten des Spielers in der realen Welt
2. *XR Interaction Manager* - Basis für jedes Interaktionssystem des XRTKs.

Um die Controller im Spiel abzubilden, muss für jede Hand je ein Objekt der Hierarchie hinzugefügt werden. Diese Objekte erhalten anschließend folgende Komponenten:

1. *Device-Based XR Controller* - Grundkomponente für das Auslesen von Trackingdaten und Input auf dem jeweiligen Controller, wobei jeder Controller eine eigene Komponente benötigt.
2. *XR Ray Interactor* - ermöglicht Interaktionen mit dem Controller, z. B. das Greifen von Objekten oder Teleportieren über eine einstellbare Distanz.
3. *XR Line Visualizer* - rendert eine Linie vom Controller bis zur maximalen Distanz des *XR Ray Interactor*, um die Interaktionsreichweite zu visualisieren.

Szenenaufbau und interagierbare Elemente

Nach Beendigung des Setups des SDKs wird die Szene grundlegend aufgebaut, um spätere Features wie das Locomotion-System besser testen zu können. Eine Ebene wird erstellt, auf der zwei Würfel seitlich lang skaliert und parallel zueinander aufgestellt werden. Diese sollen die Tische für Ball und Dosenpyramide repräsentieren. Die Erstellung des Balls und der Dosenpyramide erfolgt dabei initial als Prefab. Der Ball kann mit Hilfe einer Kugel erstellt werden, dessen Scale auf 0.1 auf allen Achsen gesetzt und mit der Komponente *XR Grab Interactable* versehen wird, um den Ball für den Spieler greifbar zu machen. Die Dosenpyramide besteht aus sechs übereinander gestapelten Dosen-Prefabs. Aus einem Zylinder konstruiert entstehen die Dosen-Modelle, deren Scale bei X und Z auf 0.2 und bei Y auf 0.1 gesetzt wurden. Sowohl der Ball als auch die Dosen sind mit *Collidern* und *RigidBodies* ausgestattet, um physikalische Interaktionen zwischen diesen beiden zu ermöglichen. In Abbildung 3.1 ist die vollständige Szene nach Abschluss der genannten Schritte zu sehen. Abbildung 3.2 zeigt abstrakt die hierarchische Struktur der Objekte in der Szene. Dabei wurden nur die Objekte in die Grafik aufgenommen, die explizit für das Projekt erstellt wurden.

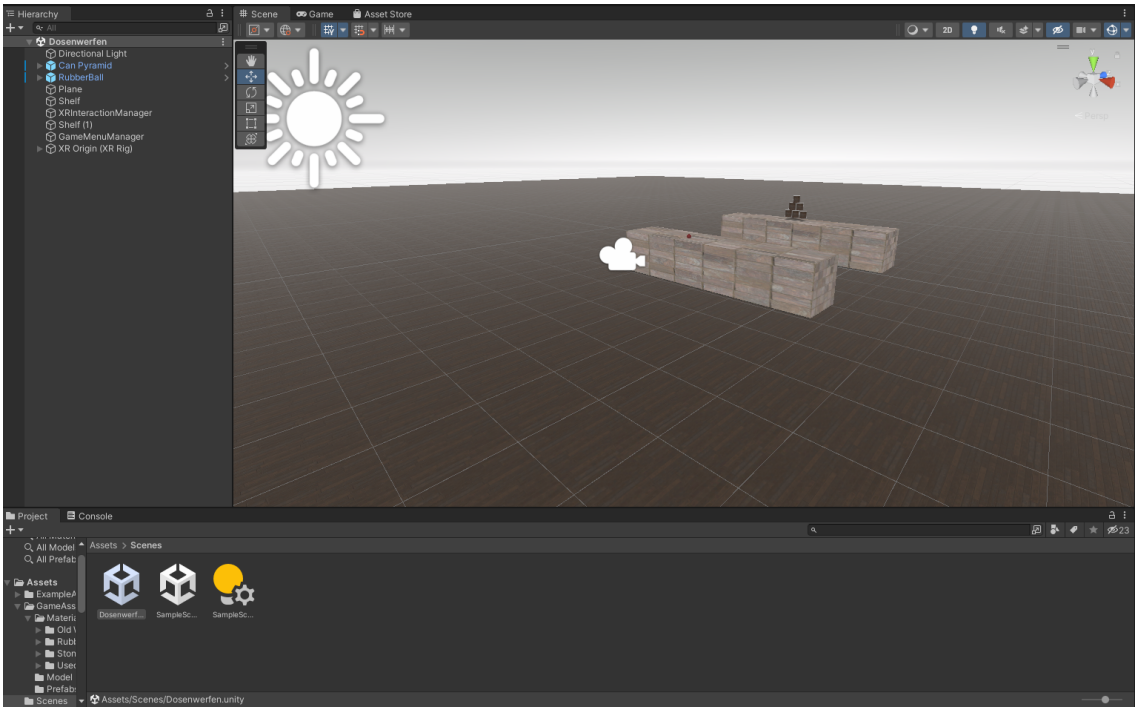


Abbildung 3.1.: Aufbau der Szene in Unity

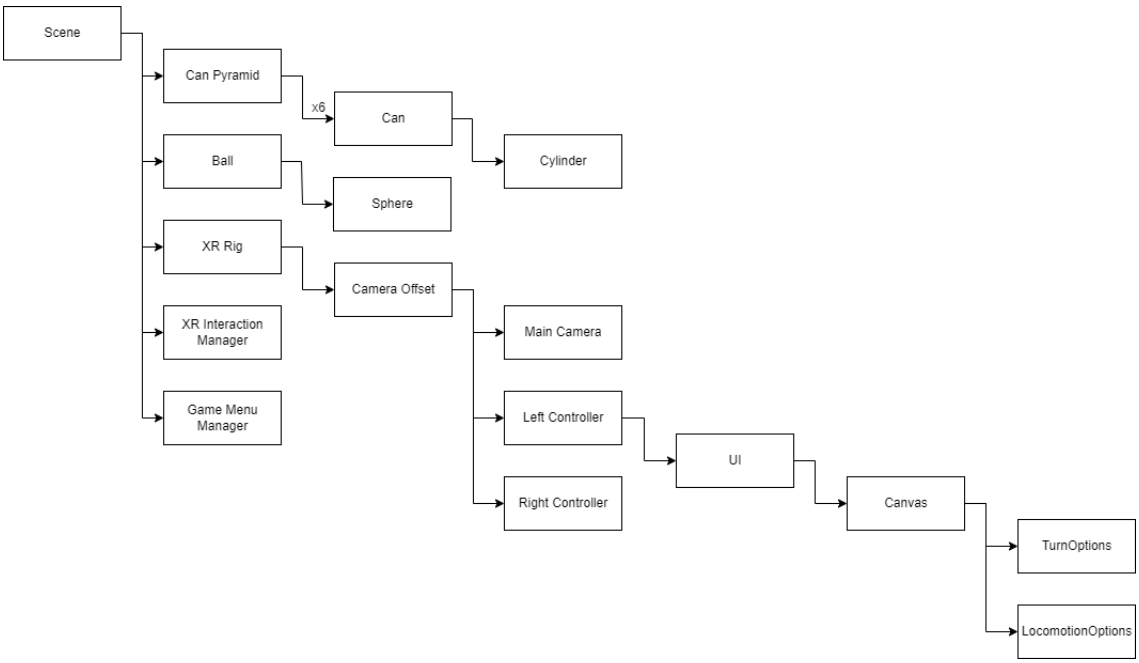


Abbildung 3.2.: Grafische Darstellung der Objekt-Hierarchie

Bewegung

Nach Schaffung der Grundlagen kann das Locomotion-System mit Hilfe einer *Locomotion-System-Komponente* des XRTKs, die die Grundlage für alle Bewegungsarten bildet, umgesetzt werden. Das Drehen des Spielers kann durch die Komponenten *Device-based Snap Turn Provider* und *Device-based Continuous Turn Provider* realisiert werden, die auf den rechten Controller gelegt und per Joystick gesteuert werden. Um Teleportationsfortbewegung umzusetzen, müssen der Grundebene der Szene eine *Teleportation Area* und dem Spieler ein *Teleportation Provider* hinzugefügt werden. Für die kontinuierliche Fortbewegung ist schließlich nur noch ein *Device-based Continuous Move Provider* nötig. All diese Skript-Komponenten wurden auf dem XRRig-Objekt platziert, da sie gegenseitige Referenz zueinander benötigen. In der Abbildung 3.3 ist die Liste an Skript-Komponenten grafisch dargestellt. Die Abhängigkeiten der Skriptkomponenten untereinander ist in Abbildung 3.4 zu sehen.

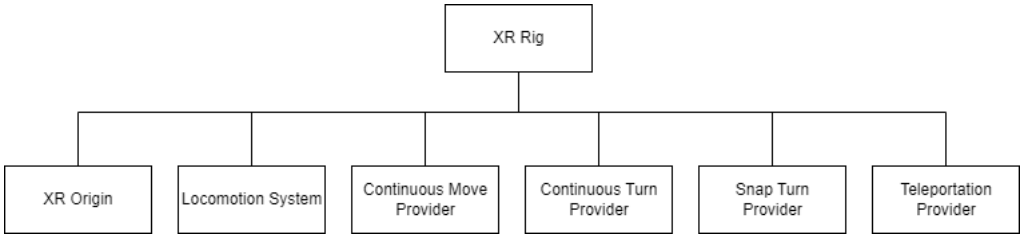


Abbildung 3.3.: Grafische Darstellung der Skriptkomponenten des XRRig-Objektes

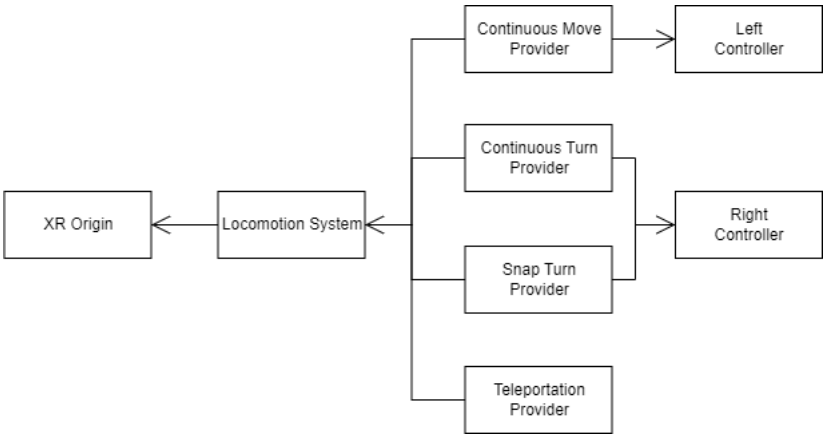


Abbildung 3.4.: Abhängigkeiten der Skriptkomponenten des XRRig-Objektes

User Interface

Um zwischen den einzelnen Bewegungsoptionen wechseln zu können, wird ein User Interface, weiterhin als UI bezeichnet, entwickelt. Dieses besteht im Wesentlichen aus zwei Dropdown-Menüs, die jeweils einmal die Drehoptionen und die Bewegungsoptionen anzeigen. Das UI wird hierarchisch unter den linken Controller platziert, wodurch es im Spiel fest damit verankert ist und sich mit dem Controller bewegt. Das Ein- und Ausschalten des UIs geschieht per Knopfdruck durch die eigens erstellte Skript-Komponente *MenuToggle* (siehe Listing 1). Dies ermöglicht eine bessere Interaktion mit dem Menü.
Für die Umschaltlogik muss eine neue Skript-Komponente *ChangeTurnMode* (siehe Listing 2) erstellt werden, die die Auswahl-ID der Dropdown-Menüs ausliest, wenn diese sich verändert haben, und in diesem Zusammenhang die *Snap Turn Provider* Komponente und die *Continuous Turn Provider* Komponente auf dem XRRig an- und ausschaltet. So wird bei der Wahl von „Smooth Turn“ der *Snap Turn Provider* aus- und der *Continuous Turn Provider* eingeschaltet. Auf dieselbe Art und Weise funktioniert das Umschalten zwischen den Fortbewegungsoptionen durch die Komponente *ChangeLocomotionMode* (siehe Listing 3).

```
1  using UnityEngine;
2  using UnityEngine.XR.Interaction.Toolkit;
3
4  public class MenuToggle : MonoBehaviour
5  {
6      public InputHelpers.Button menuButton = InputHelpers.Button.SecondaryButton;
7
8      public XRController controller;
9
10     public GameObject menu;
11
12     private bool lastPressOutput = false;
13
14     // Update is called once per frame
15     void Update()
16     {
17         if (controller.inputDevice.IsPressed(menuButton, out bool pressed, 0))
18         {
19             if(pressed && pressed != lastPressOutput)
20                 menu.SetActive(!menu.activeSelf);
21             lastPressOutput = pressed;
22         }
23     }
24 }
```

Listing 1: MenuToggle.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.XR.Interaction.Toolkit;

public class ChangeTurnMode : MonoBehaviour
{
    public DeviceBasedSnapTurnProvider snapTurnProvider;

    public DeviceBasedContinuousTurnProvider smoothTurnProvider;

    public void OnOptionChange(int option)
    {
        if (option == 0)
        {
            smoothTurnProvider.enabled = false;
            snapTurnProvider.enabled = true;
        }
        else
        {
            smoothTurnProvider.enabled = true;
            snapTurnProvider.enabled = false;
        }
    }
}
```

Listing 2: ChangeTurnMode.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.XR.Interaction.Toolkit;

public class ChangeLocomotionMode : MonoBehaviour
{
    public DeviceBasedContinuousMoveProvider continuousMoveProvider;

    public TeleportationProvider teleportationProvider;

    public void OnOptionChange(int option)
    {
        if (option == 0)
        {
            continuousMoveProvider.enabled = false;
            teleportationProvider.enabled = true;
        }
        else
        {
            continuousMoveProvider.enabled = true;
            teleportationProvider.enabled = false;
        }
    }
}
```

Listing 3: ChangeLocomotionMode.cs

Zurücksetzen der Spielelemente

Nach Fertigstellung des grundlegenden Gameplays kann an dessen Umwandlung zu einer Game Loop gearbeitet werden. Um dies zu erreichen, wird ein Zurücksetzen der interagierbaren Objekte, also des Balles und der Dosen der Dosenpyramide, implementiert. Somit kann der User mehrere Runden Dosenwerfen spielen, ohne das Spiel ständig neu starten zu müssen.
Um den Ball auf seine Ursprungspostion zurückzusetzen, wird eine weitere Skriptkomponente *BallBehaviour* (siehe Listing 4) erstellt, die einen Controller und ein Button Mapping annimmt. In der Start-Methode wird dann die initiale Position des Balls festgehalten sowie die *Rigidbody-Komponente* abgespeichert, um sowohl Position als auch Physik später zurücksetzen zu können. Wird das Drücken des Buttons auf dem spezifizierten Controller innerhalb der Update-Methode registriert, wird der Ball auf die am Spielstart identifizierte Position zurückgesetzt, ebenso die Geschwindigkeit, die auf den Ball einwirkte, bevor dieser zurückgesetzt wurde.
Auch für die Zurücksetzung der Dosen wird eine eigene Skriptkomponente *CanBehaviour* (siehe Listing 5) entwickelt. Dabei besitzt jede Dose diese Komponente. Die Arbeitsweise des Skriptes funktioniert dabei genauso wie beim Ball, nur dass hier ebenfalls beim Spielstart die Rotation der Dosen mit gespeichert wird, damit diese wieder aufrecht stehen, wenn sie zurückgesetzt werden. Diese Funktionalität wird auf denselben Button desselben Controllers gelegt, auf dem auch das Zurücksetzen des Balls liegt. Somit werden bei jedem Zurücksetzen sowohl der Ball als auch die gesamte Dosenpyramide in den Ursprungszustand gebracht.

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.XR.Interaction.Toolkit;

public class BallBehaviour : MonoBehaviour
{
    public InputHelpers.Button aButton = InputHelpers.Button.PrimaryButton;
    public XRController rightController;
    private Vector3 ballPositionOnStartUp;

    private Rigidbody rb;
    // Start is called before the first frame update
    void Start()
    {
        ballPositionOnStartUp = this.transform.position;
        rb = this.GetComponent<Rigidbody>();
    }

    // Update is called once per frame
    void Update()
    {
        if (rightController.inputDevice.IsPressed(aButton, out bool isPressed))
        {
            if (isPressed)
            {
                this.transform.position = ballPositionOnStartUp;
                rb.velocity = Vector3.zero;
            }
        }
    }
}
```

Listing 4: BallBehaviour.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.XR.Interaction.Toolkit;

public class CanBehaviour : MonoBehaviour
{
    public InputHelpers.Button aButton = InputHelpers.Button.PrimaryButton;

    public XRController rightController;

    private Vector3 canPositionOnStart;

    private Quaternion canRotationOnStart;

    private Rigidbody rb;
    // Start is called before the first frame update
    void Start()
    {
        canPositionOnStart = transform.position;
        rb = GetComponent<Rigidbody>();
        canRotationOnStart = transform.rotation;
    }

    // Update is called once per frame
    void Update()
    {
        if (rightController.inputDevice.IsPressed(aButton, out bool isPressed))
        {
            if (isPressed)
            {
                transform.position = canPositionOnStart;
                transform.rotation = canRotationOnStart;
                rb.velocity = Vector3.zero;
            }
        }
    }
}
```

Listing 5: CanBehaviour.cs

3.2.2. Umsetzung in der Unreal Engine 5

Initialisierung

Das Projekt wird mit Hilfe des VR-Templates der Unreal Engine 5 initialisiert. Dadurch stehen zu Beginn des Projektes folgende Features und Assets zur Verfügung:

- eine Beispielmap,
- *GrabComponent* zum Greifbarmachen von Objekten,
- Spielercharakter mit in Blueprint umgesetzten Snap Turn und Teleportation,
- Kopf- und Controllertracking,
- *VR-Spectator* zum Beobachten des VR-Spielers außerhalb der virtuellen Realität.

Für die Unreal Engine 5 existiert nativ kein SDK für VR, wodurch das Setup dafür entfällt.

Szenenaufbau und interagierbare Elemente

Zu Beginn erfolgte die Generierung eines neuen Levels sowie der Nachbau der grundlegenden Struktur des Unity-Projektes. So wird eine Ebene erstellt, auf der zwei Würfel seitlich lang skaliert und parallel zueinander aufgestellt wurden. Die Erzeugung eines *Actor-Blueprints* dient als Grundlage für den Ball. Nach Erstellung einer Kugel erfolgt die Verkleinerung um den Faktor 0.1 auf allen Achsen des Objektes. Weiterhin wird dem Ball eine *GrabComponent* gegeben, damit dieser vom Spieler aufgehoben werden kann. Auch die Dosenpyramide ist von der Grundstruktur her ähnlich wie im Unity-Projekt aufgebaut. So wird zunächst ein Dosen-Actor erstellt, indem eine aus der *Quixel Bridge*, eine in der Unreal Engine integrierte Bibliothek für 3D-gescannte Modelle, importierte alte Dose verwendet wird. Mit Hilfe dieses Actors erfolgt anschließend die Erstellung der Dosenpyramide, die aus sechs Dosen-Actors besteht. Die Abbildung 3.5 zeigt den Aufbau der Szene nach Abschluss der genannten Arbeitsschritte.

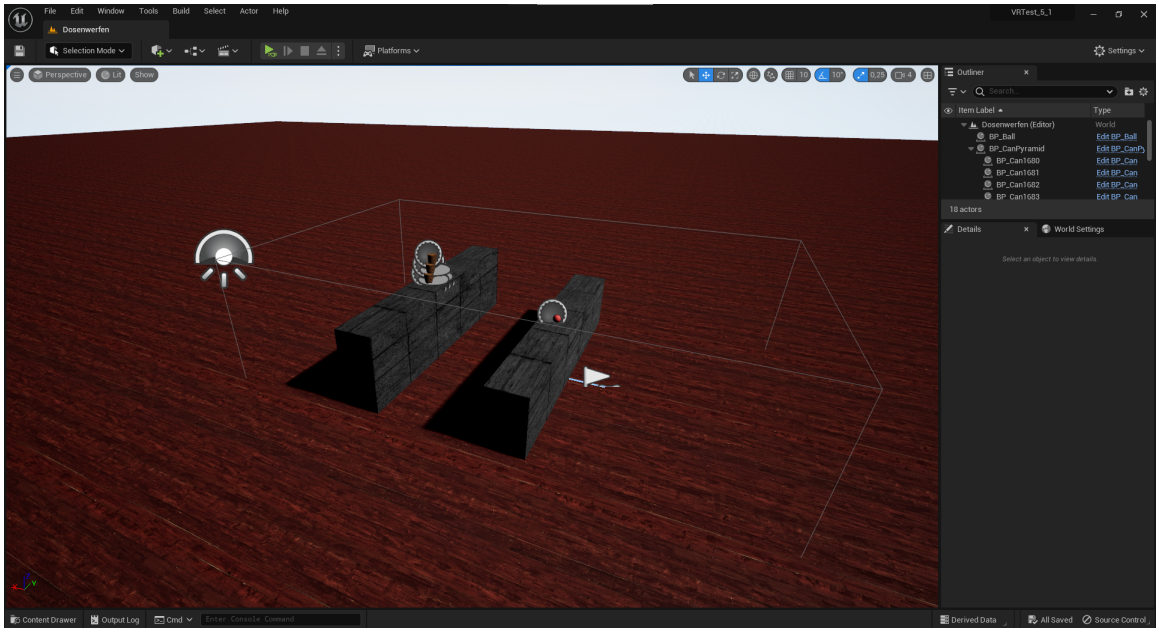


Abbildung 3.5.: Aufbau der Szene in der Unreal Engine 5

Bewegung

Da Snap Turn und Teleport bereits implementiert waren, soll deren Grundstruktur weitergenutzt und um die Optionen Continuous Locomotion und Continuous Turn erweitert werden. Das erfolgt mit der Erstellung von Boolean-Variablen, um den Status der aktuell ausgewählten Optionen zu repräsentieren. Für die Implementierung des Continuous Turns kann die durch das Template bereits vorhandene Methode *SnapTurn* für die Snap Turn Funktion verwendet werden. Somit geschieht das Auslösen der Funktion während der Laufzeit nicht mehr einmalig, sondern öfter hintereinander, solange der Joystick in eine gewünschte Richtung gedrückt wird. Sobald der Joystick in seine neutrale Position zurückkehrt, stoppt das Aufrufen der Funktion und der Spieler dreht sich nicht mehr. Dabei wird abhängig von der Drehart die pro Aufruf der Methode gedrehte Gradanzahl festgelegt. So dreht man sich beim *Snap Turn* um 45° pro Betätigen des Joysticks und beim *Continuous Turn* um $4 * AxisValue$ pro Tick, damit der Spieler sich in einem angenehmen Tempo bewegt und keine Motion Sickness erfährt. Hier wird der Parameter *AxisValue* durch einen Wert zwischen -1 und 1 repräsentiert, je nachdem wie stark der Joystick nach links oder rechts gedrückt wird. Die vollständige Funktion ist in Abbildung 3.6 zu sehen.

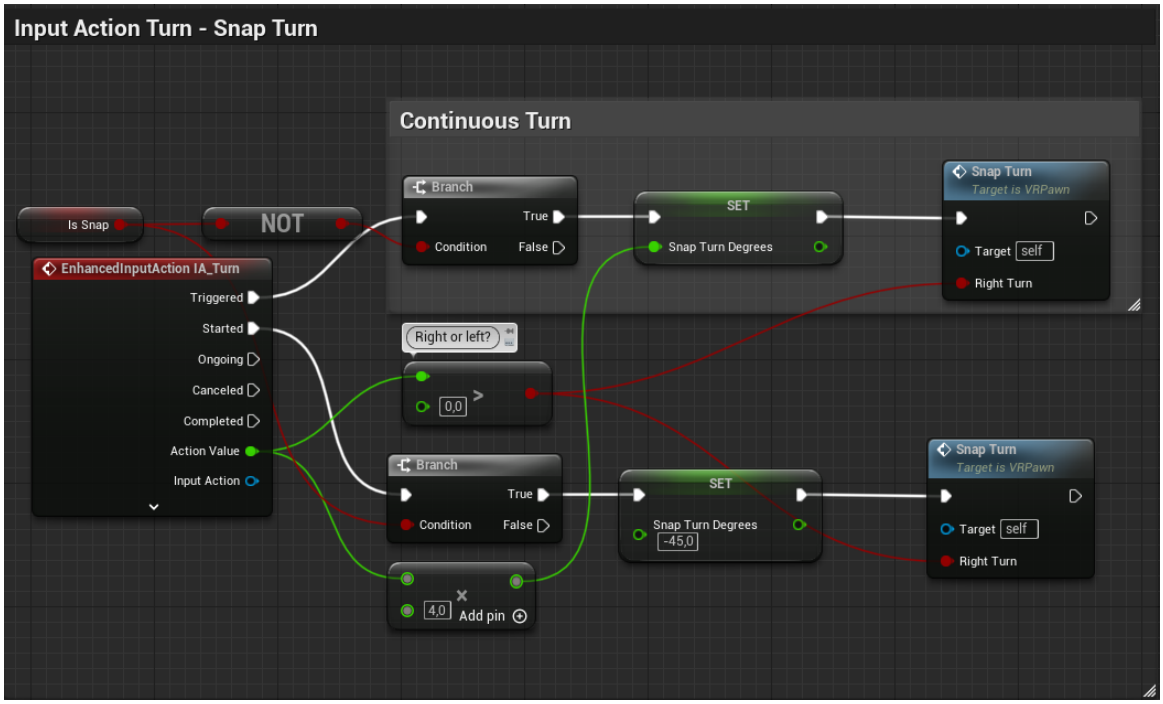


Abbildung 3.6.: Aufbau der Turn Funktionen im VRPawn Blueprint

Für die Continuous Locomotion wird dem *VRPawn-Blueprint* eine *Floating Pawn Movement Component* hinzugefügt. Das hat den Grund, dass diese Komponente ein eingebautes Movementsystem besitzt, mit dem kontinuierliche Bewegungen einfacher zu implementieren sind. Die *Pawn-Klasse* besitzt dieses Movementsystem nicht standardmäßig, wie es z.B. bei der *Character-Klasse* der Fall wäre. Da das Ändern der Parent-Klasse des *VRPawns* auf die *Character-Klasse* zu Spawnproblemen führt, bei denen der Spieler weit vom eigentlichen Spawnpunkt weg auftauchte und sich nicht mehr bewegen konnte, erweist sich diese Variante als ungeeignet. Mit Hilfe des Movementsystems der *Floating Pawn Movement Component* wird eine kontinuierliche Fortbewegung anhand der Headsetausrichtung etabliert. Das heißt, dass man für horizontale und vertikale Fortbewegung jeweils den Vorwärts- und Rechtsvektor des HMDs verwendet. Zum Skalieren der Geschwindigkeit multiplizierte man den erfassten Achsenwert des rechten Joysticks mit einer Floatkonstante mit dem Wert 0.5, um eine angenehme Geschwindigkeit zu erzeugen. In Abbildung 3.7 wird die vollständige Continuous-Locomotion-Funktion dargestellt.

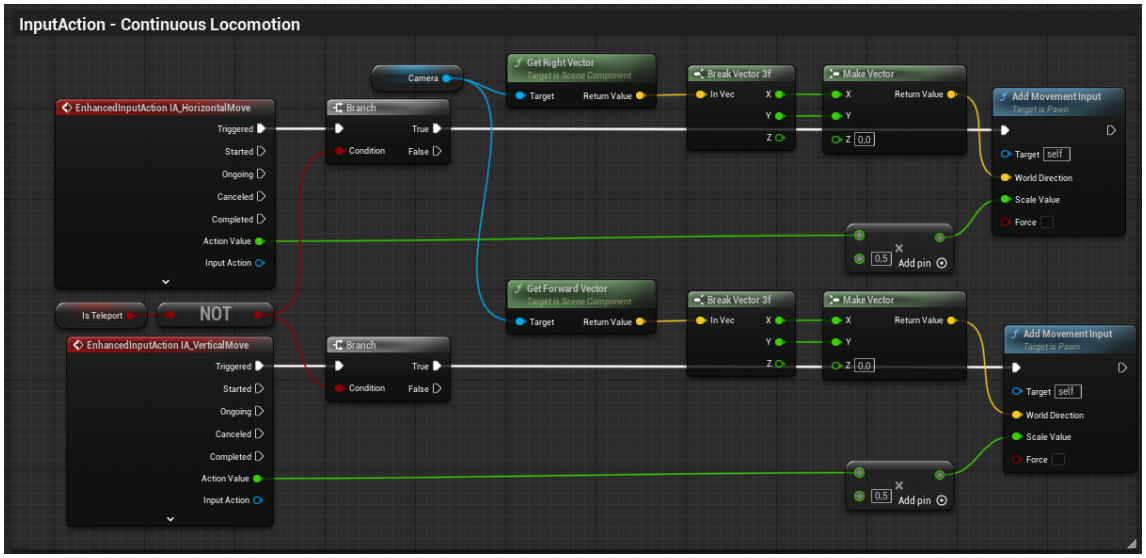


Abbildung 3.7.: Aufbau der Continuous Locomotion Funktion im VRPawn Blueprint

Um den Wechsel zwischen Teleport und Continuous Locomotion vornehmen zu können, werden kleinere Anpassungen an der Teleport-Funktion vorgenommen. So entsteht die Möglichkeit, durch die zugehörige Boolean Variable die weiteren Funktionen des Teleports abzuschalten, bis sie wieder aktiviert werden. Die Abbildung 3.8 zeigt die veränderte Teleport-Funktion.

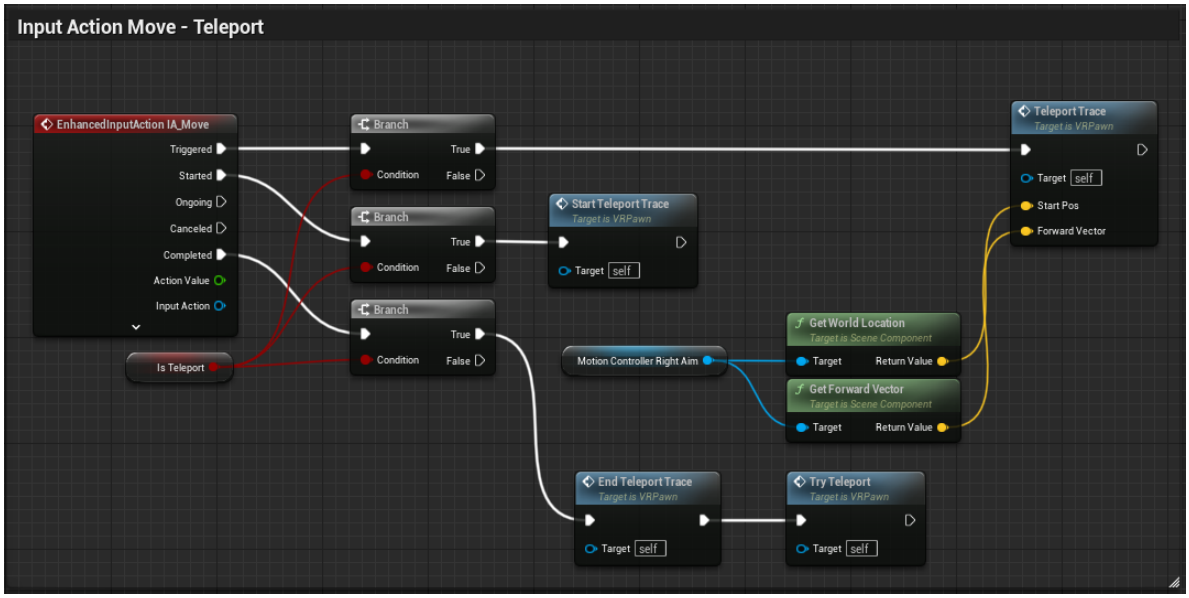


Abbildung 3.8.: Aufbau der Teleport Funktion im VRPawn Blueprint

User Interface

Für den Wechsel zwischen den Bewegungsoptionen wird das vorgegebene Menü-UI verwendet, das standardmäßig über die Controller durch Drücken eines Buttons aufgerufen werden kann und bereits interagierbar ist, sowohl mit dem Controller selbst als auch per Joystick. Wie im Unity-Projekt können Dropdown-Menüs genutzt werden, bei denen bei Wertänderung überprüft wird, welche Option aktuell ausgewählt ist. Anschließend werden die entsprechenden Optionen mit Hilfe der Boolean-Variablen innerhalb des *VRPawns* an- und ausgeschaltet. Um auf den *VRPawn* zugreifen zu können, wird in dem *Widget-Blueprint*, in dem das UI definiert ist, zu Beginn des Spiels der *VRPawn* im Level gesucht und nach dem Finden abgespeichert. In Abbildung 3.9 werden die Funktionen zum Auslesen und Ändern der Bewegungsoptionen im Menü-Blueprint gezeigt.

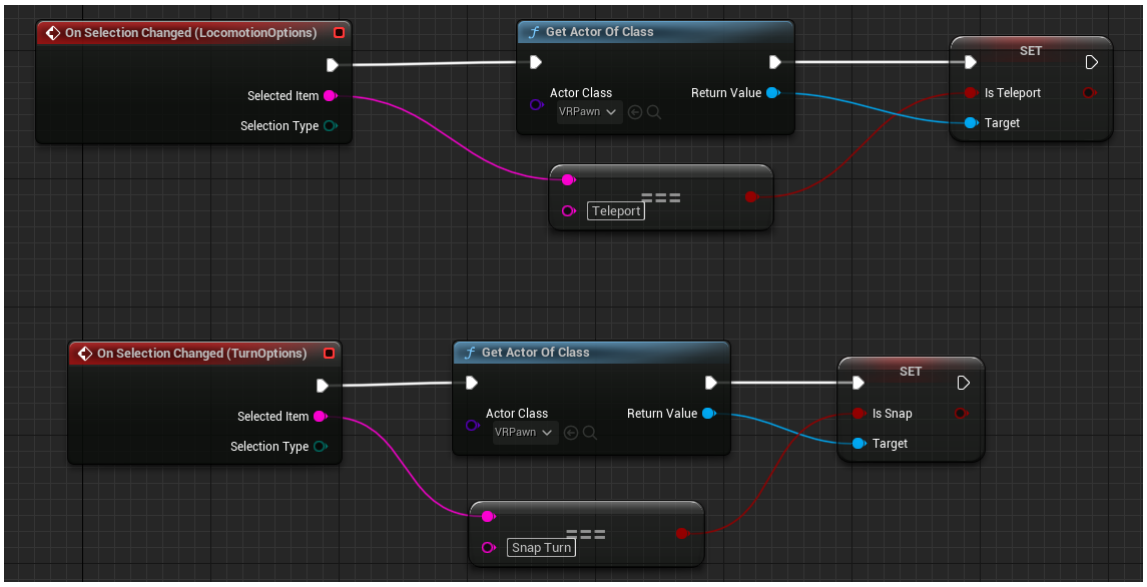


Abbildung 3.9.: Herausfinden und Einstellen des neuen Bewegungsmodus nach Änderung im Dropdown-Menü

Zurücksetzen der Spielelemente

Allein für das Zurücksetzen der Spielelemente nach Spielbeendigung wäre die Implementierung zusätzlicher Features nicht notwendig gewesen. In dem vom Template vorgegebenen UI existiert bereits ein Button, der bei Betätigung das Level sofort neu lädt. Dadurch wäre eine Resetmöglichkeit bereits gegeben. Um dennoch eine höhere Ähnlichkeit und damit eine bessere Vergleichbarkeit zum Unity-Projekt zu erreichen, wird dessen Reset-System nachgebaut. So erhalten sowohl der Ball als auch jede einzelne Dose ein *Custom Event* in ihren Blueprint-Klassen, mit dem sowohl Position als auch Geschwindigkeit des Objektes auf den Ursprung gesetzt werden (siehe Abbildung 3.10 und 3.11).

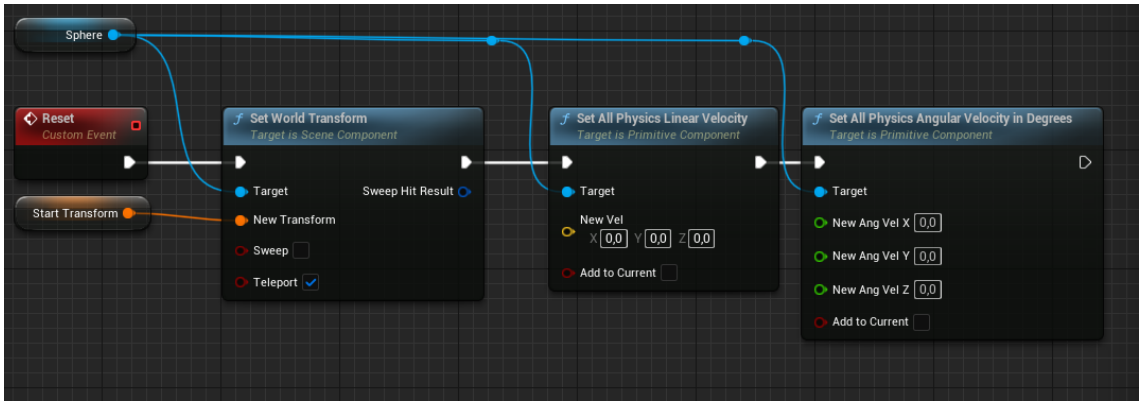


Abbildung 3.10.: Reset-Funktion des Balls

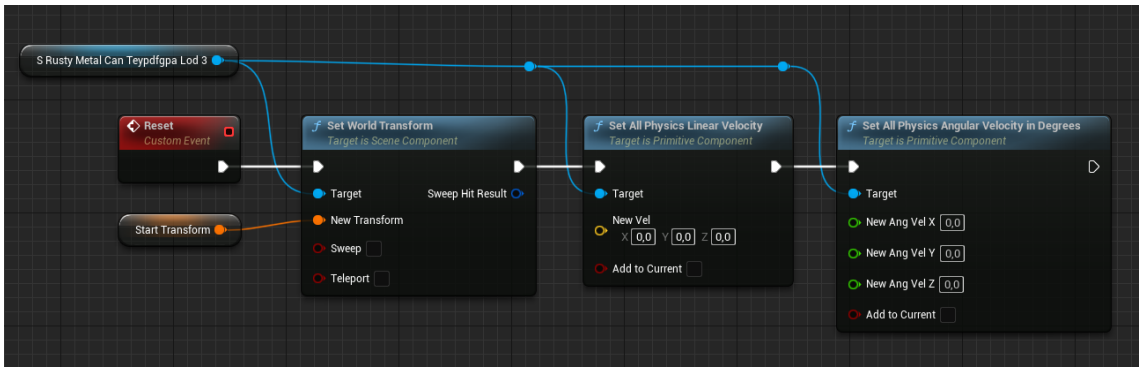


Abbildung 3.11.: Reset-Funktion der Dose

Anders als bei Unity kann hier das Zurücksetzen nicht direkt in den beiden Klassen auf denselben Button gelegt werden, da die Unreal Engine nur eine der beiden Aktionen gleichzeitig bei Betätigung des Buttons ausführen kann. Stattdessen wird in der Blueprint-Klasse des Levels, die während des Spielvorganges aktiv ist, auf einen der freien Controller-Buttons das Aufrufen beider Reset-Funktionen von Ball und Dosen gelegt. So kann sichergestellt werden, dass mit einem Knopfdruck alle zurücksetzbaren Elemente zurückgesetzt werden. Die genaue Umsetzung dieser Funktion ist in Abbildung 3.12 zu sehen.

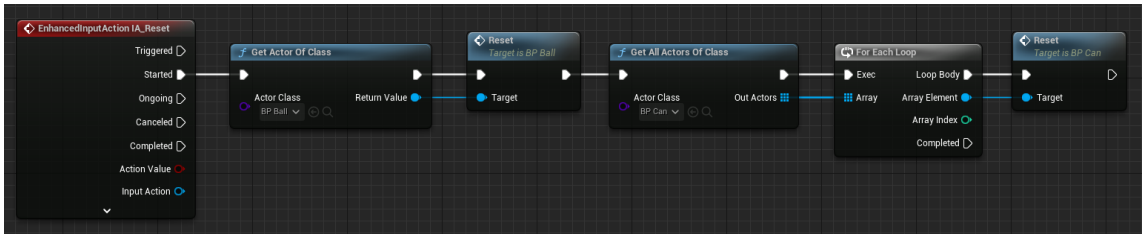


Abbildung 3.12.: Inputevent im Level-Blueprint zum Aufrufen der Reset-Funktionen

3.3. Zusammenfassung

Die Entwicklung des Projektes war in beiden Engines einfach umzusetzen. Dabei musste in Unity mehr vorbereitet werden, danach konnte man allerdings schnell viele der gesetzten Anforderungen mit Hilfe von vorgegebenen Komponenten erfüllen. In Unreal waren die meisten Anforderungen bereits im Template vorhanden. Allerdings mussten alle anderen Funktionen komplett selbstständig entwickelt werden, wodurch die Entwicklungszeit sich zu der in Unity anglich. Weiterhin muss an dieser Stelle angemerkt werden, dass in Unity sämtliche Funktionen komponentenbasiert umgesetzt werden. Dies ist theoretisch auch in der Unreal Engine 5 möglich, spiegelt aber nicht den typischen Workflow der Engine wieder und wäre somit zu umständlich für die Umsetzung gewesen.

4. Vergleich der Engines

Dieses Kapitel behandelt den Vergleich zwischen Unity und der Unreal Engine 5 basierend auf den im Kapitel 3 erworbenen Erkenntnissen und stellt somit eine Analyse durch einen Experten im Thema Game Development dar. Um die Engines effektiv miteinander zu vergleichen, werden spezielle Kriterien erarbeitet, an denen sich der Vergleich orientiert. Im folgenden Kapitel werden diese einzelnen Kriterien unter dem Aspekt beider Engines beleuchtet und gegenübergestellt.

4.1. Kriterien

Die Kriterien für die Analyse setzen sich aus den wichtigsten Schritten zur Planung von VR-Projekten zusammen. Für typische Non-VR-Videospielprojekte gibt es eine Vielzahl von 3D und 2D Templates, die in vielen Engines zur Verfügung gestellt werden und bereits eine Fülle von Standardfunktionen mit sich bringen, um den Start in das Projekt zu vereinfachen. Daraus ergeben sich die ersten Kriterien:

- Es sind Templates für die Erstellung von VR-Projekten verfügbar.
- Die VR-Templates enthalten alles, was für den Start in ein VR-Projekt benötigt wird.

Anschließend ist es wichtig, dass eine Engine eine Vielzahl von VR-Hardware unterstützt, um eine breite Masse von Anwendern mit unterschiedlichen Geräten abdecken zu können und das Testen und Ausführen der Software mit der Hardware zu ermöglichen. Dadurch entstehen zwei weitere Kriterien für die Analyse:

- Die Engine kann die gängigsten Frameworks bzw. Plattformen bedienen.
- Die Engine kann die gängigste VR-Hardware bedienen.

Dabei werden vor allem die bedienten Plattformen, welche die generelle Schnittstelle zur Hardware außerhalb der Engine darstellen, als auch die nutzbaren Frameworks, welche typische VR-Funktionen für bestimmte Plattformen zur Verfügung stellen, untersucht, da die unterstützten Plattformen Voraussetzung für die Kompatibilität der einzelnen Geräte sind und Frameworks die Implementierung der einzelnen Funktionen je nach Plattform vereinfachen. Abschließend muss der Abruf der Sensor- und Inputdaten aus den Controllern und den einzelnen Buttons möglich sein, um darauf zurückgreifend mit der virtuellen Umgebung interagieren zu können. Dafür wird das Kriterium „Es können alle nötigen Sensor- und Inputdaten der Hardware aus der Engine heraus abgelesen werden“ aufgestellt. Diese hier zusammengetragenen Kriterien spiegeln den Kern jeder VR-Anwendung wieder und werden deshalb repräsentativ für den Vergleich genutzt und im Folgenden erneut zusammengefasst:

- Es sind Templates für die Erstellung von VR-Projekten verfügbar.
- Die VR-Templates enthalten alles, was für den Start in ein VR-Projekt benötigt wird.
- Die Engine kann die gängigsten Frameworks bzw. Plattformen bedienen.
- Die Engine kann die gängigsten VR-Hardwares bedienen.
- Es können alle nötigen Sensor- und Inputdaten der Hardware aus der Engine heraus abgelesen werden.

4.2. Vergleich

4.2.1. Verfügbarkeit der VR-Templates

Für einen schnellen Start in das eigene VR-Projekt bieten die nützlichen Funktionen der Templates eine gute Voraussetzung dafür, dass der Entwickler sich auf die wesentlichen Funktionen seines Projektes konzentrieren kann. Sowohl Unity als auch die Unreal Engine besitzen ein solches Template. Bei Unity kann dies direkt im Unity Hub (s. Abbildung 4.1) erstellt werden. Bei der Unreal Engine kann das Template im Launcher (s. Abbildung 4.2) der Anwendung genutzt werden. Das heißt, dass diese Templates in beiden Engines zugänglich und einfach zu finden sind.

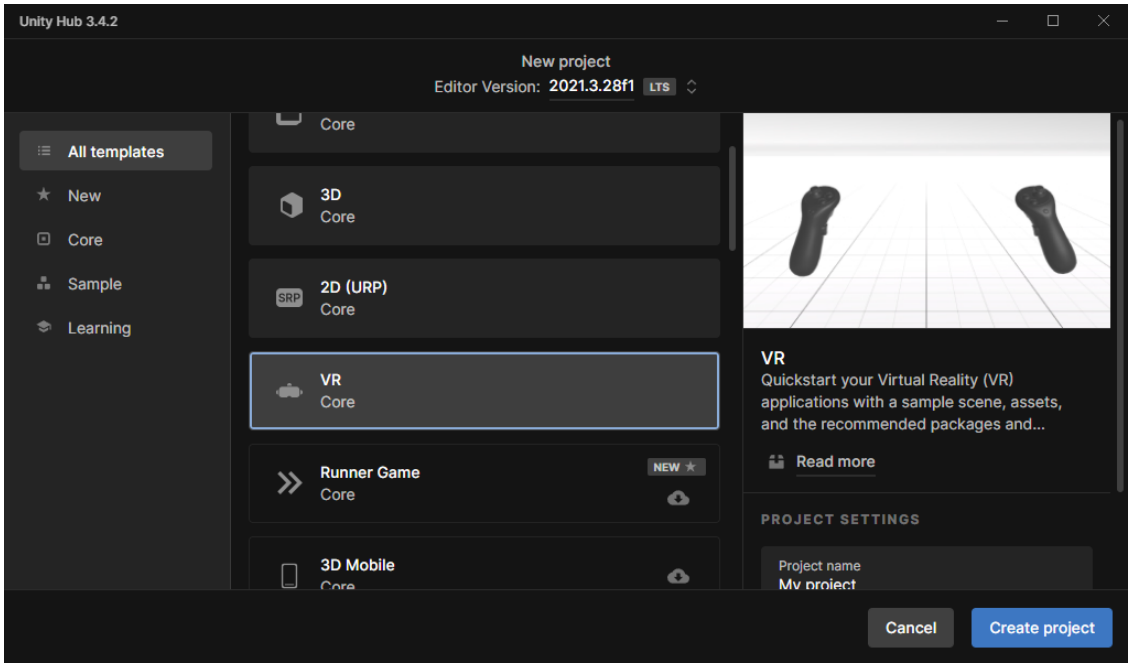


Abbildung 4.1.: Das Auswahlfenster für Templates im Unity Hub

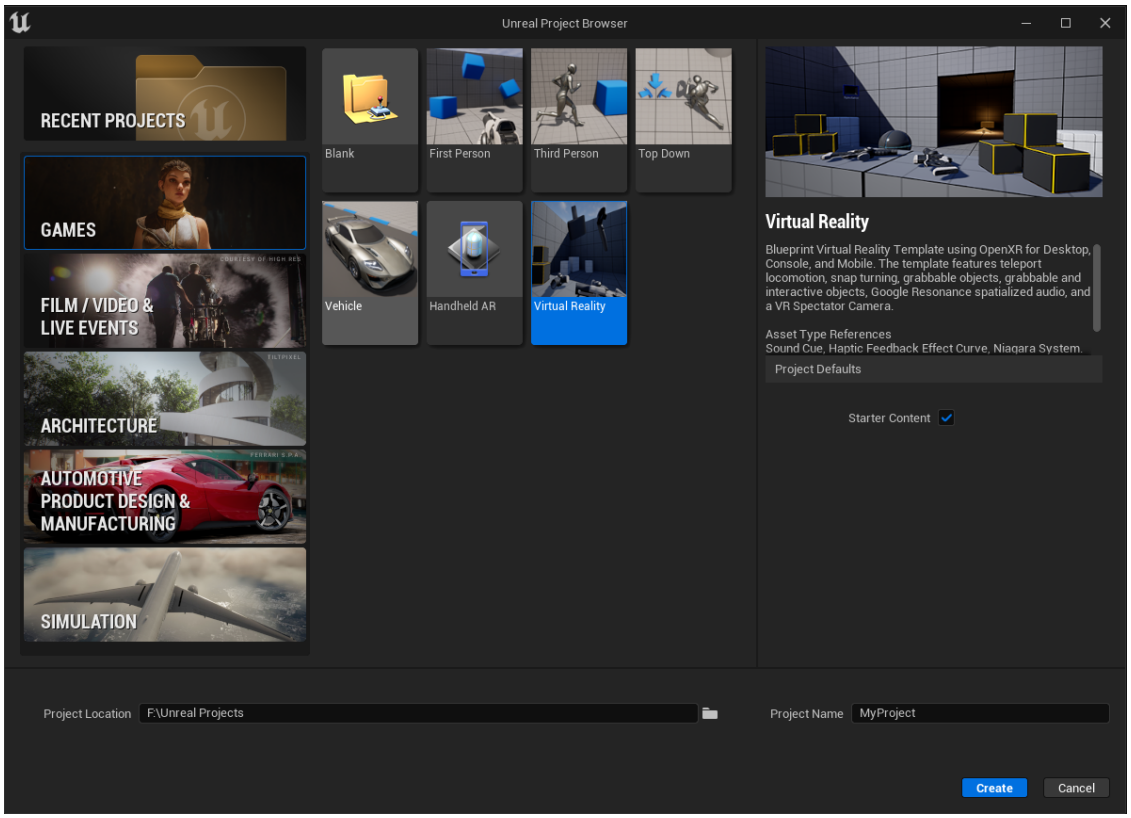


Abbildung 4.2.: Das Auswahlfenster für Templates im Unreal Engine 5 Launcher

4.2.2. Vorgefertigte Inhalte der Templates

Die Verfügbarkeit eines Templates sagt nichts über dessen Qualität aus. Im Folgenden werden deswegen die Inhalte der Templates der Engines aufgelistet und verglichen. In der Unreal Engine befindet sich innerhalb des Templates alles, was ein Entwickler benötigt, um mit einem VR-Projekt starten zu können. Dies beinhaltet:

- Beispiel-Level, in dem man alle Features zur Entwicklung von VR-Anwendungen testen kann
- Spielerfigur (*VRPawn*) mit folgenden Funktionen:
  - Tracking des Headsets sowie der Controller
  - Greifen als Blueprintfunktion
  - *GrabComponent*, die Objekten erlaubt, vom Spieler aufgehoben zu werden
  - Interaktion mit gegriffenen Objekten per Trigger-Taste, z. B. das Abfeuern einer Pistole
  - Menü, das auf beiden Controllern aufgerufen und sowohl per Joystick als auch per *Ray Interactor* bedient werden kann
  - *VRSpectator* Blueprint, das per Tastenkombination in Laufzeit zugeschaltet werden kann, um die Spielerfigur von außen beobachten zu können
  - Teleportationssystem zur Fortbewegung
  - Drehen der Spielerfigur mit Snap Turn um 45° pro Betätigung des Joysticks.

Die zuletzt genannten Arten der Charakterbewegung sind deshalb initial vom Template gegeben, da diese erwiesenermaßen weniger Motion Sickness beim Anwender hervorrufen als die kontinuierlichen Varianten [26, 27].Motion Sickness beschreibt dabei eine Art Schwindelgefühl, das dadurch hervorgerufen wird, dass die wahrgenommenen Bewegungen stark von den inneren Gleichgewichtswahrnehmungen im Ohr abweichen. Die Anfälligkeit für Motion Sickness ist bei jedem Menschen unterschiedlich, weshalb in den meisten VR-Anwendungen sowohl beim Drehen als auch beim Bewegen beide Optionen gegeben werden.

In Unity ist die Menge an vordefinierten Funktionen geringer als in der Unreal Engine. Nach der Erstellung des Projektes mit Hilfe des Templates sind folgende Features vorgegeben:

- Headset- und Controllertracking durch Tracked Pose Driver Komponenten
- Controllermodelle zur Visualisierung der Controllerposition.

Die Tracked Pose Driver Komponenten übernehmen das Tracking der Position dieser einzelnen Hardwarekomponenten. Dabei ist weder eine Interaktions- noch eine Bewegungsmöglichkeit vorhanden. Dafür muss das SDK XR Interaction Toolkit (XRTK) installiert und anschließend eingerichtet werden, was das Erstellen folgender Komponenten beinhaltet:

- *XR Origin* - Positionsursprung des Spielers
- *XR Interaction Manager* - Grundlage für alle Interaktionssysteme
- *Locomotion System* - Grundlage aller Bewegungsfunktionen
- *XR Ray- oder Direct Interactor* - ermöglichen das Greifen von Objekten.

Die Greifbarkeit von Objekten lässt sich mit dem XRTK über die XR Grab Interactable Komponente recht schnell hinzufügen, ähnlich wie in der Unreal Engine 5.
Allgemein lässt sich schlussfolgern, dass die Unreal Engine 5 einen besseren Start in die VR-Entwicklung mit ihrem Template liefert, allerdings im Laufe des Projektes nicht so modular anpassbar ist wie Unity mit dem XRTK, da z. B. die kontinuierlichen Bewegungsoptionen, aber auch andere Varianten der Grundfunktionen komplett selbst implementiert werden müssen. Beim XRTK in Unity braucht man zwar mehr Zeit zum Einrichten, man kann aber anschließend durch die Einstellungen der Skriptkomponenten diese relativ einfach anpassen, ohne sie vollkommen neu implementieren zu müssen. Außerdem sind mehr Varianten der Grundfunktionen als fertige Komponenten verfügbar, die schnell miteinander ausgetauscht werden können, z. B. Snap Turn und Continuous Turn.

4.2.3. Unterstützte Frameworks und Plattformen

Aufgrund der Vielfalt an Herstellern von VR-Hardware ist es wichtig zu wissen, welche Plattformen und damit welche Frameworks von den Engines sowohl in der Entwicklung als auch im fertigen Produkt unterstützt werden. Dadurch ist die Wahl einer Engine für bestimmte Projekte immens beeinflusst, da man so viele Plattformen wie möglich bedienen möchte, ohne die Engine wechseln zu müssen.
In Unity werden folgende Plattformen unterstützt:

- Oculus
- Microsoft HoloLens
- OpenXR[1]
- Magic Leap[2]
- PlayStation VR für registrierte PlayStation-Entwickler.

Dadurch sind in Unity eine Vielzahl von Frameworks verfügbar:

- XR Interaction Toolkit (XRTK) für den OpenXR-Standard
- OpenVR für die Entwicklung von SteamVR-Anwendungen
- Mixed Reality Interaction Toolkit (MRTK), welches auf dem XRTK aufbaut und durch Microsoft unter anderem für den Support der HoloLens erweitert wurde.

So kann ein breites Spektrum an Hardware abgedeckt werden.[28]
Im Gegensatz dazu besitzt die Unreal Engine keine Frameworks für VR. Während die SDKs in Unity den Großteil der initialen Arbeit durch qualitativ hochwertige Komponenten bereits übernehmen, müssen diese Funktionen, bis auf wenige Ausnahmen, selbst implementiert werden. Stattdessen bietet Unreal lediglich Plugins für den Plattform-Support an, z.B. für OpenXR, Microsoft HoloLens und SteamVR. Neu in der Version 5 der Engine ist das Zusammenlegen von Windows Mixed Reality und Oculus in das OpenXR Plugin. Für SteamVR und HoloLens ist weiterhin ein

[1] OpenXR ist ein OpenSource-VR-Standard, der die Kommunikation zwischen Software und einer Vielzahl von Hardwares ermöglicht, darunter gängige Plattformen wie Oculus, Windows Mixed Reality, Valve SteamVR und Vive.
[2] Magic Leap ist eine Mixed Reality Brille und somit ähnlich zur HoloLens von Microsoft.

separates Plugin erforderlich [29]. Es gibt auch ein Extraplugin für HP Reverb Motion Controller Inputs, das für die Controller der Brillen der HP Reverb Reihe zur Verfügung gestellt wird. Dieses ist für initiale Funktionen des VR-Templates nicht notwendig, muss aber installiert werden, sobald man eigene Inputs auf den Controllern definiert. Daraus resultiert auch das Umschreiben aller anderen Inputmappings auf dieses neue Plugin. Für Entwickler mit einer HP Reverb ist das Setup mit der Brille also mit einer Menge Extraaufwand verbunden. Ansonsten deckt die Unreal Engine 5 äquivalent so viele Plattformen ab wie Unity.
In dem Punkt existiert zwischen den beiden also kein signifikanter Unterschied. Allerdings besitzt Unity eine Vielzahl von Frameworks, welche die Entwicklung von VR-Projekten unterstützen, indem sie rudimentäre Funktionen bereits in Form von Komponenten vorgeben. Diese Frameworks fehlen leider in der Unreal Engine.

4.2.4. Kompatible Brillenhardware

Beide Engines bedienen eine große Menge an Plattformen, um eine Vielzahl von Brillenhardware zu unterstützen.
In Unity werden sowohl die Brillen von Oculus (s. Abbildung 4.3) als auch die Windows Mixed Reality Brillen (s. Abbildung 4.4) mit OpenXR abgedeckt. Für die HTC Vive (s. Abbildung 4.5) und Valve Index (s. Abbildung 4.6) kann ebenfalls OpenXR verwendet werden. Wenn diese Brillen für SteamVR genutzt werden sollen, sollte das OpenVR-Framework verwendet werden, da OpenVR von Valve und somit direkt für die Kommunikation zwischen den Brillen und SteamVR entwickelt wurde. Diese breite Hardwarekompatibilität trifft auch auf die Unreal Engine 5 zu. Durch die Unterstützung von OpenXR und SteamVR sind die gebräuchlichsten Brillen alle kompatibel mit der Engine. Zum Schluss kann man zusammenfassend sagen, dass beide Engines im Bezug auf die Hardwarekompatibilität identisch gut funktionieren.

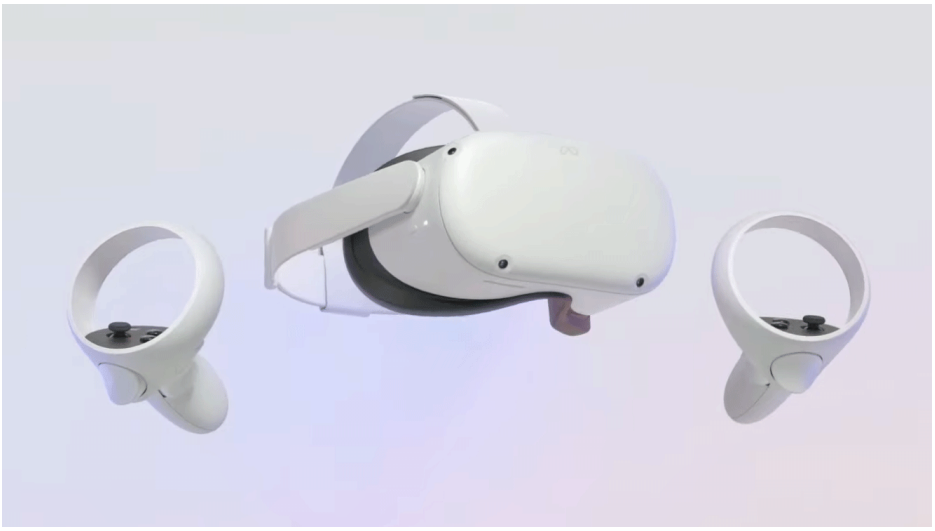


Abbildung 4.3.: Die Meta Quest 2 als Vertreter der Oculus-Hardware[30]

Abbildung 4.4.: HP Reverb G2 als Vertreter der Windows Mixed Reality Hardware[31]

Abbildung 4.5.: HTC Vive als Vertreter der Drittanbieter-SteamVR-Hardware[32]

Abbildung 4.6.: Valve Index als nativer Vertreter der SteamVR-Hardware[33]

4.2.5. Abrufen der Sensor- und Inputdaten

Um spezifische Funktionen in Projekten umzusetzen, ist es wichtig zu wissen, wie auf verschiedene Sensor- und Inputdaten zugegriffen werden kann. Hier wird aus Zeitgründen nur das XRTK von Unity mit dem allgemeinen Prozess der Unreal Engine verglichen.
Mit dem XRTK von Unity können Sensordaten auf verschiedene Weisen genutzt werden. Die einfachste Variante ist das Nutzen der gegebenen Komponenten, z.B. Tracked Pose Driver oder Skript-Komponenten, wie der XR Controller. Auch in eigenen Skripten kann auf die Tracking-Daten des Controllers zugegriffen werden, indem man z. B. bei jedem Aufruf der Update-Methode die aktuelle Position der Controller speichert. Durch das vorhergegangene XR-Setup werden diese Koordinaten basierend auf den Bewegungen des Spielers in Echtzeit aktualisiert und sind

somit als Änderung der Positionskoordinaten der Controllerobjekte über die Zeit zu verstehen. Diese Optionen gelten ebenso für das Auslesen der Inputdaten von Joystick und Buttons. Hier kann ebenfalls auf Komponenten zurückgegriffen werden, wie z. B. die verschiedenen Arten der Locomotion, die das Drehen und Bewegen des Spielers steuern und dafür die Daten der Joysticks auslesen. Diese Komponenten lassen sich einfach über die Hierarchie oder im Inspector, in dem die Eigenschaften der Objekte und darauf liegender Komponenten eingesehen und eingestellt werden können, hinzufügen. Alternativ ist das direkte Auslesen der Daten in einem Skript möglich. Dabei kann man entweder direkt nach dem verbundenen Gerät suchen und von dort Daten abgreifen oder man nutzt die Bibliothek des XRTKs und importiert einen XR Controller in das Skript, welcher dann direkt auf einen validen Input eines Buttons, Joysticks oder ähnlichem getestet werden kann. Die letzte Variante beinhaltet das Mappen von Input Actions als sogenannte Input Action Maps. Eine Input Action beschreibt ein Event, das beim Betätigen eines spezifischen Inputs ausgelöst wird. Dieser Input ist in einer Input Action Map zuweisbar, die eine flache Liste von Inputs und Input Actions darstellt und diese paarweise zueinander bindet. Diese Methode kommt der Methode von Unreal sehr nahe. Dort wird Input durch das in der 5.1 hinzugefügte Enhanced Input System in einem Input Mapping Context Blueprint mit Hilfe von selbst definierten Input Actions festgelegt. Dabei definieren Input Actions die Art des Inputs, z. B. ob es sich dabei um einen Boolean handelt, der zwischen true und false wechselt, je nachdem ob der Input gefeuert wurde oder nicht oder ob es ein Achsenwert ist, der je nach Dimensionalität 1 bis 3 Achsenwerte zwischen -1 und 1 beinhaltet. Den Input Actions wird anschließend im Input Mapping Context ein Controller Input zugewiesen, z.B. linker Controller Y-Achse des Joysticks. Für das Auslesen der Tracking-Daten werden Motion Controller Components verwendet, die ähnlich funktionieren wie die XR Controller Komponenten des XRTKs in Unity.

Sowohl in Unity als auch in Unreal ist der Zugriff auf die einzelnen Sensordaten möglich. Während in Unity die Daten sowohl per Komponenten als auch per Skript und Input Action Maps ausgelesen werden können, wird in Unreal alles unter einer Methode zusammengefasst, dem Input Mapping, egal welche Plattform im Endeffekt bedient wird.

4.3. Zusammenfassung

Die Ergebnisse dieser Analyse werden in der Tabelle 4.1 zusammengefasst. Darin wird dargestellt, welche der vorab etablierten Kriterien von den jeweiligen Engines erfüllt werden. Dabei stehen grün gefärbte Zellen für vollständig erfüllte Kriterien und gelb gefärbte Zellen für teilweise erfüllte Kriterien.

Kriterium	Unity	Unreal Engine 5
Es sind Templates für die Erstellung von VR-Projekten verfügbar.	erfüllt	erfüllt
Die VR-Templates enthalten alles, was für den Start in ein VR-Projekt benötigt wird.	teilweise erfüllt, Setup für Frameworks erforderlich	erfüllt
Die Engine kann die gängigsten Frameworks bzw. Plattformen bedienen.	Frameworks: erfüllt, Plattformen: erfüllt	Frameworks: nicht erfüllt, Plattformen: erfüllt
Die Engine kann die gängigsten VR-Hardwares bedienen.	erfüllt	erfüllt
Es können alle nötigen Sensor- und Inputdaten der Hardware aus der Engine heraus abgelesen werden.	erfüllt	erfüllt

Tabelle 4.1.: Ergebnisse des analytischen Vergleichs

5. Ermittlung der Einsteiger-freundlichkeit der Engines

Die vorhergegangene Analyse beschränkt sich auf die Sichtweise eines erfahrenen Entwicklers, der schon zuvor Projekte in Unity oder Unreal umgesetzt hat. Der Einstieg in VR und die Umsetzung eines solchen Projektes fällt daher meistens leichter, da man aus einem bereits vorhandenen Wissensschatz Schlüsse auf neue Themen ziehen kann. Darauf können sich Neueinsteiger nicht verlassen. Für sie ist es wichtig, ein System zu haben, welches das Umsetzen ihrer Vision so einfach wie möglich macht. Um diese Sichtweise besser zu verstehen, wurde ein Experiment konzipiert, bei dem Studierende, die zuvor wenig bis gar keine Erfahrung in Bezug auf Game Engines haben, eine kurze Einführung in diese bekommen, um anschließend das gleiche Projekt wie in Kapitel 3 umzusetzen, das zur Verifizierung der gelernten und umsetzbaren Inhalte dienen soll.

5.1. Teilnehmer des Experimentes

An dem Experiment beteiligten sich bis zum Schluss insgesamt acht Teilnehmer. Die Zielgruppe dieses Experiments sind Studierende, die wenig bis gar keine Erfahrung mit Game Engines haben. Zu diesem Zwecke wurden zwei Gruppen etabliert. Die eine Gruppe, bestehend aus fünf Studierenden aus dem Ingenieurswesen und einem Studierenden aus der Informatik, wurde innerhalb des Wahlpflichtkurses „Augmented und Virtual Reality“ untersucht. Da die am Anfang des Experimentes vermittelten Themen zu den Grundlagen der Engines bereits zum Teil für Unity Inhalt dieses Kurses sind, bot es sich an, den Lehrinhalt um die Grundlagen der Unreal Engine zu erweitern und die Teilnehmer des Kurses mit Pflichtanwesenheit für das Experiment

zu gewinnen. Das hat den Vorteil, dass diese Gruppe an Teilnehmern über die zwei Wochen des Experimentes konstant blieb. Allerdings ist es eine Pflichtteilnahme, wodurch die Motivation für das Experiment erst aufgebaut werden musste. Die andere Gruppe hingegen, bestehend aus Studierenden der Informatik, beteiligte sich in einem Workshop an der Untersuchung. Da die Teilnehmer der ersten Gruppe hauptsächlich aus Studierenden des Ingenieurswesens bestehen, wurden zusätzlich Informatikstudenten in das Experiment mit einbezogen. Dies diente nicht nur zur Erhöhung der Teilnehmerzahlen, sondern konnte auch genutzt werden, um eventuelle Vorteile durch vorherige Programmiererfahrungen festzustellen. Die Teilnahme der Informatikstudenten an dem Workshop war rein freiwillig. Das hatte den Vorteil, dass die Teilnehmer dieser Gruppe eine höhere Anfangsmotivation mitbrachten. Allerdings konnte die Anzahl an Teilnehmern nicht von Anfang bis Ende gehalten werden. So meldeten sich ursprünglich sechs Studierende aus der Informatik für das Experiment, davon erschienen allerdings nur vier zu den ersten Terminen. Bei der letzten Veranstaltung waren schließlich nur noch zwei Teilnehmer übrig. Folgende Informationen wurden zur genaueren Identifikation der Zielgruppen zu den einzelnen Teilnehmern erhoben:

- Das Alter bewegte sich zwischen 20 und 30, wobei sieben von acht Teilnehmern angaben, zwischen 20 und 25 Jahre alt zu sein.
- Es gab drei Teilnehmer, die sich der Informatikwissenschaften und fünf Teilnehmer, die sich den Ingenieurswissenschaften angehörig gefühlt haben.
- Sieben Teilnehmer haben vor dem Experiment einen Grundkurs für Programmierung besucht und drei Teilnehmer haben vor dem Experiment bereits an einer Lehrveranstaltung zum Thema Spieleentwicklung teilgenommen und somit schon ein wenig Erfahrung mit Game Engines gesammelt.
- Vier Teilnehmer gaben an, Vorkenntnisse zu Unity zu besitzen.
- Ein Teilnehmer besitzt Vorkenntnisse zur Unreal Engine 5.
- Ein Teilnehmer hat eine VR-Brille im privaten Besitz.
- Zwei Teilnehmer gaben an, kein Vorwissen zu den gefragten Themen zu besitzen.

Das Geschlecht soll für diese Betrachtung irrelevant sein. Im weiteren Verlauf der Arbeit wird auf Teilnehmer mit Programmiervorkenntnissen verwiesen. Diese setzen

sich aus den Personen zusammen, die vorher bereits einen Programmierkurs besucht oder Vorerfahrung in Unity / der Unreal Engine gesammelt haben.

5.2. Ablauf des Experimentes

Die Durchführung des Experimentes geschah parallel mit zwei Gruppen. Dies hat den Grund, dass eine Gruppe innerhalb eines Wahlpflichtkurses teilnahm, während die andere Gruppe dies zusätzlich außerhalb des besagten Kurses in einem Workshop unternahm. Innerhalb dieser zwei fachlichen Gruppen arbeiteten die Teilnehmer als Teams aus zwei Personen zusammen. Der Ablauf wurde allerdings bei allen Gruppen identisch gehalten. Das Experiment ging über zwei Wochen. Das Ziel der ersten Woche war es, in je zwei Unterrichtseinheiten von 90 Minuten zunächst die Grundlagen zu den beiden Engines an sich und anschließend die Grundlagen der VR-Entwicklung in den Engines zu vermitteln. Dies verfolgte den Zweck, dass alle Teilnehmer auf ein Grundverständnis für die Entwicklung von Anwendungen, insbesondere VR-Anwendungen, gebracht werden. In der zweiten Woche sollte dieses Grundverständnis nun angewendet werden. Die Teilnehmer bekamen die Aufgabe, sowohl in Unity als auch in der Unreal Engine 5 ein Minispiel umzusetzen. Dabei handelte es sich um das bereits im Kapitel 4 implementierte Spiel „Dosenwerfen“. Die entsprechenden Anforderungen an die Studierenden finden sich im Anhang A. Der grobe Ablauf ist grafisch in Abbildung 5.1 nachvollziehbar.

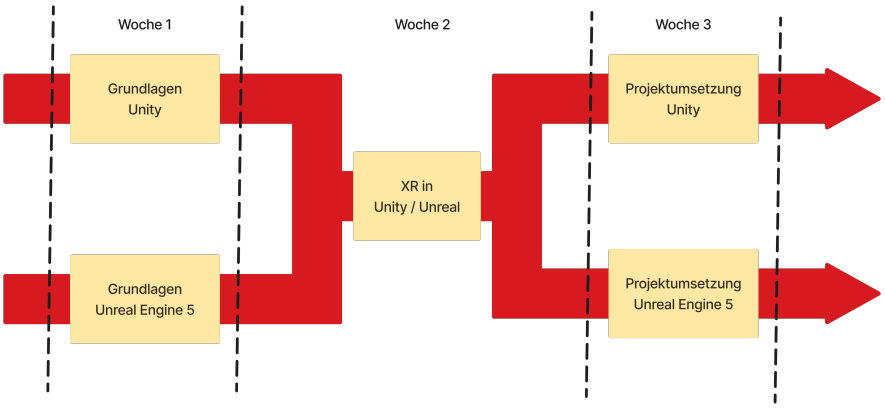


Abbildung 5.1.: Ablauf des Experimentes als Flowchart

5.2.1. Vorbereitung

In der Analyse in Kapitel 4 fiel auf, dass das VR-Template von Unity im Vergleich zur Unreal Engine ein viel komplexeres Setup benötigt, um auf ein Level mit den Inhalten des Templates der Unreal Engine zu gelangen. Diese Inhalte zusätzlich in der begrenzten Zeit zu vermitteln, wäre nicht möglich gewesen. Deshalb musste für das Experiment für Unity ein erweitertes Template geschrieben und zur Verfügung gestellt werden, welches grob denselben Stand wie das Unreal Engine 5 Template besitzt. Um dies zu erreichen, müssen folgende Schritte durchgeführt werden:

1. Installation des XRTKs
2. Einrichtung einer XR Origin Komponente
3. Controllerinteraktionen mit Hilfe von Ray Interactor
4. Teleportationsfortbewegung und Snap Turn Drehung
5. Erstellung einer Beispielszene mit einer Ebene als Boden und einer mit einer XR Grab Interactable greifbar gemachten Kugel.

5.2.2. Durchführung der Projekte

Die Durchführung der Projekte geschah jeweils immer an einem der zwei für das Experiment festgelegten Zeitslots. Es wurde in beiden Gruppen zunächst das Projekt in Unity umgesetzt und am Folgetag in der Unreal Engine 5. Pro Projekt hatten die Teilnehmer 80 Minuten Zeit, anschließend kreuzten sie auf dem Aufgabenblatt die geschafften Inhalte ab und beantworteten nach der Umsetzung mit der Unreal Engine einen Fragebogen zu ihrer gesamten Erfahrung über die zwei Wochen. Auch der Fragebogen ist im Anhang B zu finden. Während der Durchführung der Projekte war es den Teilnehmern möglich, sowohl einem anwesenden Dozenten als auch dem Autor Fragen zu stellen, sollten diese aufkommen. Zudem war die Benutzung des Internets erlaubt sowie jede hilfreiche Quelle innerhalb dessen.

5.2.3. Qualitative Beobachtungen

Zunächst bekamen die Teilnehmer eine Einführung in Unity und der Unreal Engine 5, sowohl Grundlagen als auch spezifisch VR-Development. Diese beinhaltete folgende Themen:

1. **Engine-Grundlagen**
    - Aufbau des Editors
    - Erstellung von Spielobjekten
    - Objekteigenschaften und deren Einstellung
    - Schreiben und Anwenden von Spiellogik
    - Klassenhierarchien
2. **VR-Grundlagen**
    - Entdecken des grundlegenden Aufbaus eines VR-Projektes
    - Komponenten zur Umsetzung grundlegender Funktionen wie Bewegung der Spielfigur
    - Funktionsweise des Trackings
    - Aufbau der Templates und ihrer vorgegebenen Funktionen

Die Themenwahl sollte dabei den Fokus auf die Engine-Grundlagen legen, damit die VR-Grundlagen durch eigenständiges Ausprobieren besser ergründet werden können. Da das Thema VR recht umfangreich ist, konnte nicht auf alle Grundlagen detailreich eingegangen werden. Nach diesen beiden Einführungsstunden wurde nach den ersten Eindrücken der Teilnehmer gefragt. Dabei merkten die Teilnehmer vor allem an, dass man zunächst bei beiden Engines die einzelnen Befehle und Funktionen der Engines lernen muss. Für die Unreal Engine 5 spezifisch wurde ausgesagt, dass die Visual Scripting Language Blueprint auf den ersten Blick recht einfach wirkt und einigen besser verständlich war als normale Codestrukturen. Trotzdem erschien Unity für die meisten Teilnehmer einsteigerfreundlicher, da Unreal mit der Menge an Möglichkeiten für Einsteiger überfordernd wirke.
Anschließend wurden die Projekte in Unity entwickelt. Bei der Ingenieursgruppe

konnte beobachtet werden, dass trotz der vorgefertigten Funktionen der Engine und des XRTKs viele dieser Funktionen selbst implementiert wurden, da die meisten Teilnehmer sich an deren Existenz nicht erinnerten und entsprechend nicht danach suchten. Nach einigen Hilfestellungen konnten dann Suchanfragen im Internet beobachtet werden, die den Teilnehmern Denkanstöße für gewisse Funktionen gaben. Weiterhin wurde in dieser Gruppe das Konzept der Prefabs nicht einmal verwendet, obwohl es sich z.B. für die Dosenpyramide angeboten hätte. Dabei handelt es sich bei Prefabs um Assets, die sowohl einzelne Spielobjekte als auch ganze Hierarchien von Spielobjekten mit allen an ihnen eingestellten Eigenschaften und Komponenten speichern. So müssen komplexe Assets wie Spielerfiguren oder bewegbare Elemente nicht von Hand immer neu in den Szenen zusammengebaut werden, sondern können einfach per Prefab direkt in die Szene gezogen werden. Aufgrund der wenigen Erfahrung mit Unity der meisten Teilnehmer der Ingenieurswissenschaften lassen sich diese Umstände erklären. Bei der Informatikergruppe sind die Grundverständnisse durch bereits erlangte grundlegende Erfahrung mit der Engine gegeben. Hier wurden von alleine Prefabs benutzt und auch die Implementierung der verschiedenen Funktionen, wie z.B. der Respawn von Ball und Pyramide, liefen weitaus schneller ab. Dadurch konnten diese Teams im Endeffekt mehr Funktionen in Unity umsetzen als die Teams der Ingenieure. Dabei kann die Motivation der Teilnehmer ebenfalls ausschlaggebend gewesen sein.

Nachdem die Gruppen das Unity Projekt abgeschlossen hatten, mussten sie das gleiche Projekt am nächsten Tag in der Unreal Engine 5 umsetzen. Dies sollte dazu dienen, dass die Gruppen in Unity zunächst ihre eigenen Erfahrungen sammeln konnten, um mit einem gewissen Grundverständnis das Projekt in Unreal angehen zu können, da diese Engine eine höhere Komplexität besitzt. Es fiel auf, dass das Arbeiten mit der Grundstruktur von Unreal schnell angewendet werden konnte. Obwohl die Objekteigenschaft „Simulate Physics“ im Theorieteil aus Zeitgründen nicht weiter erklärt wurde, haben alle Teilnehmer diese gefunden und sofort verwendet. Der ähnliche Aufbau wie Rigidbodies aus Unity unterstützte das Verständnis und die Anwendungsbereitschaft. Weiterhin wurden viele Assets aus dem Template wieder verwendet und angepasst, sodass die meisten Anforderungen sehr schnell umgesetzt werden konnten. Allerdings haben die meisten Teams bei den Bewegungsoptionen aufgegeben, da es dafür keine vorgefertigten Funktionen gab. Man musste teilweise

eigene Input Actions definieren, was für einige Teilnehmer zu komplex wurde. Hierbei fiel wieder auf, dass die Teilnehmer aus den Informatikwissenschaften die einzigen waren, die sich den Bewegungsoptionen überhaupt angenommen haben und dabei auch gute Resultate erzielten. Im Allgemeinen wurde die Blueprint-Language von der Mehrheit der Teilnehmer als schwieriger und komplexer bezeichnet, da es eine Vielzahl unbekannter Funktionen gab, die erst erforscht werden mussten. Allerdings waren vor allem die Teilnehmer, die vorher kaum bis gar nichts mit gängigen Programmiersprachen zu tun hatten, sehr gewillt, die Blueprint-Language zu lernen. Während der Implementierung der Input Action Mappings wurde bei einer Gruppe ein Problem mit der HP Reverb festgestellt. Als diese die Inputs für die Continuous Locomotion definieren wollten, wurden die Input Mappings von Mixed Reality nicht erkannt. Bei den vordefinierten Funktionen war dies jedoch der Fall. Daraufhin wurde das HP Motion Controller Input Mapping Plugin installiert, was speziell für die HP Reverb Input Mappings hinzufügt. Nach der Installation wurde der Input mit den neuen Mappings erkannt. Daraus resultierend musste dann allen vordefinierten Input Mappings das HP Controller Mapping hinzugefügt werden, da die Mixed Reality Mappings hier plötzlich aufhörten zu funktionieren.

5.2.4. Quantitative Auswertung

Nach der Umsetzung in der Unreal Engine 5 wurde allen Teilnehmern ein Fragebogen ausgehändigt. Für das Ausfüllen hatten diese dann 10 Minuten Zeit. Die Anzahl der Teilnehmer an diesem Experiment ist zu gering für eine aussagekräftige statistische Analyse. Dennoch wurden die Ergebnisse der Fragebögen auf einer quantitativen Basis ausgewertet, um mögliche Tendenzen zu erkennen.
Bei der Frage, welche Engine für die Teilnehmer einfacher war, antworteten fünf Personen, dass sie Unity als einfacher empfanden. Dabei gaben sie als Grund an, dass sie durch ihr Vorwissen von Unity bzw. vom Programmieren profitierten und somit mit dieser Engine besser zurechtkamen. Als Gegenargumente zur Unreal Engine 5 nannte man dabei grundlegend die Fülle der Funktionen, die einem dort zur Verfügung stehen. Diese „erschlage“ zu Beginn. Das traf für diese Personen auch besonders auf die Blueprint-Language zu. Wiederum erklärten die drei Personen, die sich für die Unreal Engine 5 entschieden haben, dass die Blueprint-Language

zwar etwas mehr Gewöhnung braucht, im Endeffekt allerdings einfacher sei als das Unity Scripting. Dabei ist auffällig, dass diese Aussagen nur von Teilnehmern aus den Ingenieurswissenschaften getätigt wurden, besonders von denen, die kaum Vorwissen zum Programmieren haben. Weiterhin empfanden diese, dass die vom Template gestellten Beispiele und Funktionen in der Unreal Engine sehr hilfreich und nützlich waren. Auch die Dokumentation der Unreal Engine wurde hier gelobt. Diese ist umfangreicher und verständlicher als die von Unity und half somit beim Verstehen der einzelnen Funktionen.
Die darauf folgende Frage zielte auf die schwierigsten Inhalte des Projekts ab. Dort waren die Meinungen der einzelnen Teilnehmer recht unterschiedlich. Fünf Personen waren mit der Umsetzung des Inputs überfordert, sowohl in Unity als auch in Unreal. Zwei Personen hatten Schwierigkeiten, den Reset des Balls und der Pyramiden umzusetzen. Vor allem wurde von fünf Personen das Einstellen der Eigenschaften von Objekten und Komponenten sowohl in Unity als auch in Unreal kritisiert. Die Menge an Optionen macht es unübersichtlich und es ist schwer erkennbar, wo welches Feature eingestellt werden muss.
Danach ging es um die Frage, mit welcher Engine die Teilnehmer in Zukunft weiterarbeiten würden und weshalb. Hier gaben wieder fünf Personen Unity an. Dabei waren die Gründe dafür häufig, dass die Teilnehmer lieber in C# Unity Skripte schreiben, als mit der Blueprint-Language in Unreal zu arbeiten. Zwei Personen gaben auch an, dass sie zunächst in Unity ihre Programmierkenntnisse verbessern wollen, um im Allgemeinen ein besseres Verständnis zu erlangen. Danach könne man sich die Arbeit mit der Unreal Engine vorstellen. Für die Unreal Engine sprachen sich hier nur drei Personen aus. Es wurde angegeben, dass zwar die Einarbeitungszeit höher sei als bei Unity, danach allerdings die einzelnen Prozesse schneller und angenehmer ablaufen. Sehr interessant hierbei war, dass besonders die Teilnehmer, die noch keine Programmiererfahrung haben, lieber mit Unreal weiterarbeiten würden als mit Unity, denn sie empfanden die Blueprint-Language einfacher als das Programmieren. Die Visual Scripting Language scheint eher attraktiv für Entwickler ohne große Programmiererfahrung zu sein, da diese sich nicht vom klassischen Programmieren umgewöhnen müssen.
Die letzte Frage bezog sich darauf, welches Thema für die Teilnehmer insgesamt am interessantesten war. Dabei war die Einstellung der Objektphysik für zwei Per-

sonen ein interessantes Thema. In VR wird diese vollkommen anders erlebt als in einem normalen Videospiel. Dies machte die Arbeit damit in dem Kontext deutlich wichtiger und spannender. Für drei Personen war es verblüffend, wie einfach die Umsetzung eines VR-Projektes im Allgemeinen ist, egal in welcher Engine. Gerade das Tracking der Hardware stellten sie sich weitaus schwieriger vor, als es anschließend wahrgenommen wurde.

5.3. Zusammenfassung

Generell ergab das Experiment, dass Unity sich besonders gut für Einsteiger in die VR-Entwicklung eignet, die bereits Kenntnisse im Programmieren besitzen. Die Nähe zum klassischen Programmieren vereinfacht ihnen die Arbeit mit der Engine. Die Unreal Engine 5 dagegen eignet sich besonders gut als Einstieg für Personen ohne ausgeprägte Programmierkenntnisse. Mit seiner Blueprint Visual Scripting Language erzeugt es durch das simple Graphenprinzip eine Abstrahierung des klassischen Programmierens. Dadurch ist der Umgang mit dieser Programmiersprache intuitiver für diese Personen.

6. Ergebnisse

Im Folgenden werden die Ergebnisse des analytischen Vergleichs und des Experimentes ausgewertet und zu jedem eine Schlussfolgerung gezogen.
Im analytischen Vergleich wurden zunächst Vergleichskriterien erarbeitet, anhand derer anschließend der Vergleich vollzogen wurde. Dabei waren die Unreal Engine 5 und Unity gleichwertig in folgenden Punkten:

- Erstellung von Projekten mit Hilfe eines Templates
- Unterstützte Frameworks / Plattformen
- Kompatible Brillenhardware.

Lediglich in drei Punkten unterscheiden sich die Engines stark:

1. **Das Template**
   In der Unreal Engine 5 wird in dem VR-Template alles an Grundlagen vordefiniert, was man benötigt, um ein VR-Projekt umzusetzen. Somit müsste man sich zu Beginn nicht mit den typischen Anfangshürden auseinandersetzen, wie z.B. Bewegungssteuerung. Allerdings ist das Anpassen dieser vordefinierten Funktionen recht mühsam und schwierig, da es sich bei den meisten Funktionen nicht um vorgefertigte Komponenten handelt. Dadurch muss alles, was an diesen Funktionen geändert werden soll, selbst implementiert werden, z. B. alternative Optionen zum Bewegen der Spielfigur. In dieser Hinsicht hat Unity den Vorteil, dass nach Installation des XRTKs eine Vielzahl von optionalen Features per Komponenten mit wenigen Handgriffen hinzugefügt werden kann.

2. **Frameworks**
   Während in Unity eine Vielzahl kostenloser SDKs zur Verfügung steht, besitzt Unreal keine SDKs für die Entwicklung von VR-Projekten. Dies führt dazu, dass die meisten VR-Funktionen, wie die Locomotion, selbst implementiert werden müssen.

3. **Auslesen der Sensor- und Inputdaten**
   In Unity gibt es eine Vielzahl von Möglichkeiten, wie z.B. das Tracking der Controller per Komponente oder Auslesen von Input per Skript. Dagegen setzt Unreal auf sein in der 5.1 hinzugefügtes Enhanced Input System, in dem mit Hilfe von Input Actions Arten von Aktionen definiert werden, um diesen im Anschluss in einem Input Mapping Context ein konkretes Controllermapping zuzuweisen. Allerdings kann das recht kompliziert werden und bei manchen Brillen zu Problemen führen, wie z.B. bei der HP Reverb, welche im Experiment verwendet wurde und ein eigenes Plugin brauchte, um in selbst definierten Input Actions erkannt zu werden.

Die Unreal Engine 5 vereinfacht den Start in ein VR-Projekt deutlich stärker als Unity, dies trifft allerdings nur zu, wenn man an den vordefinierten Funktionen nichts ändern möchte. Dagegen kann man in Unity mit einem etwas längeren, aber simplen Setup seine Projekte individueller konfigurieren. Im Endeffekt eignet sich die Unreal Engine 5 für erfahrene Entwickler, die größere VR-Projekte umsetzen möchten, während Unity sich perfekt für Einsteiger eignet, die ein Projekt vorerst prototypisch oder generell eher kleinere Projekte umsetzen möchten. Diese Ergebnisse werden in der folgenden Tabelle 6.1 zusammengefasst. Dabei sind die einzelnen Zellen mit den jeweiligen Eigenschaften der Engine teilweise rot oder grün gefärbt. Eine rote Zellenfärbung steht für eine schlecht umgesetzte oder zu komplexe Eigenschaft, während grün für einfache und verständliche Eigenschaften steht.

	Unity	**Unreal Engine 5**
Template-Inhalt	Headset- und Controllertracking	Headset- und Controllertracking, Spielercharakter mit grundlegender Steuerung für Drehen und Bewegen, vorgefertigtes UI am Controller, Komponenten zum Greifbarmachen von Objekten, Spectatorkamera
Kompatibilitäten	Mixed Reality, SteamVR, Oculus, HoloLens	Mixed Reality, SteamVR, Oculus, HoloLens, HP Reverb Brillen gesondert durch Plugin
SDKs	Verschiedene: Virtual Reality Toolkit, Windows Mixed Reality Toolkit, Mixed Reality Toolkit, XR Interaction Toolkit u.v.m.	keine, das meiste ist im Template vorgefertigt oder muss selbst implementiert werden
Art des Auslesens der Sensor- und Inputdaten	Input Action Mapping, Auslesen im Skript, Komponenten der SDKs	Input Action Mapping, Auslesen im Skript

Tabelle 6.1.: Ergebnisse des analytischen Vergleichs

In dem Experiment sollte ermittelt werden, wie anfängerfreundlich die jeweiligen Engines mit ihren Templates sind. Dabei war das Feedback zum größten Teil recht eindeutig. Unity hat den meisten für den Anfang besser gefallen und sie würden diese Engine auch weiterhin verwenden. Diese Erkenntnis gewannen vor allem Teilnehmer, welche bereits vorher Erfahrung mit Unity oder mit Programmieren im Allgemeinen gesammelt haben. Interessant zu beobachten war allerdings, dass Teilnehmer, die vorher kaum bis gar keine Programmiererfahrung gesammelt haben, besser mit dem Blueprint-System von Unreal umgehen konnten als die anderen Teilnehmer.
Die Unreal Engine 5 mit ihrer Visual Scripting Language Blueprint eignet sich besser als Einstieg, wenn einem das Programmieren schwerfällt bzw. wenn man damit noch keinerlei Erfahrung gesammelt hat, da das Verbinden der Nodes miteinander ein greifbareres Konzept darstellt als pures Code Scripting. Unity eignet sich für Menschen mit Programmiererfahrung besser, die z. B. erst in das Thema Game Development einsteigen. Sie sind mit dem Schreiben von Programmcode vertraut, deshalb fällt es ihnen leichter, Skripte zu schreiben als mit grafischen Programmiersprachen umzugehen. Das erfordert ein starkes Umdenken, welches über die Zeit antrainiert werden muss. Als Ergebnis des Experiments kann festgehalten werden, dass sich Unity gut für Programmierer eignet, die noch keine Erfahrungen in Bezug auf den Umgang mit Game Engines besitzen. Dagegen ist die Unreal Engine 5 für Menschen ohne Programmiererfahrung und Programmierern mit Erfahrungen in Bezug auf den Umgang mit Game Engines zu empfehlen.

7. Fazit und Ausblick

Sowohl Unity als auch die Unreal Engine 5 sind gut für die Umsetzung von VR-Projekten geeignet. Beide besitzen Vor- und Nachteile, die in der Auswahl der Engine beachtet werden müssen. Die eindeutige Empfehlung hier aber lautet:

Unity eignet sich besonders gut für Einsteiger und / oder kleinere Projekte bzw. Projektprototypen, da mit Hilfe der SDKs recht schnell ein umfangreiches Grundkonstrukt geschaffen werden kann. Die Unreal Engine 5 empfiehlt sich vor allem eher für erfahrene Game Developer oder mehr designorientierte Entwickler, da man in dieser Engine sowohl über C++ bekannte Skriptstrukturen erstellen als auch mit der Visual Scripting Language Blueprint das gewohnte Schreiben von Programmcode abstrahieren kann. Durch zusätzliche Features wie das Nanite-System[34][1] und der Quixel Bridge[35][2] können größere Projekte mit weniger Zeitaufwand und besserer Performance erstellt werden. So empfiehlt sich, mit der Unreal Engine 5 vor allem größere und komplexere Projekte umzusetzen, die grafisch anspruchsvoll sein sollen.

[1] Das Nanite-System ist ein Rendering-System der Unreal Engine 5 zur dynamischen Verringerung von Meshdetails in Abhängigkeit von der Distanz zum Objekt.

[2] Die Quixel Bridge ist eine große Bibliothek an hoch detaillierten eingescannten 3D-Objekten, die für alle Nutzer mit einem EpicGames-Account oder ähnlichem frei nutzbar ist.

7.1. Limitierungen

Während der Durchführung der beschriebenen Arbeitsschritte sind einige Probleme aufgetreten, die innerhalb des gegebenen Zeitkontingents nicht lösbar waren und deshalb in Kauf genommen werden mussten. Als Erstes ist anzumerken, dass die Hardwarekompatibilität bei den meisten Brillen nicht getestet werden konnte. Es wurden nur die Oculus Rift S und die HP Reverb getestet. Für den Rest wurde sich auf die Aussagen der Dokumentationen der jeweiligen Engines gestützt, welche Angaben über die unterstützen Plattformen machen. Zudem wurde auf andere wichtige Kriterien verzichtet. So war zunächst die Performance als weiteres Kriterium im Vergleich angedacht, was allerdings aufgrund des simplen Szenenaufbaus nicht vergleichbar gewesen wäre. Dafür hätte eine große, komplexe Szene geschaffen werden müssen, was zeitlich nicht gepasst hätte.
Während des Experiments ergaben sich einige Bedingungen, die nicht optimal waren. Zum einen war die Anzahl an Teilnehmern sehr gering, weshalb die Auswertung der Ergebnisse auf einer qualitativen Basis geschehen musste, da keine stochastisch signifikante Teilnehmerzahl erreicht werden konnte. Insgesamt nahmen nur acht Personen an der Studie teil, wobei sechs davon in einer Vorlesung und zwei in einem Workshop separat betreut werden mussten, da die Teilnehmer aus verschiedenen Fachbereichen kamen und zeitlich nicht kompatibel waren. Auch die Durchführung war aus zeitlichen Gründen nicht optimal. Es wäre besser gewesen, für dieses Unterfangen alle Teilnehmer über ein gesamtes Semester hinweg im Umgang mit Unity und der Unreal Engine 5 zu unterrichten, um ein fundiertes Grundverständnis für beide Engines zu haben. Mit einer Woche Vorbereitungszeit mit je zwei 90 Minuten langen Unterrichtseinheiten konnten die Einstiegshürden der Engines nicht vollständig überwunden werden, weshalb bei der Umsetzung einige Punkte scheiterten, da Teilnehmer gewisse Standardfunktionen der Engines noch nicht beherrschten. In der erweiterten Vorbereitungsphase hätte man dann auch Unreal C++ Skripte behandeln können, um die Vergleichbarkeit mit Unity C#-Skripten herstellen zu können. Bei der Durchführung der Projekte musste aufgrund der Teilnehmerzahl auf thematisch getrennte Gruppen verzichtet werden. Das heißt, dass alle Teilnehmer mit beiden Engines arbeiten mussten. Auch wenn sich Unity von der Unreal Engine 5 deutlich unterscheidet, so können mit ihr Grunderkenntnisse im Prozess der Spieleentwicklung

erlangt werden. Diesen Vorteil nahmen die Teilnehmer dann mit in die Unreal Engine. Auch Strategien zur Umsetzung des Vergleichsprojektes konnten in die zweite Runde mitgenommen werden, da die Ziele gleich blieben. Dadurch geht bei der Umsetzung des zweiten Projektes unweigerlich der Aspekt des Ersteinstiegs verloren.
Man kann also abschließend sagen, dass in der gesamten Arbeit nur die Sichtweisen zweier Zielgruppen repräsentiert wurden: einmal die Sichtweise eines erfahrenen Programmierers mit erweiterter Kenntnis sowohl in Unity als auch in der Unreal Engine 5 und einmal die Sichtweise von Anfängern, die zuvor keine oder kaum Erfahrung mit Game Engines oder Programmierung gesammelt haben.

7.2. Lösungsansätze und Verbesserungen

Zunächst könnte man im Teil des analytischen Vergleichs einen umfangreicheren Test mit der Hardwarekompatibilität durchführen. Schon bei der HP Reverb zeigte sich in der Unreal Engine 5, dass das Auslesen der Inputdaten mit Schwierigkeiten verbunden sein kann, z.B. durch Benötigen eines zusätzlichen Plugins. Außerdem könnte das bereits besprochene fehlende Kapitel zur Performance abgedeckt werden.
Am Experiment können auch einige Verbesserungen vorgenommen werden. So könnte eine größere Gruppe von Teilnehmern gesammelt werden, mit der über längere Zeit der grundlegende Umgang mit beiden Engines erarbeitet wird. Dabei hätten die Teilnehmer in zwei Gruppen unterteilt werden müssen, wobei die eine Gruppe sich ausschließlich mit Unity und die andere mit der Unreal Engine 5 beschäftigt. Dadurch wird die Menge an Informationen für den einzelnen Teilnehmer halbiert und die Umsetzung der Projekte würde nur einmal stattfinden, sodass Erfahrungen aus einer vorherigen Umsetzung nicht gegeben sind. Daraus ergäbe sich ein noch besseres Bild über die Eignung und Nutzerfreundlichkeit der Engines im Bezug auf die Umsetzung von VR-Projekten. Weiterhin könnten bei der Projektumsetzung eine größere Anzahl an Teilnehmern hinzugezogen werden, die einen hohen Wissensschatz zur jeweiligen Engine besitzen, z. B. durch berufliche oder akademische Erfahrung. Das führt zu einer größeren Diversität der Zielgruppen, sodass eine allgemeingültigere Aussage zur Effektivität der jeweiligen Engine in Bezug auf die Umsetzung von VR-Projekten getätigt werden könnte.

Literaturverzeichnis

[1] U. Technologies, ,,Software für die VR AR-Spieleentwicklung," https://unity.com/de/solutions/vr, Zugegriffen: 25.07.2023.

[2] U. Technologies, ,,Unity 2020.3 Documentation Manual: XR," https://docs.unity3d.com/2020.3/Documentation/Manual/XR.html, Zugegriffen: 03.04.2023.

[3] P. Rauschnabel, R. Felix, C. Hinsch, H. Shahab, und F. Alt, ,,What is XR? Towards a Framework for Augmented and Virtual Reality," *Computers in Human Behavior*, Vol. 133, 2022.

[4] M. Billinghurst und H. Kato, ,,Collaborative Augmented Reality," *Communications of the ACM*, Vol. 45, Nr. 7, 2003, S. 64 - 70.

[5] K. Steuerwald, ,,Mit der IKEA App per Augmented Reality einrichten," https://www.ikea.com/de/de/this-is-ikea/corporate-blog/ikea-place-app-augmented-reality-puba55c67c0, Zugegriffen: 27.06.2023.

[6] Microsoft, ,,Dynamics 365 Remote Assist: Zusammenarbeit – von jedem Ort," https://dynamics.microsoft.com/de-de/mixed-reality/remote-assist/, Zugegriffen: 27.06.2023.

[7] G. Riva, ,,Virtual Reality in Psychotherapy: Review," *Cyberpsychology behavior : the impact of the Internet, multimedia and virtual reality on behavior and society*, Vol. 8, Nr. 3, 2005, S. 220 - 230; discussion 231.

[8] A. Klippel, ,,Immersion, Presence, and Interaction," https://www.e-education.psu.edu/geogvr/node/875, Zugegriffen: 15.07.2023.

[9] S. Finkelstein, A. Nickel, Z. Lipps, T. Barnes, Z. Wartell, und E. Suma Rosenberg, ,,Astrojumper: Motivating Exercise with an Immersive Virtual Reality Exergame," *Presence*, Vol. 20, Nr. 1, 2011, S. 78 - 92.

[10] W. R. Sherman und A. B. Craig, ,,Chapter 1 - Introduction to Virtual Reality, Chapter 2 - VR: The Medium," in *Understanding Virtual Reality (Second Edition)*, Second Edition Aufl., Serie The Morgan Kaufmann Series in Computer Graphics, W. R. Sherman und A. B. Craig, Hrsgg. Boston: Morgan Kaufmann, 2019, S. 4 - 100. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128009659000027

[11] M. Slater und S. Wilbur, ,,A Framework for Immersive Virtual Environments (FIVE): Speculations on the Role of Presence in Virtual Environments," *Presence: Teleoperators & Virtual Environments*, Vol. 6, Nr. 6, 1997, S. 603 - 616.

[12] J. J. Cummings und J. N. Bailenson, ,,How Immersive Is Enough? A Meta-Analysis of the Effect of Immersive Technology on User Presence," *Media Psychology*, Vol. 19, Nr. 2, 2016, S. 272 309.

[13] J. Bailenson, *Experience on Demand: What Virtual Reality Is, How It Works, and What It Can Do*, 2018. [Online]. Available: https://books.google.de/books?id=2fkqDwAAQBAJ

[14] C. Fabris, J. Rathner, A. Fong, und C. Sevigny, ,,Virtual Reality in Higher Education," *International Journal of Innovation in Science and Mathematics Education*, Vol. 27, Nr. 8, 2019, S. 69 - 80.

[15] I. Strand, ,,Virtual Reality in Design Processes: - a literature review of benefits, challenges, and potentials," *FormAkademisk - forskningstidsskrift for design og designdidaktikk*, Vol. 13, Nr. 6, 2020.

[16] C. Bohil, B. Alicea, und F. Biocca, ,,Virtual reality in neuroscience research and therapy," *Nature reviews. Neuroscience*, Vol. 12, 2011, S. 752 - 762.

[17] J. Gregory, *Game engine architecture*, CRC Press, Boca Raton, Fla. [u.a.], 2. Aufl., 2014.

[18] N. Sharma, ,,Identification and evaluation of alternatives to Unity which are not under U.S. Sanction lists," https://www.hs-anhalt.de/fileadmin/Dateien/FB6/personen/tuemler_j/ProjectReport_Nipun_Sharma.pdf, Zugegriffen: 15.07.2023.

[19] U. Technologies, ,,Echtzeit-Tools und mehr - Machen Sie mehr mit Unity," https://unity.com/de, Zugegriffen: 27.06.2023.

[20] H. Koranne, ,,History of Unity3D," https://www.linkedin.com/pulse/history-unity3d-harsh-koranne/, Publikation: 13.04.2021. Zugegriffen: 27.06.2023.

[21] U. Technolgies, ,,Unity 2020.3 Documentation Manual," https://docs.unity3d.com/2020.3/Documentation/Manual/, Zugegriffen: 15.06.2023.

[22] U. Technologies, ,,Unity Personal," https://unity.com/products/unity-personal, Zugegriffen: 16.07.2023.

[23] U. Technologies, ,,Unity Personal," https://unity.com/compare-plans, Zugegriffen: 16.07.2023.

[24] E. Games, ,,Unreal Engine 5.0 Documentation," https://docs.unrealengine.com/5.0/en-US/ Zugegriffen: 15.06.2023.

[25] E. Games, ,,Unreal® Engine End User License Agreement," https://www.unrealengine.com/de/eula-reference/unreal-de, Zugegriffen: 16.07.2023.

[26] Y. Farmani und R. Teather, ,,Viewpoint Snapping to Reduce Cybersickness in Virtual Reality," in *Proceedings of the 44th Graphics Interface Conference (GI '18). Canadian Human-Computer Communications Society*, Waterloo, CAN, 2018, S. 168 - 175.

[27] A. Prithul, I. B. Adhanom, und E. Folmer, ,,Teleportation in Virtual Reality; A Mini-Review," *Frontiers in Virtual Reality*, Vol. 2, 2021. [Online]. Available: https://www.frontiersin.org/articles/10.3389/frvir.2021.730792

[28] U. Technologies, ,,Unity 2020.3 Documentation Manual: XR packages," https://docs.unity3d.com/2020.3/Documentation/Manual/xr-support-packages.html, Zugegriffen: 03.04.2023.

[29] E. Games, ,,Unreal Engine 5.0 Documentation: Supported XR Devices," https://docs.unrealengine.com/5.0/en-US/supported-xr-devices-in-unreal-engine/, Zugegriffen: 03.04.2023.

[30] B. Lang, ,,Meta Raises Quest 2 Price to Stave off Growing Costs," https://www.roadtovr.com/wp-content/uploads/2022/04/meta-quest-2-logo.png, Zugegriffen: 05.07.2023.

[31] HP, ,,HP Reverb G2 VR Headset," https://www.hp.com/de-de/shop/Html/Merch/Images/1N0T5AA-ABD_500x367.jpg, Zugegriffen: 05.07.2023.

[32] Amazon, ,,HTC VIVE VR Videobrille," https://m.media-amazon.com/images/I/4162+eWwY7L._AC_UF894,1000_QL80_.jpg, Zugegriffen: 05.07.2023.

[33] V. Corporation, ,,Valve Index," https://cdn.akamai.steamstatic.com/valvesoftware/images/index/videos/Main.jpg, Zugegriffen: 05.07.2023.

[34] E. Games, ,,Nanite Virtualized Geometry," https://docs.unrealengine.com/5.0/en-US/nanite-virtualized-geometry-in-unreal-engine/#:~:text=Nanite%20is%20Unreal%20Engine%205%27s,be%20perceived%20and%20no%20more., Zugegriffen: 10.07.2023.

[35] Quixel, ,,Bridge - Your gateway to Megascans and Metahumans," https://quixel.com/bridge, Zugegriffen: 10.07.2023.

A. Anforderungen Projekt „Dosenwerfen“

Anforderungen Projekt „Dosenwerfen“

Aufgabenstellung:

Es soll ein VR-Minispiel in Form von Dosenwerfen implementiert werden. Dabei sollen folgende Mindestanforderungen an das Spiel beachtet werden:

1. Der Spieler muss in der Lage sein, sich zu bewegen (Drehen, Laufen, Arme bewegen)
2. Der Spieler muss in der Lage sein zu jeder Zeit einen Ball greifen zu können
3. Der Spieler muss in der Lage sein diesen Ball auf eine „Dosenpyramide“ zu werfen und diese umwerfen zu können
4. Der Spieler muss in der Lage sein können, die „Dosenpyramide“ wieder in ihren Ursprungszustand zurückzusetzen, um eine weitere Runde Dosenwerfen zu spielen
5. Die Bewegung (auch Locomotion genannt) vom Spieler soll folgende Optionen enthalten:
    a. Drehen: Snap Turn (Der Spieler dreht sich mit jedem Betätigen des Joysticks um einen festgesetzten Winkel um die eigene Achse) oder Continuos Turn (Der Spieler kann den Joystick in eine Richtung gedrückt halten und dreht sich flüssig in diese Richtung um die eigene Achse, bis der Joystick wieder in der neutralen Position ist)
    b. Laufen: Teleport (Der Spieler kann mit seinem Controller auf eine Stelle des Spielfelds zeigen und wird dorthin teleportiert) und Continuos Locomotion (Der Spieler bewegt sich mit Hilfe des Joysticks flüssig in alle Richtungen. Es ist unerheblich, ob die Richtung dabei vom Headset oder den Controllern bestimmt wird.)

Folgende Anforderungen sind ausdrücklich **nicht** Teil der Anforderungen und rein optional:

- Sounddesign
- UI Design
- Umgebungsgestaltung / Level Design

Anforderungen an das Projekt:

1. Das Projekt muss sowohl in Unity als auch in der Unreal Engine 5 umgesetzt werden
2. Alle vorimplementierten Features, die durch die Templates, Engines und den SDKs zur Verfügung gestellt werden, dürfen so genutzt werden. Es müssen keine bereits vorhandenen Features neu implementiert werden

B. Fragebogen

Fragebogen zur Studie und Umsetzung des Projekts „Dosenwerfen“ in Unity und Unreal Engine 5

Dieser Fragebogen dient zur besseren Evaluierung Ihrer Erfahrungen, die Sie während des Entwickelns Ihrer Projekte gemacht haben. **Bitte kreuzen Sie bei allen Fragen mit Auswahl, bei denen nichts anderes vermerkt ist, nur eine Antwortmöglichkeit an.** Vielen Dank für das Teilnehmen an dieser Studie.

Sollten Sie im Anschluss an die Studie noch Fragen zu den behandelten Themen haben, können Sie mich gerne folgendermaßen erreichen:

Per Email: Jonas.Bongartz@student.hs-anhalt.de

Per Discord: GethPrime89#6382

Daten zur Person

Welcher Wissenschaft fühlen Sie sich durch Ihren Studiengang zugeordnet?

- ○ Ingenieurswissenschaften
- ○ Informatikwissenschaften

Welcher Altersgruppe gehören Sie an?

- ○ 20-25
- ○ 25-30
- ○ 30-35
- ○ 35-40

Welches Vorwissen bringen Sie mit? (Mehrfachantwort möglich)

Lehrveranstaltungen (Bitte nur abgeschlossene Lehrveranstaltungen ankreuzen!)

- ○ Grundkurs Programmierung
- ○ Spieleentwicklung
- ○ VR-/AR-Entwicklung

Allgemeines Vorwissen

- ○ Privater Besitz eines VR-Headsets
- ○ Unity Kenntnisse
- ○ Unreal Engine 4 Kenntnisse
- ○ Unreal Engine 5 Kenntnisse

Fragen zum Projekt

Welche der Engines empfanden Sie einfacher?

○ Unity

○ Unreal Engine 5

Begründen Sie Ihre Auswahl!

Welchen Teil des Projektes empfanden Sie am schwierigsten? Begründen Sie!

Mit welcher Plattform würden Sie in Zukunft gerne weiterarbeiten?

○ Unity

○ Unreal Engine 5

Begründen Sie Ihre Auswahl!

Welches der Themen hat Sie besonders interessiert? Begründen Sie!