Philipps-Universität Marburg

Fachbereich 12 – Mathematik Informatik

Bachelorarbeit

Konfigurierbares VR-Vortragstraining mittels Unreal Engine und MetaHumans

Von
Osman Yozgat
3554294

aus
Biedenkopf

Betreuer:
Prof. Dr. Thorsten Thormählen

Arbeitsgruppe Grafik und Multimedia Programmierung

Erklärung

Ich, Osman Yozgat (Informatikstudent an der Philipps-Universität Marburg, Matrikelnummer 3554294), versichere an Eides statt, dass ich die vorliegende Bachelorarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe. Die hier vorliegende Bachelorarbeit wurde weder in ihrer jetzigen noch in einer ähnlichen Form einer Prüfungskommission vorgelegt.

Zusammenfassung

In dieser Bachelorarbeit wurde eine umfassende VR-Anwendung entwickelt, die Unreal Engine 5 und MetaHuman-Technologie integriert, um eine immersive und interaktive Lernumgebung zu schaffen. Der Fokus lag auf der Erstellung von realitätsnahen Leveln und der Implementierung fortschrittlicher Animationen für MetaHumans, um eine hohe Immersion zu erreichen.

Im Rahmen der Arbeit wurden verschiedene Level erfolgreich erstellt, wobei visuelle und akustische Elemente sorgfältig integriert wurden. Die Störanimationen wurden durch eine benutzerfreundliche Oberfläche im VR-Umfeld konfigurierbar gemacht. Die Interaktion über VR-Controller ermöglichte eine intuitive Steuerung der MetaHuman-Modelle und deren Animationen, was zur Realitätsnähe der Anwendung beigetragen hat.

Die Implementierung der Animationen und die Steuerung über die VR-Interface wurden erfolgreich getestet. Die Ergebnisse zeigen, dass die Anwendung die definierten Anforderungen erfüllt, eine stabile Leistung bietet und eine klare, benutzerfreundliche Oberfläche aufweist. Die Rückmeldungen von Testnutzern bestätigen die Effektivität und Benutzerfreundlichkeit der Anwendung.

Für zukünftige Arbeiten werden Erweiterungen wie die Integration von Sprachanalyse und erweiterte Interaktionen durch KI-basierte Systeme in Betracht gezogen, um die Immersion und Benutzererfahrung weiter zu verbessern. Die kontinuierliche Weiterentwicklung der verwendeten Technologien, wie der Unreal Engine und MetaHumans, bietet zusätzliche Potenziale für die zukünftige Optimierung und Erweiterung der Anwendung.

abstract

This bachelor thesis developed a comprehensive VR application that integrates Unreal Engine 5 and MetaHuman technology to create an immersive and interactive learning environment. The focus was on creating realistic levels and implementing advanced animations for MetaHumans to achieve high immersion.

The work successfully created various levels, integrating visual and auditory elements with care. Disruption animations were made configurable through a user-friendly interface in the VR environment. Interaction through VR controllers allowed intuitive control of MetaHuman models and their animations, contributing to the realism of the application.

The implementation of animations and control through the VR interface was successfully tested. Results show that the application meets the defined requirements, provides stable performance, and features a clear, user-friendly interface. Feedback from test users confirms the effectiveness and usability of the application.

Future work will consider extensions such as integrating speech analysis and advanced interactions through AI-based systems to further enhance immersion and user experience. The continuous development of the technologies used, such as Unreal Engine and MetaHumans, offers additional potential for future optimization and expansion of the application.

Inhaltsverzeichnis

1. Einleitung

Diese Bachelorarbeit befasst sich mit der Entwicklung einer virtuellen Welt mittels Unreal Engine, in der MetaHumans als realitätsnahe, virtuelle Menschen agieren. Ziel ist es, dem Benutzer ein immersives Erlebnis zu bieten, sodass er das Gefühl hat, sich tatsächlich in der simulierten Umgebung zu befinden. Diese virtuelle Welt soll es dem Benutzer ermöglichen, Vorträge in einer realitätsnahen Umgebung zu simulieren, um so Ängste zu überwinden oder Fähigkeiten zu verbessern.

Im Rahmen dieser Arbeit wurden verschiedene Umgebungen entwickelt, die typischerweise im Alltag vorkommen, wie beispielsweise ein Meetingraum, ein Hörsaal oder ein Klassenzimmer. Besonderes Augenmerk wurde daraufgelegt, diese Umgebungen möglichst realistisch zu gestalten, indem das Ambiente als auch das Verhalten der jeweiligen Bezugsgruppe in der jeweiligen Umgebung authentisch nachgeahmt wurden.

1.1 Motivation

Mit der stetigen Entwicklung der virtuellen Welten und insbesondere der Virtual Reality (VR) eröffnen sich neue Möglichkeiten, um Umgebungen äußerst realistisch zu gestalten. Diese Fortschritte führen zu einer steigenden Nutzung von VR und erhöhen zugleich die Erwartungen der Nutzer. In dieser Arbeit wird eine Anwendung entwickelt, die es dem Benutzer ermöglicht, mit einer VR-Brille in eine virtuelle Welt einzutauchen und dabei das Gefühl zu haben, sich wirklich in der jeweiligen Situation zu befinden. Diese Anwendung simuliert das Halten eines Vortrags, um Nutzern zu helfen, ihre Ängste zu überwinden oder ihre Fähigkeiten zu verbessern.

Der Bedarf an solchen Anwendungen wächst, da viele Menschen unter Rede- oder Präsentationsangst leiden. Diese Personen fühlen sich oft unwohl und werden schnell nervös, wenn sie vor Anderen sprechen müssen. Die entwickelte Anwendung richtet sich speziell an die Studierenden der Philipps-Universität Marburg und wurde entsprechend in realistischen, relevanten Umgebungen gestaltet.

Durch den Einsatz von Virtual Reality wird eine immersive und realitätsnahe Erfahrung ermöglicht, die diese Anwendung besonders wirksam macht. Zusätzlich sorgt der Einsatz von MetaHumans aus der Unreal Engine für einen hohen Grad an Realismus, was das Projekt einzigartig macht. Dank der Integration von Live-Link wurden die Animationen der MetaHumans besonders detailgetreu und lebensnah umgesetzt, was die Authentizität der simulierten Umgebung weiter steigert.

Zukunftspotenzial

Dank der Entwicklungen des MetaHuman-Plugins und der Virtual Reality (VR) bietet diese Arbeit in der Zukunft großes Ausbau- und Entwicklungspotenzial. So könnten beispielsweise die Auflösungen von VR-Brillen weiter verbessert oder hochwertigere Lautsprecher integriert werden, was die Wahrnehmung der virtuellen Welt erheblich steigern würde.

Darüber hinaus wird die Unreal Engine zusammen mit dem MetaHuman-Plugin kontinuierlich weiterentwickelt. Diese Fortschritte könnten in Zukunft noch realistischere MetaHuman-Figuren ermöglichen und damit die Authentizität der simulierten Umgebungen weiter verbessern.

1.2 Virtual Reality

Als virtuelle Realität kurz VR, wird eine Darstellung und Wahrnehmung einer scheinbaren Wirklichkeit mit ihren physischen Eigenschaften in einer Echtzeit computergenerierten, interaktiven virtuellen Umgebung bezeichnet. Die visuellen Strategien heutiger Illusionstechniken lassen sich bis zur Antike zurückverfolgen, ein bekanntes Beispiel wären die 360-Grad-Rundbilder in der Villa dei Misteri [1].

Die moderne Ära der VR begann 2012 mit der Entwicklung durch das Startup-Unternehmen Oculus VR, das den Weg für die heutigen VR-Headsets ebnete.

Das Gefühl der Immersion wird durch das VR-Headset erzeugt, welches zwei Bilder aus unterschiedlichen Perspektiven erzeugt und darstellt, um ein stereoskopisches 3D-Erlebnis zu bieten. Zudem steuert man die virtuelle Welt

mit den Eingabegeräten, welche die menschliche Hand nachahmen. Diese Technologien verstärken das Gefühl der Immersion weiter.

Heute ist der Anwendungsbereich der Virtual Reality sehr umfangreich und wächst kontinuierlich. VR spielt eine bedeutende Rolle in Bereichen wie Flugsimulationen für Piloten, Videospielen und sogar in der Immobilienbranche als visuelle Darstellung.

1.3 Problemstellung und Zielsetzung

Wie schon in Kapitel 1 erwähnt, stellte die realitätsnahe Gestaltung der Umgebung eine besondere Herausforderung dar. Dazu gehören nicht nur das Ambiente und das Verhalten der Bezugsgruppen, sondern auch viele weitere detailreiche Aspekte, wie die Animation der Körperbewegungen und der Mimik der MetaHumans. Darüber hinaus war auch die allgemeine Gestaltung der virtuellen Umgebung, einschließlich der Lichteinstellungen und der Szenengestaltung, von großer Bedeutung.

Um sicherzustellen, dass das Projekt mit optimalen Frames per Second (FPS) läuft und nicht unter FPS-drops leidet, mussten Auflösungen, Texturen und MetaHumans optimiert werden. Diese Optimierungsprozesse waren unerlässlich, um das Ziel einer möglichst realistischen Darstellung zu erreichen. Gleichzeitig stellten sie jedoch eine erhebliche technische Herausforderung dar, da sie eine Balance zwischen visueller Qualität und flüssiger Performance erforderten.

1.4 Aufbau der Arbeit

Im ersten Abschnitt des zweiten Kapitels wird eine Einführung in die Unreal Engine gegeben, um dem Leser die Entwicklungsumgebung näherzubringen und einen besseren Einblick in den Entwicklungsprozess zu ermöglichen. Außerdem wird das MetaHuman-Plugin erläutert, einschließlich seiner Funktion und des Verwendungszwecks. Es wird auch einen Überblick über die Werkzeuge gegeben, die für die Animation der MetaHumans verwendet wurden. Die darauffolgenden Kapitel, sollen den Ablauf des Entwicklungsprozesses widerspiegeln.

Im dritten Kapitel wird die Erstellung der Level mithilfe von Assets und Materialien sowie die Integration der MetaHumans beschrieben. Dazu gehören auch die Erstellung von Lichtquellen und die Nutzung verschiedener Funktionalitäten der Unreal Engine. Ergänzend werden die Verfahren erläutert, die zur Optimierung der Umgebungen angewendet wurden.

Im vierten Kapitel werden die im zweiten Kapitel erwähnten Werkzeuge zur Animation der MetaHumans vertieft behandelt und genauer dargestellt. Der gesamte Ablauf des Animationsprozesses wird dabei erläutert und visualisiert.

Kapitel fünf behandelt die Integration von Audiodaten und deren Einfluss auf die Immersion. Es wird beschrieben, wie 3D- und Ambient-Audios gezielt eingesetzt wurden, um eine realistische und immersive Umgebung zu schaffen, und wie diese Audiodaten technisch in das Projekt integriert wurden.

Im sechsten Kapitel werden die MetaHumans detaillierter betrachtet, insbesondere hinsichtlich ihrer Individualisierungsmöglichkeiten, wie den LOD-Einstellungen und den verschiedenen Auflösungsstufen.

Im siebten Kapitel liegt der Fokus auf der Integration der Virtual Reality in die Arbeit. Es werden die Implementierung der VR-Funktionalität in die Unreal Engine sowie die damit verbundene Performance und Optimierung beschrieben.

Kapitel acht beschreibt die Entwicklung der Benutzeroberfläche, einschließlich des Level-Select-Menus, des MetaHuman-Select-Menus und des Störanimations-Konfigurations-Menus, mit Fokus auf eine benutzerfreundliche und intuitive Bedienung.

Im neunten Kapitel wird die Implementierung der Anwendung erläutert, dabei werden die Strukturen in den Blueprint-Klassen betrachtet. Im darauffolgenden Kapitel werden die Ergebnisse der Arbeit, sowie der Zukunfts-Ausblick bewertet.

2. Grundlagen

In diesem Kapitel werden die grundlegenden Konzepte und Technologien erläutert, die für das Verständnis der folgenden Abschnitte von Relevanz sein werden. Es bietet einen Überblick über die Funktionalitäten der Unreal Engine, beschreibt deren Aufbau und Nutzung, und stellt das MetaHuman-Plugin vor, welches in dieser Arbeit eine zentrale Rolle spielt [2] [3].

2.1 Einführung in die Unreal Engine

Die Unreal Engine ist eine Grafik-Engine, die seit 1995 entwickelt wird. Das erste Spiel, welches mit der Unreal Engine entwickelt wurde, wurde im Jahr 1998 vorgestellt. Ursprünglich wurde die Unreal Engine für die Unreal-Spielserie entwickelt, später fanden auch weltbekannte Spiele wie Fortnite und Rocket League ihren Ursprung in dieser Engine.

Dank ihrer grafischen Leistungsfähigkeit, die Technologien wie Ray Tracing, Lumen für Echtzeit-GI (Global Illumination) umfasst, ihrer breiten Anwendbarkeit wie in Bereichen der Filmproduktion, ihrer Benutzerfreundlichkeit durch das Blueprint-System und der großen Auswahl an Ressourcen im Marketplace, hat sich die Unreal Engine, als eine der angesehensten und bewährtesten Engines etabliert. Ein weiterer entscheidender Faktor ist, dass die Unreal Engine als Open Source verfügbar ist [4]. In dieser Arbeit wird die Unreal Engine 5 benutzt, die seit 2020 zur Verfügung steht und auch das MetaHuman-Plugin beinhaltet.

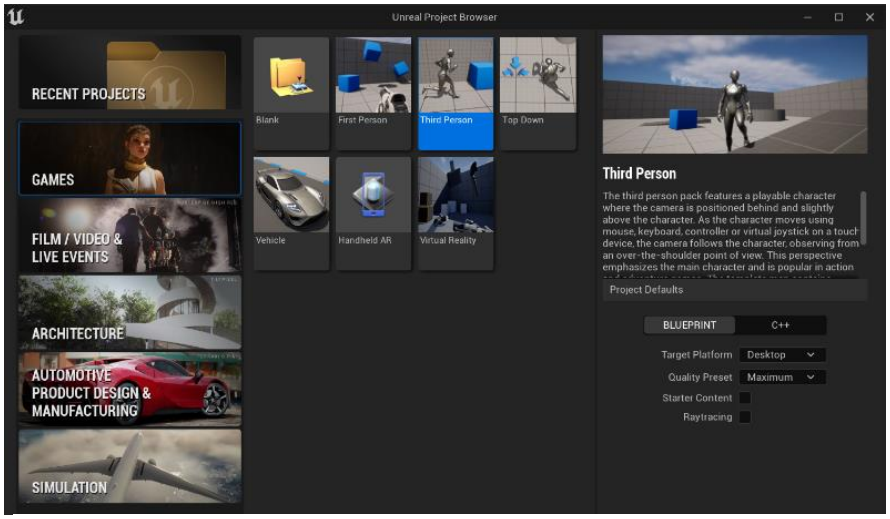

Abbildung 1: Auswahl des Templates und Projekt-Typs bei der Projekterstellung in Unreal Engine 5.

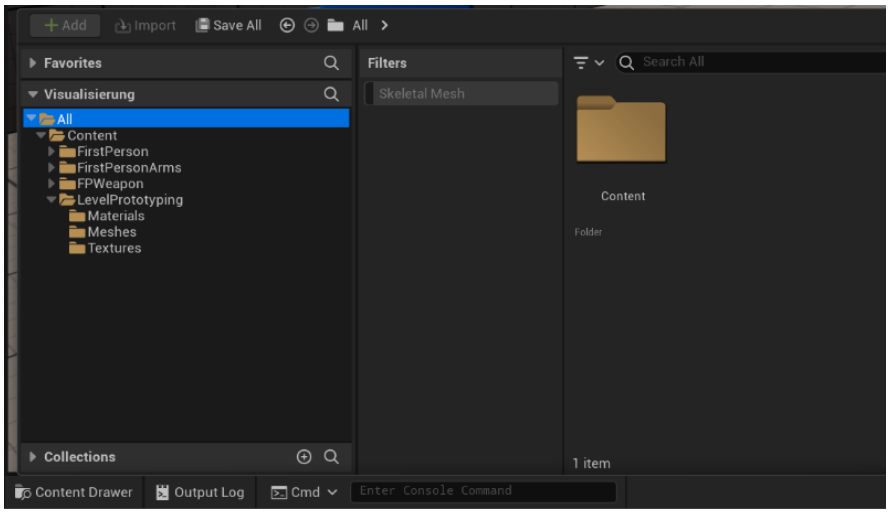

Abbildung 2: Darstellung des Content Drawers und der Ordnerstruktur in Unreal Engine 5.

In Unreal Engine hat man bereits bei der Erstellung eines Projektes eine große Auswahl und Diversifikation, je nachdem, ob man eine Simulation, ein Spiel oder Cinematics entwickeln möchte. Zusätzlich bietet Unreal Engine die Möglichkeit, ein vorgefertigtes Template auszuwählen, in dem bestimmte primitive Funktionen und Steuerungen bereits definiert sind Diese Templates bieten eine solide Grundlage für die weitere Entwicklung und ermöglichen es, schneller in den kreativen Entwicklungsprozess einzusteigen (siehe Abbildung 1).

Der Content-Drawer in Unreal Engine ist in einer Baumstruktur aufgebaut, die es ermöglicht Inhalte wie Meshes, Animationen, Materialien und Figuren in verschiedenen Ordnern zu organisieren. Eine gut strukturierte Ordnerhierarchie fördert die Wartbarkeit und Lesbarkeit des Projektes und sollte daher mit größter Sorgfalt gehandhabt werden. Durch eine klare Organisation lassen sich Inhalte schneller finden und Projekte effizienter verwalten (siehe Abbildung 2).

Meshes sind 3D-Modelle, die in Grafik-Engines als grundlegende Bausteine, die für die Gestaltung von Levels und Szenen dienen. Sie bestehen aus einer Vielzahl von Polygonen, die zusammen eine Form eines Objektes bilden und können aus externen 3D-Modellierungsprogrammen wie Blender oder Maja importiert werden. In Unreal Engine werden Meshes verwendet, um physische Objekte in der virtuellen Welt darzustellen, von einfachen Strukturen wie Wänden und Stühlen bis hin zu komplexen Mesh-Strukturen.

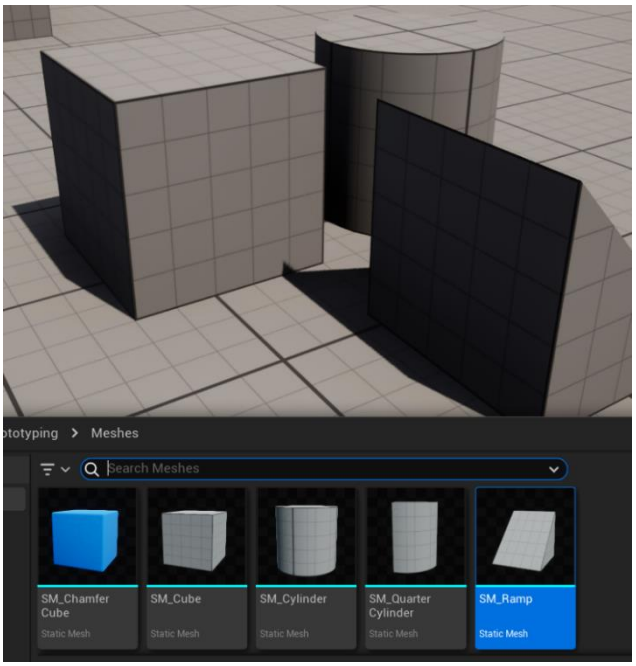

Abbildung 3: Übersicht der Meshes im Content Drawer von Unreal Engine 5.

Wie in Abbildung 3 dargestellt, können Meshes in Unreal Engine ausgewählt und in das Level gezogen werden, wodurch sie dort platziert werden. Die Meshes können beliebig positioniert werden. Um eine Textur auf Meshes anzuwenden, werden Materialien verwendet, die die Texturen auf den Meshes darstellen und somit deren visuelle Darstellung beeinflussen. Zusammenfassend spielen Meshes eine entscheidende Rolle in der Levelgestaltung, da sie die visuelle und physische Grundlage für die Spielwelt bilden.

In Unreal Engine können Objekte nicht nur verschoben, sondern auch skaliert und rotiert werden. Das Verschiebungswerkzeug ermöglicht es, Objekte entlang der X-, Y- und Z-Achsen zu verschieben. Dies ist nützlich, um die Position eines Objekts im Raum präzise anzupassen.

Neben dem Verschiebungswerkzeug (siehe Abbildung 4) gibt es das Rotationswerkzeug, mit dem Objekte um die drei Achsen gedreht werden können, um ihre Ausrichtung zu ändern. Dies ist besonders wichtig, wenn man beispielsweise die Blickrichtung einer Kamera oder die Neigung eines Gebäudes anpassen möchte. Schließlich ermöglicht das Skalierungswerkzeug, Objekte in ihrer Größe zu verändern, indem man sie entlang der X-, Y- oder Z-Achse vergrößert oder verkleinert.

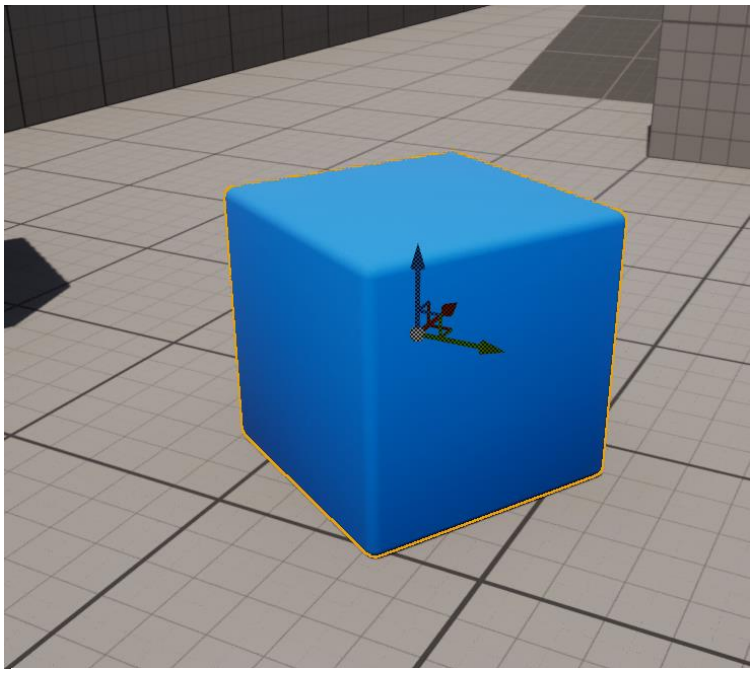

Abbildung 4: Verschiebung eines Meshes im Level Über die Transformationswerkzeuge kann die Position eines Objekts entlang der X-, Y- und Z-Achse angepasst werden.

Die drei erwähnten Werkzeuge werden im Kapitel 3 eine bedeutende Rolle spielen.

Das Detailpanel (siehe Abbildung 5) ermöglicht es, objektspezifische Einstellungen vorzunehmen. Hier können verschiedene Parameter eines ausgewählten Meshes oder Objektes angepasst werden, wie beispielsweise die Materialien, die dem Mesh zugewiesen sind, die Render-Einstellungen, die die Sichtbarkeit und Darstellung beeinflussen, sowie physikalische Eigenschaften und Kollisionsparameter. Dieses Panel ermöglicht eine detaillierte Kontrolle über die einzelnen Aspekte eines Objektes, was für die präzise Gestaltung und Optimierung der Spielwelt unerlässlich ist.

Der Unreal Engine Marketplace bietet eine umfangreiche Auswahl an digitalen Assets, darunter 3D-Modelle, Animationen, Texturen und Sounds [5]. Diese Ressourcen ermöglichen es Entwicklern, Projekte effizient zu gestalten und auf hochwertige, vorgefertigte Inhalte zurückzugreifen. Die Assets werden häufig von professionellen Designern sowie der Unreal Engine-Community erstellt und über den Marketplace vertrieben. Neben kostenpflichtigen Angeboten stehen auch zahlreiche kostenlose Assets zur Verfügung. Der Marketplace ist daher ein essenzielles Werkzeug zur schnellen und professionellen Erstellung komplexer virtueller Umgebungen.

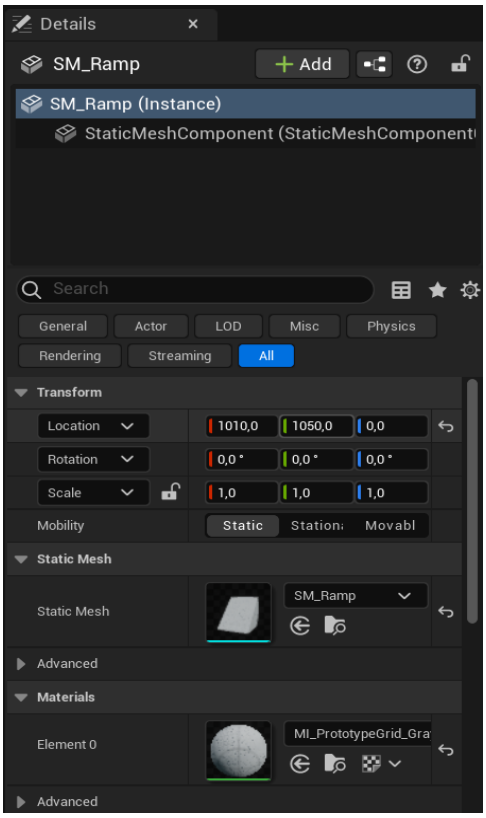

Abbildung 5: Das Detailpanel in Unreal Engine ist ein zentrales Interface-Element, das es Entwicklern ermöglicht, die Eigenschaften und Parameter von ausgewählten Objekten im Editor zu bearbeiten. Sobald ein Objekt im Level ausgewählt wird, zeigt das Detailpanel eine Liste aller verfügbaren Einstellungen und Attribute des Objekts an. Dazu gehören unter anderem Transformationswerte (Position, Rotation, Skalierung), Materialeigenschaften, Beleuchtungsoptionen, physikalische Parameter und spezifische Eigenschaften je nach Objekttyp, wie z.B. Animationseinstellungen für MetaHumans.

2.2 MetaHumans: Technologie und Integration

Abbildung 6: Portrait eines MetaHumans. Das Bild zeigt die Detailgenauigkeit und den Realismus der Charaktere, die durch fortschrittliche Technologien wie Deep Learning und Motion Capturing erreicht werden.

MetaHumans wurden von Epic Games im Jahr 2021 veröffentlicht und sind seitdem für die Unreal Engine 5 verfügbar. Diese Technologie zielt darauf ab, Entwicklern interne, hochdetaillierte und fotorealistische 3D-Charaktere zur Verfügung zu stellen (siehe Abbildung 6).

Die MetaHumans zeichnen sich durch detaillierte Hauttexturen, realistische Mimik und hochwertige Haare aus. Sie nutzen fortschrittliche Shader und Texturen, um ein hohes Maß an Realismus zu erreichen [2].

Die MetaHumans sind für Echtzeitanwendungen optimiert und bieten ein ausgewogenes Verhältnis zwischen Detailtreue und Performance. Dies ermöglicht ihren Einsatz in verschiedenen Bereichen, von Videospielen bis hin zu Simulationen, ohne signifikante Leistungseinbußen. Die Realitätsnähe wird durch eine Kombination aus Deep Learning, Gesichtsscans und Motion Capturing erreicht, was zu realistischen Bewegungen und Ausdrücken führt. Aufgrund ihrer umfassenden grafischen Merkmale finden MetaHumans auch Anwendung in der Filmindustrie. Ein weiterer wesentlicher Vorteil ist der kostenlose Zugang zu dieser Technologie, der die Attraktivität und Verbreitung der MetaHumans weiter erhöht.

2.2.1 Nutzung von Live-Link

Live-Link ist eine Echtzeit-Datenübertragungs- und Verknüpfungstechnologie, die von Epic Games entwickelt wurde. Diese Technologie wurde erstmals mit Unreal-Engine 4 eingeführt und dient der effizienten Integration von Motion-Capture-Daten in die Unreal-Engine. Live-Link ermöglicht die direkte Übertragung von Animationsdaten in Echtzeit zwischen verschiedenen Anwendungen und der Unreal Engine.

Das Live-Link-Plugin ermöglicht es Entwicklern, Bewegungen oder Gesichtsausdrücke mittels Kameras im realen Leben aufzuzeichnen und diese Aufnahmen in Echtzeit auf MetaHumans in einem Unreal-Engine-Projekt zu übertragen. Durch diese Technologie können Entwickler den Entwicklungsprozess beschleunigen und die Qualität der Animationen verbessern, da die Aufnahmen auf realen Menschen basieren. Das Hauptziel von Live-Link ist es, eine nahtlose Übertragung und Synchronisation von Bewegungsdaten zu gewährleisten, sodass Entwickler und Animatoren Echtzeit-Feedback erhalten und umgehend Anpassungen vornehmen können. Für MetaHumans bietet Live-Link die Möglichkeit, direkt mit diesen virtuellen Charakteren während der Entwicklung zu interagieren. Dies führt zu sofortigen visuellen Rückmeldungen und erleichtert die Anpassung der Charakteranimationen [6].

Die Technologie unterstützt eine Vielzahl von Datenquellen, darunter Motion-Capture-Systeme wie Vicon oder OptiTrack. Diese Vielfalt an unterstützten Systemen erhöht die Flexibilität und Integration von Live-Link in unterschiedliche Produktionspipelines.

Im Kapitel 4 wird die Nutzung des Live-Link-Plugins für die Animation der MetaHumans detaillierter erläutert und visuell dargestellt.

2.2.1 Nutzung des Control-Rigs

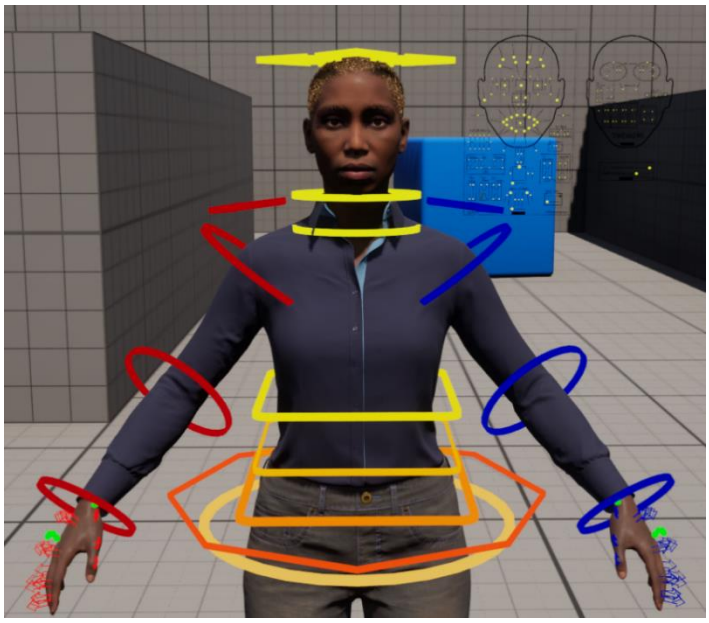

Abbildung 7: MetaHuman Control Rig im Unreal Engine Editor: Die Abbildung zeigt das Control Rig für einen MetaHuman, welches für die präzise Steuerung und Animation des Charakters verwendet wird.

Der MetaHuman-Control-Rig ist ein leistungsstarkes Tool innerhalb der Unreal Engine, das speziell für die Animation von MetaHumans entwickelt wurde. Im Gegensatz zum Standard-Control-Rig ermöglicht der MetaHuman-Control-Rig auch die detaillierte Animation des Gesichts der MetaHumans. Er besteht aus verschiedenen Steuerungselementen und Hierarchien, die die menschliche Anatomie präzise abbilden [7]. Beispielsweise sind die Körperanimationen von den Gesichtsanimationen getrennt, und es ist möglich, sowohl die gesamte Hand als auch die einzelnen Finger zu bewegen. Die Steuerungselemente sowie die Hierarchie sind in Abbildung 7 dargestellt.

Mit dem MetaHuman-Control-Rig können Animatoren Änderungen in Echtzeit überwachen und testen, was den Animationsprozess beschleunigt und unmittelbares Feedback ermöglicht. In dieser Arbeit wurde der MetaHuman-Control-Rig in Kombination mit dem Live-Link-Plugin von Epic Games verwendet, um eine besonders realistische Animation der MetaHuman-Körper und der Mimik der MetaHumans zu erreichen. Diese Verknüpfung hatte einen bedeutenden Einfluss auf die realistische Umsetzung der Arbeit, und hat unmittelbar den Prozess beschleunigt.

3. Projektstruktur und Level-Erstellung

In diesem Kapitel werden die Prozesse der Levelerstellung sowie die Strukturierung des Projekts detailliert erläutert. Der Aufbau des Kapitels folgt dabei dem Ablauf, der auch bei der praktischen Umsetzung der Level verwendet wurde. Zunächst wird auf die Auswahl und Anwendung der Assets eingegangen, gefolgt von einer Betrachtung der Materialien, die zur Gestaltung der Oberflächen verwendet wurden. Im Anschluss wird das Licht- und Schattenmanagement und dessen Optimierung diskutiert. Schließlich wird die Integration der MetaHumans in das Projekt beschrieben, wobei auf deren Platzierung innerhalb der Level eingegangen wird.

3.1 Erstellung der Levels in Unreal Engine

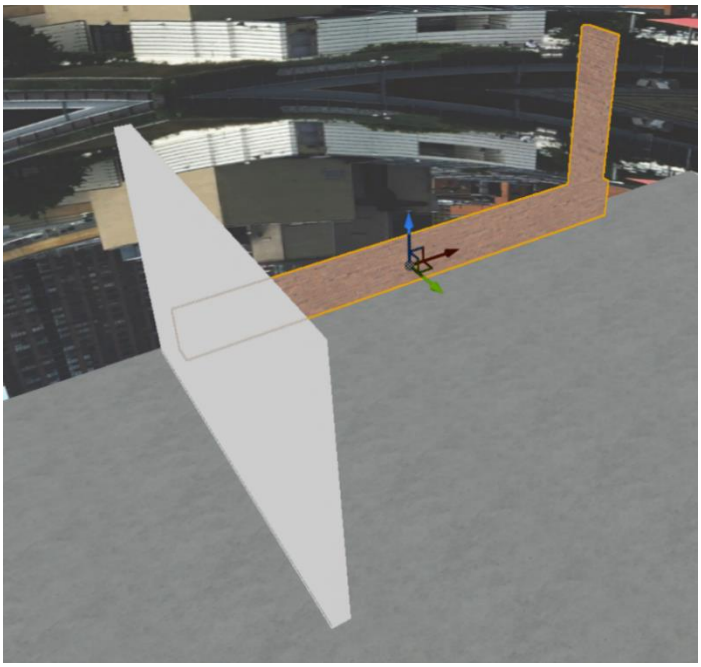

Abbildung 8:Visualisierung der Levelerstellung in Unreal Engine, mittels Assets und den Transformationswerkzeugen.

Im Rahmen dieser Arbeit wurden drei unterschiedliche Level entwickelt, die auf die Bedürfnisse der Zielgruppe der Nutzer abgestimmt sind: ein Hörsaal, ein Meetingraum und ein Klassenzimmer. Diese Szenarien wurden ausgewählt, um verschiedene Nutzungsszenarien der Anwendung abzudecken, wie z. B. Vorlesungen, Besprechungen und Unterrichtsstunden.

Die Gestaltung der Levels erfolgte eigenständig, wobei die Auswahl und Platzierung der Assets entscheidend für die Schaffung einer immersiven und

funktionalen Umgebung war. Die im Unterkapitel 3.1.1 beschriebenen Assets wurden mithilfe der in Unreal-Engine 5 zur Verfügung gestellten Transformationswerkzeuge sorgfältig platziert, um eine realistische und nutzerfreundliche Anordnung der Level zu erreichen. Die Eigenschaften dieser Werkzeuge wurden im Kapitel 2 bereits sorgfältig erläutert.

Dabei wurde besonders auf die genaue Positionierung und die sinnvolle Skalierung der Objekte geachtet, um eine authentische und stimmige Umgebung zu gestalten, die den funktionalen Anforderungen der jeweiligen Nutzungsszenarien entspricht.

Während des Levelentwicklungsprozesses liegt der anfängliche Fokus auf der grundlegenden Gestaltung der Umgebung. In diesem Stadium werden zunächst die wesentlichen Elemente wie Wände, Böden und andere strukturelle Komponenten erstellt (siehe Abbildung 8). Daraufhin folgt die detailliertere Ausarbeitung des Innenbereichs, bei der Objekte wie Tische und Stühle hinzugefügt werden (siehe Abbildung 9). Im abschließenden Schritt wird die Feinausstattung des Innenbereichs vorgenommen, einschließlich der Platzierung von kleineren Objekten wie Laptops auf den Tischen, Vasen oder Lampen an den Decken. Dieser iterative Ansatz gewährleistet, dass die grundlegende Struktur des Levels steht, bevor die detailliertere Gestaltung und Dekoration erfolgt.

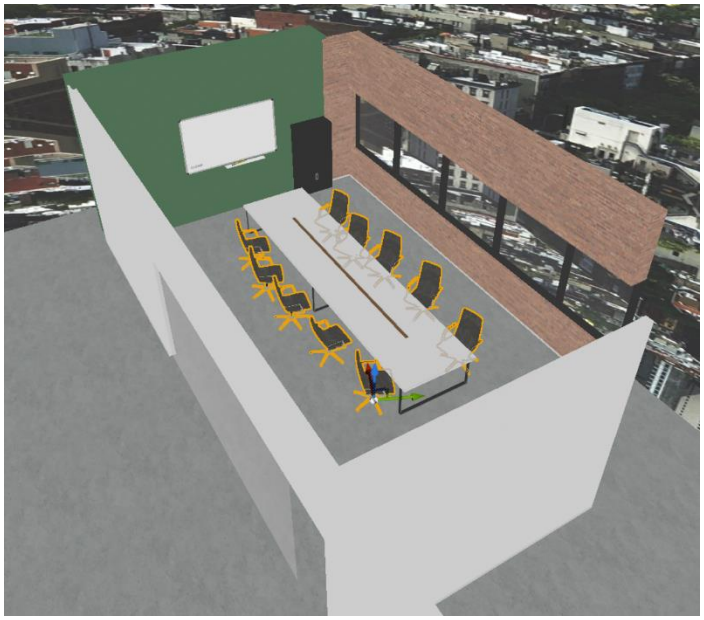

Abbildung 9: Darstellung des Meeting-Raums in der Entwicklungsphase unter Verwendung von Unreal-Engine 5.

Nach der grundlegenden Levelerstellung folgen feinere Schritte wie das Licht- und Schattenmanagement sowie deren Optimierung, die in den Unterkapiteln 3.1.2 und 3.1.3 detailliert erläutert werden.

3.1.1 Verwendung von Assets und Materialien

Die für die Levelgestaltung verwendeten Assets wurden sorgfältig aus dem Unreal-Engine-Marketplace ausgewählt, um die spezifischen Anforderungen der drei Szenarien Hörsaal, Klassenraum und Meetingraum zu erfüllen. Die Verwendung vorgefertigter Assets verkürzt nicht nur die Entwicklungszeit erheblich, sondern gewährleistet auch eine hohe Qualität, da viele dieser Assets von professionellen Designern erstellt wurden [8] [9].

Neben der Auswahl an Meshes spielte die Zuweisung verschiedener Materialien eine entscheidende Rolle bei der Erzeugung der gewünschten Umgebungseffekte. Durch die Anwendung spezifischer Materialien auf die Meshes konnte das visuelle Erscheinungsbild flexibel angepasst werden. Beispielsweise kann eine Wand durch die Zuweisung eines weißen Materials eine glatte und neutrale Oberfläche erhalten. Wird jedoch ein Material mit einer Ziegelmuster-Textur verwendet, verwandelt sich die gleiche Wand in eine realistisch anmutende Ziegelmauer. Diese Möglichkeit der Individualisierung der Meshes ermöglichte es, die gestalterischen Anforderungen der einzelnen Level präzise zu erfüllen und die Vielfalt der Umgebungen innerhalb des Projekts zu erweitern.

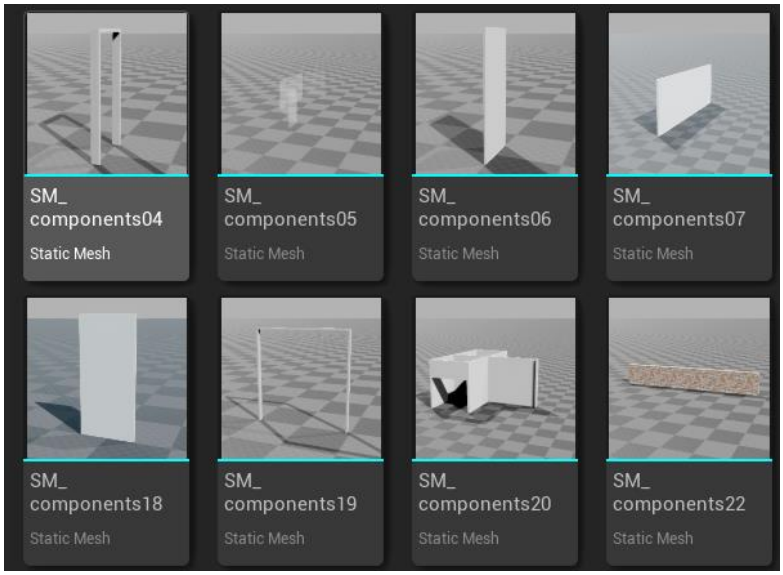

Abbildung 10: Auswahl der im Projekt verwendeten Meshes zur Gestaltung der Levels (Hörsaal, Meetingraum, Klassenzimmer).

Die ausgewählten Meshes wurden aus dem Mesh-Ordner (siehe Abbildung 10) des Projektes durch einfaches Ziehen und Ablegen in der virtuellen Umgebung positioniert und anschließend mit den in Kapitel 3.1 beschriebenen Transformationswerkzeugen weiterbearbeitet.

3.1.2 Lichtmanagment

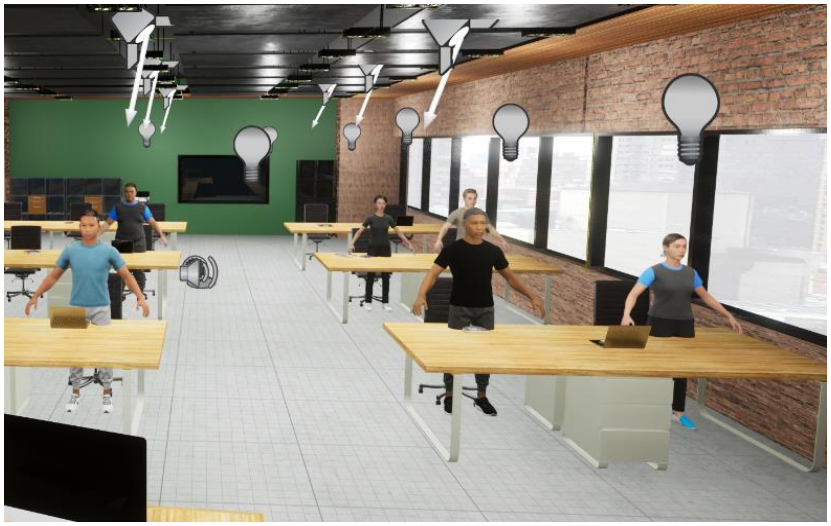

Abbildung 11: Integration der Lichter in das Klassenzimmer, um eine realistische Wahrnehmung zu generieren.

Um einen hohen Realitätsgrad in der Anwendung zu erreichen, war ein besonderes Augenmerk auf das Licht- und Schattenmanagement erforderlich, da diese Elemente eine entscheidende Rolle in der Wahrnehmung durch die Nutzer spielen. Die Beleuchtung in den verschiedenen Levels musste so gestaltet werden, dass sie die Lichtverhältnisse der realen Welt möglichst genau nachbildet.

In der Unreal Engine stehen verschiedene Lichtquellen zur Verfügung, um Innenräume passend zu beleuchten, darunter Spot-Light, Point-Light, Directional-Light und Rect-Light, die jeweils spezifische Eigenschaften aufweisen. Das Spot-Light erzeugt einen kegelförmigen Lichtstrahl, ideal zur gezielten Beleuchtung bestimmter Bereiche. Das Point-Light emittiert Licht gleichmäßig in alle Richtungen von einem zentralen Punkt aus, ähnlich einer Glühbirne, und eignet sich gut für die allgemeine Ausleuchtung von Räumen. Das Directional-Light simuliert Sonnenlicht mit parallel verlaufenden Lichtstrahlen, die für großflächige Beleuchtungseffekte verwendet werden. Das Rect-Light bietet eine rechteckige Lichtquelle, die für eine gleichmäßige

Beleuchtung breiter Flächen sorgt. In dieser Arbeit wurden primär Point-Lights, Spot-Lights sowie Directional-Lights eingesetzt, um die gewünschten Lichteffekte zu erzielen [10].

Die Integration und Anpassung der Beleuchtung erfolgte individuell für jedes Level, um die spezifische Anforderung gerecht zu werden. So wurden beispielsweise im Hörsaal andere Lichtverhältnisse geschaffen als im Meetingraum, um den unterschiedlichen Raumgrößen und den Anforderungen gerecht zu werden.

Wie in Abbildung 11 zu sehen ist, wurde im Klassenzimmer eine Kombination aus Point-Lights und Directional-Lights verwendet. Zudem wurde mehr Beleuchtung in der Nähe der Fenster positioniert, um das einfallende Tageslicht zu simulieren. Diese Vorgehensweise wurde auch im Meetingraum angewendet, da die räumlichen Gegebenheiten ähnlich sind.

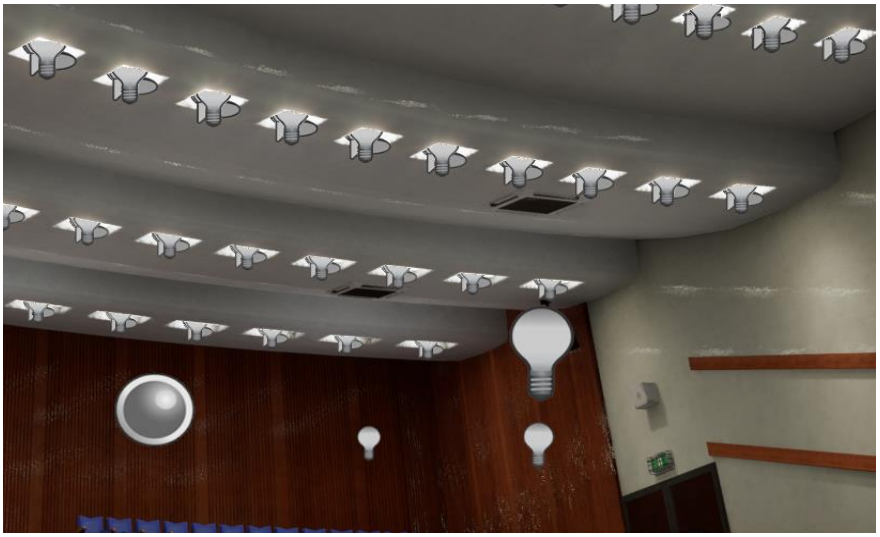

Abbildung 12: Integration der Lichter in den Hörsaal, um eine realistische Wahrnehmung zu generieren.

Im Hörsaal wurde ebenfalls eine Kombination aus Point-Lights und Directional-Lights eingesetzt (siehe Abbildung 12). Zusätzlich kam eine Sphere-Reflection-Capture-Komponente zum Einsatz, die Umgebungsreflexionen erfasst und realistische Spiegelungen auf Oberflächen innerhalb der Szene erzeugt. Diese Komponente trägt dazu bei, dass reflektierende Materialien wie Glas oder Metall natürlicher wirken und die visuelle Qualität der gesamten Szene erhöht wird.

3.1.3 Licht- und Schattenoptimierung

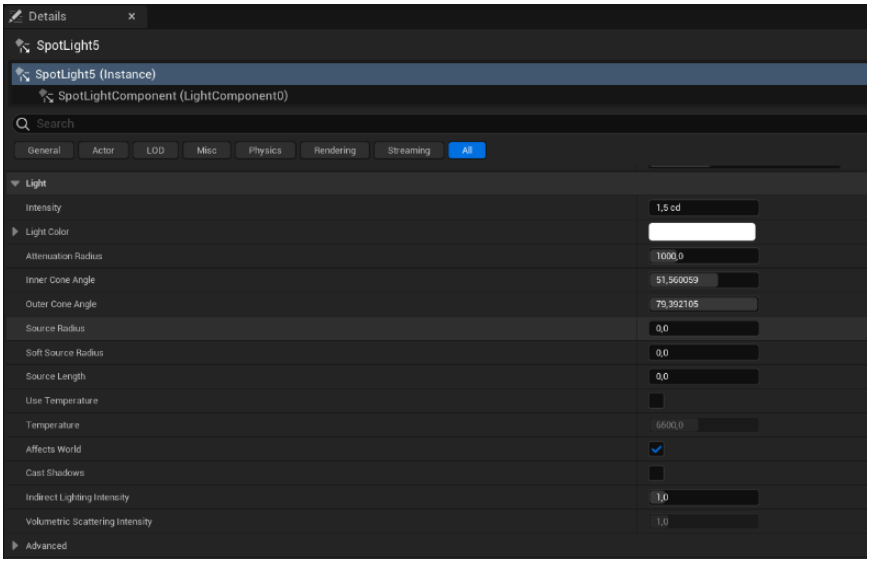

Abbildung 13: Die Einstellungen der Spot-Lights im Level "Klassenzimmer".

Um die Schatten- und Lichtverhältnisse in Unreal Engine zu optimieren, stehen zahlreiche Werkzeuge zur Verfügung. Die globalen Licht-Einstellungen können über die World-Settings angepasst werden, insbesondere unter dem Abschnitt Lightmass. Hierbei lassen sich Parameter wie Static-Lighting-Level-Scale und Num-Indirect-Lighting-Bounces einstellen. Der Parameter Static-Lighting-Level-Scale bestimmt die Detailgenauigkeit der statischen Beleuchtung, wobei ein höherer Wert zu detaillierteren, aber rechenintensiveren Lichtberechnungen führt. Num-Indirect-Lighting-Bounces legt die Anzahl der Reflexionen fest, die von indirektem Licht im Raum berechnet werden, was die Tiefe und Realismus der Beleuchtung beeinflusst. Die spezifischen Lichtkomponenten werden im Details-Panel angepasst, das die Kontrolle über lichtspezifische Parameter wie Schattenwurf und Lichtintensität ermöglicht [11].

Um Überbelichtung zu vermeiden und die Leistung zu optimieren, wurde die Lichtintensität auf 1,5 Candela (cd) eingestellt und die Option „Cast Shadows“ auf „false“ gesetzt (siehe Abbildung 13).

Die Deaktivierung der Schattenprojektion („Cast Shadows“) reduziert die Berechnungslast, da das Licht keine Schatten wirft. Dies trägt zur Verbesserung der Performance bei und ermöglicht ein flüssiges Erlebnis, insbesondere bei der Nutzung mit Virtual-Reality-Headsets, da Lichtberechnungen erhebliche Ressourcen beanspruchen können [12].

Diese Anpassungen wurden bewusst vorgenommen, um die Hardwareanforderungen zu minimieren und sicherzustellen, dass die Zielgruppe nicht auf leistungsstarke Hardware angewiesen ist.

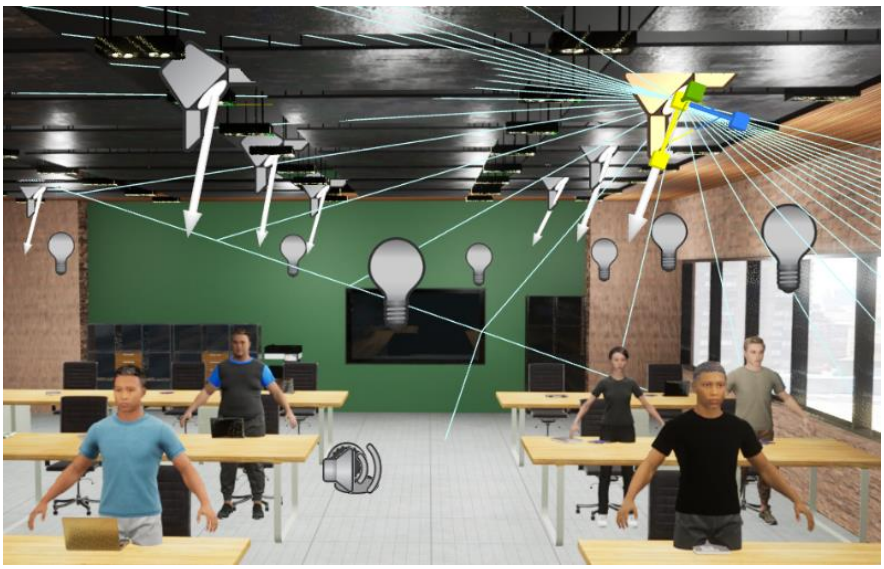

Abbildung 14: Rotation der Spotlights zur besseren Darstellung der Lichtstrahlen, die durch das Fenster ins Klassenzimmer gelangen.

Um die Lichtverhältnisse, die durch das Tageslicht von den Fenstern in das Klassenzimmer gelangen, realistisch darzustellen, wurden die Spotlights gezielt in Richtung des Innenraums ausgerichtet. Diese Rotation der Lichter diente dazu, die natürliche Lichtstreuung besser zu simulieren und so eine authentische Beleuchtung des Raumes zu erreichen. Diese Vorgehensweise wird auch in Abbildung 14 visualisiert.

In dieser Arbeit wurde die Lichtintensität in der Einstellung „Intensity-Units“ auf Candela (cd) festgelegt. Candela, als Maßeinheit für die Lichtstärke, bietet eine präzise Kontrolle über die Intensität der Lichtquelle, während Lumen die gesamte Lichtmenge misst, die von der Lichtquelle abgestrahlt wird. Durch die Wahl von Candela konnte eine differenzierte Lichtgestaltung erzielt werden, da diese Einheit eine gezielte Steuerung der Lichtstärke ermöglicht. Zudem wurde die „Mobility“-Einstellung auf „Static“ gesetzt. Diese Konfiguration bewirkt, dass das Licht statisch bleibt und keine dynamischen Änderungen durchläuft. „Static“ Lichtquellen werden nur einmal berechnet, was die Laufzeit-Performance entlastet, da keine fortlaufenden dynamischen Berechnungen erforderlich sind. Dies reduziert

die Renderkosten und verbessert die Performance, insbesondere bei komplexen Szenen oder Echtzeitanwendungen.

Nach der Integration der Lichter muss das Level erneut gebuildet werden, um die Lichtverhältnisse zu berechnen und die Lichtquellen korrekt in die Szene integriert werden.

3.2 Integration der MetaHumans

Abbildung 15: Integration von MetaHumans in das Klassenzimmer. Die Abbildung zeigt die Platzierung der MetaHumans durch Ziehen und Ablegen der Blueprints in die virtuelle Umgebung.

Die Integration der MetaHumans in die Levels erfolgte durch das einfache Ziehen und Ablegen der entsprechenden Blueprints in der virtuellen Umgebung. Die MetaHumans wurden gezielt an festgelegte Positionen, wie an Stühlen oder Sitzen, platziert (siehe Abbildung 15). Nach der Platzierung der MetaHumans ist es erforderlich, das Level neu zu erstellen (zu rebuilden), um sicherzustellen, dass alle Änderungen korrekt umgesetzt werden.

4. Animationserstellung mit der Unreal Engine

Die Unreal Engine bietet über den Marketplace die Möglichkeit, bereits von Fachleuten erstellte Animationen zu erwerben oder eigene Animationen mithilfe der Animationswerkzeuge der Unreal Engine zu erstellen. In dieser Arbeit wurden sämtliche Animationen eigenständig unter Verwendung der im Kapitel 2.2 beschriebenen Methoden erstellt.

Der Animationsprozess stellte einen erheblichen zeitlichen Aufwand dar. Ein wesentlicher Aspekt war die Realisierung möglichst authentischer Animationen der MetaHumans, wobei die menschliche Anatomie und Verhaltensweisen als Referenz dienten. Beispielsweise wurden subtile Mimiken, wie die Reaktion eines MetaHumans auf ein klingendes Gerät, detailliert umgesetzt. Diese Feinheiten trugen erheblich zur realistischen Atmosphäre bei.

Des Weiteren wurden menschliche Verhaltensweisen wie Ablenkung, Unkonzentriertheit oder Langeweile präzise animiert. Eine besondere Herausforderung bestand darin, Störanimationen zu entwickeln, die gezielt die Atmosphäre der Level beeinflussen, um den Nutzern der VR-Umgebung ein authentisches Erlebnis zu vermitteln.

4.1 Die Animationswerkzeuge

Aufbauend auf den in Kapitel 2.2 behandelten Grundlagen der Animationswerkzeuge wie dem MetaHuman-Control-Rig und dem Live-Link-Plugin, wird in diesem Kapitel die Anwendung dieser Werkzeuge detaillierter erläutert [6] [7] [13].

Wie in Abbildung 16 dargestellt, zeigt das MetaHuman-Control-Rig verschiedene Skelettkomponenten, die die menschliche Anatomie präzise repräsentieren. Jede dieser Komponenten, darunter Knochenstrukturen und Gelenke, kann unabhängig bewegt, rotiert oder skaliert werden. Beim Bewegen einer einzelnen Komponente, wie etwa der Schulter, wirken sich diese Änderungen automatisch auf benachbarte Komponenten wie den Arm und die Hand aus, wodurch eine natürliche und konsistente Bewegungsdynamik gewährleistet wird.

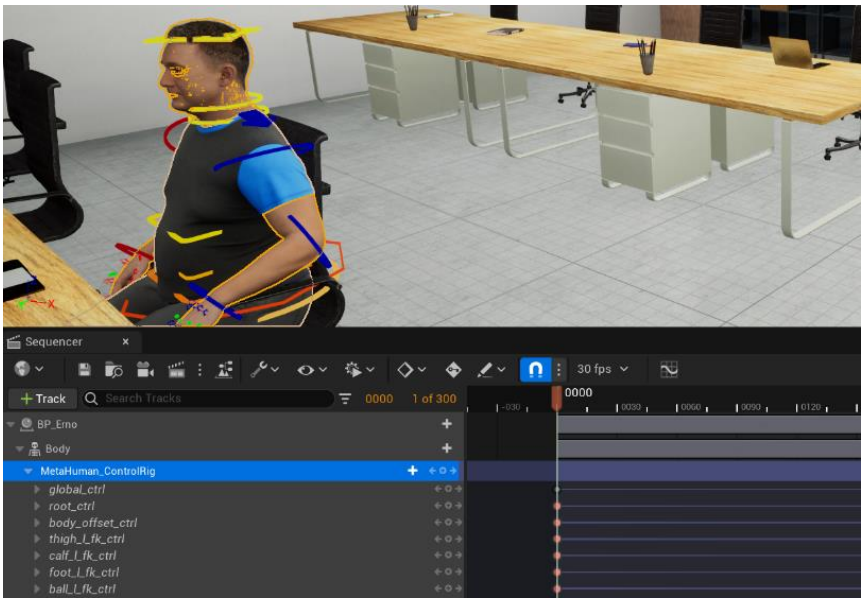

Abbildung 16: Level-Sequence-Player mit MetaHuman-Animation. Das Bild zeigt den Einsatz des MetaHuman-Control-Rigs zur Animation des MetaHumans „Erno“ innerhalb der Level-Sequence.

Das Control-Rig berücksichtigt die anatomischen Verknüpfungen und biomechanischen Beziehungen zwischen den einzelnen Körperteilen, was eine realistische Nachbildung menschlicher Bewegungen ermöglicht. Diese Verknüpfung der Komponenten trägt zur Authentizität der Animation bei, indem sie sicherstellt, dass alle Bewegungen kohärent und physikalisch korrekt sind [7].

In der Abbildung 17 ist ein Level-Sequencer vorzufinden, in dem ein MetaHuman-Control-Rig eingebunden ist. Wie in der Abbildung zu sehen ist, hat jede Skelettkomponente eine eigene Spalte, der Animationspunkte zugewiesen werden können, die zu bestimmten Zeitpunkten im Sequencer erreicht werden sollen. Diese Animationspunkte repräsentieren Keyframes der jeweiligen Skelettkomponente und sind integraler Bestandteil des Animations-Workflows. In einer typischen Animationssequenz werden diese Keyframes genutzt, um flüssige Übergänge zwischen Bewegungen zu erzeugen und die Interaktion der MetaHumans mit der Umgebung präzise zu steuern [14].

Zudem zeigt die Abbildung, dass Skelettkomponenten, die in einer hierarchischen Struktur miteinander verbunden sind, gemeinsam funktionieren, beispielsweise bewegt sich bei einer Schulterbewegung auch der Arm mit. Diese präzise Steuerung ermöglicht es, realistische und

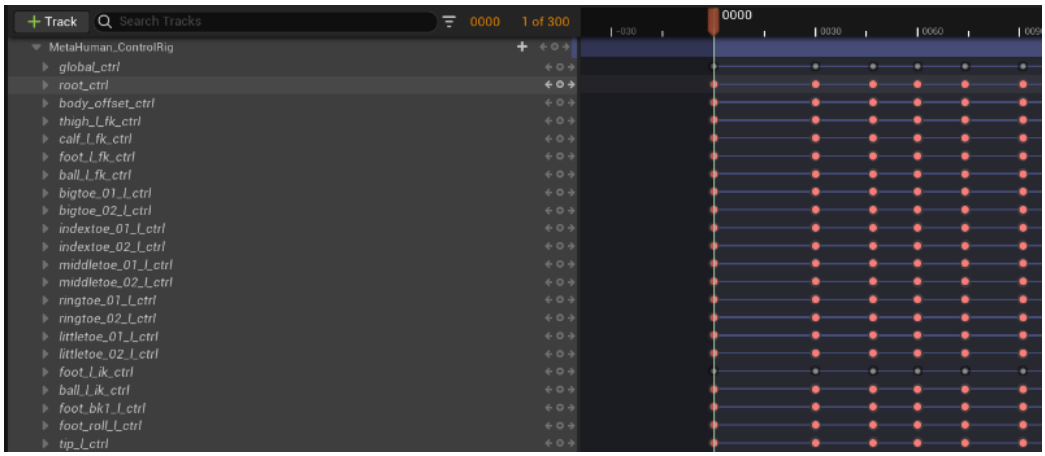

Abbildung 17: Der Level-Sequence-Player zeigt das MetaHuman-Control-Rig, bei dem verschiedene Animationspunkte hervorgehoben sind. Diese Punkte repräsentieren spezifische Steuerungs-Handles innerhalb des Rigs, die für die Animation des MetaHumans verwendet werden. Die Punkte ermöglichen präzise Anpassungen und Bewegungen der einzelnen Körperteile, wie z.B. Arme, Beine oder Gesichtsausdrücke, und tragen zur realistischen Darstellung der Charakteranimation bei.

ausdrucksstarke Bewegungen zu erzeugen, was die Immersion in der VR-Umgebung erheblich steigert.

4.1.1 Verwendung des Level-Sequencers

Im Kapitel 4.1 wurde der Level-Sequencer bereits thematisiert. Auf dieser Grundlage wird nun eine detailliertere Betrachtung des Level-Sequencers vorgenommen. Der Level-Sequencer ist ein Werkzeug innerhalb der Cinematics-Funktionalitäten der Unreal Engine und ermöglicht die Aufnahme und das Abspielen von Szenen. Dabei können verschiedene Komponenten eines Levels, wie beispielsweise Actors oder andere Objekte, individuell getrackt werden. Insbesondere die MetaHumans werden innerhalb der Level-Sequencer getrackt und animiert, wodurch ihre Bewegungen und Interaktionen in der Szene präzise gesteuert werden können. [14]

Der Level-Sequencer bietet vielfältige Anpassungsmöglichkeiten, etwa die Festlegung der Abspielzeit oder die Konfiguration der Frames per Second (fps), um eine optimale Darstellung zu gewährleisten. In dieser Arbeit spielt der Level-Sequencer eine zentrale Rolle, da er nicht nur für die Animationen der MetaHumans verwendet wird, sondern auch ein wesentlicher Bestandteil der zugrunde liegenden Implementierung ist.

4.1.2 Verwendung des Take-Recorders

Der Take Recorder gehört ebenfalls zu den Cinematic-Funktionalitäten der Unreal Engine und wurde in dieser Arbeit verwendet, um die Gesichtsanimationen der MetaHumans präzise aufzunehmen und zu speichern. Ähnlich wie beim Level-Sequencer können auch hier Actors, in diesem Fall die MetaHumans, ausgewählt und für die Aufnahme vorbereitet werden. Durch seine Fähigkeit, komplexe Szenen effizient aufzuzeichnen und zu speichern, trägt der Take Recorder erheblich zur Optimierung der Produktionspipeline bei, insbesondere bei Projekten mit hohem Animationsaufwand [14]. Dies ermöglicht eine schnelle und präzise Erstellung von realistischen Animationen, was den gesamten Workflow wesentlich verbessert.

In dieser Arbeit wird der Take-Recorder mit der Motion-Capture-Technologie „Live-Link“ in Kombination verwendet.

4.1.3 Einsatz von Live-Link

Abbildung 18: Die Live-Link-Face App im Einsatz, zeigt die Erfassung und Analyse der Gesichtsdaten in Echtzeit, einschließlich der Tracking-Punkte wie beispielsweise 'EyeLookDownRight' und 'MouthClosed', um die Mimik für die MetaHuman-Gesichtsanimation in der Unreal Engine zu übertragen.

Wie bereits im Kapitel 4.1.2 erläutert, wird das Live-Link-Plugin in Kombination mit dem Take-Recorder eingesetzt, wobei der Take-Recorder

als Aufnahme-Tool dient und das Live-Link-Plugin die Übertragung von Gesichtsdaten in Echtzeit ermöglicht. Diese Gesichtsdaten werden für den im Take-Recorder ausgewählten MetaHuman in Echtzeit in eine entsprechende Gesichtsanimation umgewandelt.

Um das Live-Link-Plugin in der Unreal Engine zu verwenden, muss es zunächst in den Plugin-Einstellungen der Engine aktiviert werden. Zusätzlich ist ein Aufnahmegerät erforderlich, welches in dieser Arbeit durch die App „Live Link Face“ im Apple Store bereitgestellt wird. Diese App ermöglicht es, ein Smartphone als Aufnahmegerät zu verwenden. Die Verbindung zwischen der Unreal Engine und dem Smartphone wird durch die App und die Engine selbst über die IP-Adresse des Geräts hergestellt [6] [13].

Im Virtual-Production-Fenster der Unreal Engine wird die Verbindung zwischen Smartphone und Engine bestätigt und anschließend im Details-Panel des MetaHuman aktiviert. Sobald diese Verbindung etabliert ist, kann der Take Recorder gestartet werden, um die Gesichtsausdrücke des Darstellers in Echtzeit aufzuzeichnen (siehe Abbildung 18). Diese Aufnahmen werden während der aktiven Aufnahmezeit des Take-Recorders erfasst und in eine animierte Gesichtsdarstellung des MetaHumans umgewandelt und anschließend gespeichert.

Zur Verwaltung der Animationen wurde für jede Animation ein separater Level-Sequencer erstellt, der entsprechend beschriftet wurde, um eine klare Identifikation und Organisation zu gewährleisten. Die in den Level-Sequencer aufgenommenen Animationen können zudem in Form von Animations-Assets weiterverwendet werden. Hierzu wird in der Level-Sequence auf den MetaHuman-Body oder MetaHuman-Face zugegriffen und die Option „Bake Animation Sequence“ ausgewählt. Dadurch werden die entwickelten Animationen als unabhängige Animations-Assets gespeichert, die nicht mehr an den ursprünglichen Level-Sequencer gebunden sind. Diese Vorgehensweise ermöglicht die Wiederverwendung der Animationen bei mehreren MetaHumans innerhalb verschiedener Szenen oder Projekte.

4.2 Verwaltung und Synchronisation

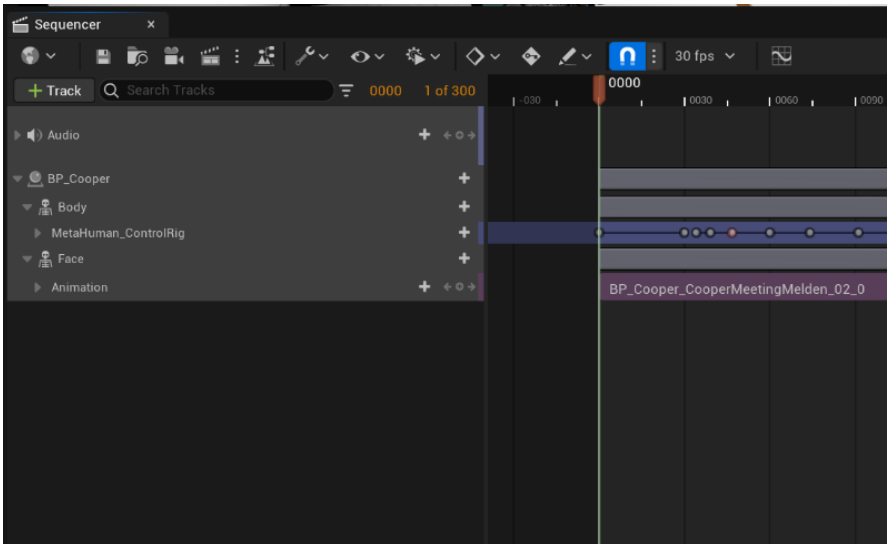

Abbildung 19: Level Sequencer zur Synchronisation von Gesichts- und Körperanimationen.

Um eine präzise Synchronisation der Animationen sicherzustellen, wurde entschieden, die Level-Sequencer weiterhin im Projekt zu verwenden. Dies stellte die optimale Lösung dar, um sicherzustellen, dass die Gesichtsanimationen der MetaHumans synchron zu den Körperanimationen abgespielt werden (siehe Abbildung 19). Um mögliche Konflikte zwischen den verschiedenen Animationen der MetaHumans zu vermeiden, wurde festgelegt, dass alle Animationen eines MetaHumans dieselbe Abspielzeit haben, die in der Regel auf 10 Sekunden eingestellt ist. Diese Konsistenz in der Abspielzeit gewährleistet eine zeitlich präzise Abstimmung aller Bewegungen und fördert somit die Kohärenz und Genauigkeit der Animationen im gesamten Projekt.

Zusammenfassend lässt sich sagen, dass zur effizienten Verwaltung und Strukturierung des Projekts der Ordner „MetaHumanAnimations“ eingerichtet wurde. Innerhalb dieses Hauptordners wurden spezifische Unterordner für die verschiedenen Levels erstellt, wie beispielsweise „AnimationClassroom“ für die Animationen im Klassenzimmer-Level. Diese strukturierte Organisation ermöglicht eine klare Trennung und effiziente Verwaltung der Animationen und Cinematics. Unter dem Hauptordner folgt anschließend die Trennung der Cinematics von den Animation-Assets, welche sicherstellt, dass die unterschiedlichen Elemente des Projekts ordnungsgemäß organisiert sind und erleichtert die Handhabung der Daten.

5. Audiodaten und ihre Integration

In diesem Kapitel werden die verwendeten Audiodaten, deren Optimierungsprozesse sowie die Vorgehensweise bei der Integration dieser Daten in das Projekt detailliert beschrieben. Die nachfolgenden Abschnitte basieren auf den Informationen und Richtlinien, die in der Unreal Engine Dokumentation bereitgestellt werden [15] [16].

5.1 Importieren der Audiodaten

Die im Projekt verwendeten Audiodaten stammen entweder aus eigenen Aufnahmen oder wurden von lizenzierten Sound-Websites heruntergeladen. Dabei wurde sorgfältig darauf geachtet, dass die rechtlichen Vorgaben und Lizenzen eingehalten wurden. Der Import der Audiodateien in die Unreal Engine unterscheidet sich nicht, unabhängig davon, ob die Audiodaten selbst aufgenommen oder aus dem Internet bezogen wurden. Ein wichtiger Aspekt beim Import ist jedoch, dass die Dateien im WAV-Format vorliegen. Das WAV-Format ist ein unkomprimiertes Audioformat, das eine hohe Klangqualität bietet, jedoch auch größere Dateigrößen erzeugt. Die Unreal Engine unterstützt und verarbeitet primär dieses Format, da es eine verlustfreie Audiowiedergabe gewährleistet.

Zur besseren Organisation der Audiodaten wurde im Projektverzeichnis ein Hauptordner mit der Bezeichnung „Sounds“ angelegt. Innerhalb dieses Ordners befinden sich Unterordner, die die Audiodateien nach den verschiedenen Levels des Projekts sortieren. Diese Struktur ermöglicht eine effiziente Verwaltung und erleichtert den schnellen Zugriff auf die benötigten Audiodateien während der Entwicklung.

5.2 Audiodaten in den Animationen

Audiodaten spielten in den Störanimationen eine zentrale Rolle, da sie temporär präzise und realitätsnahe Audiodateien erforderten, die der jeweiligen Umgebung entsprechen. Die Integration der Audiodaten erfolgte in zwei Hauptbereichen: in den Level-Sequencern der Störanimationen und in den Animation Montages.

In den Level-Sequencern wurden die Audiodaten zeitlich exakt an die Gesichtsmimik und Körperbewegungen der MetaHumans angepasst. Besonders wichtig war hierbei die Synchronisation der Mundbewegungen mit der gesprochenen Aussage des MetaHumans. Dies wurde erreicht, indem die im Kapitel 4.1.3 erläuterten Live-Link-Aufnahmen während des Sprechens der passenden Sätze aufgezeichnet wurden. So konnte die Gesichtserfassung direkt mit den gesprochenen Sätzen verknüpft werden, um realistische Mundbewegungen zu erzeugen.

Die zweite Methode der Audiowiedergabe erfolgte über Animation Montages. Diese verbinden die jeweiligen Animations-Assets direkt mit den Audiodateien. Ähnlich wie bei den Level-Sequencern wurden auch hier die Audiodaten sorgfältig positioniert, um eine genaue Synchronisation mit der Mimik und den Körperbewegungen der MetaHumans zu gewährleisten.

5.3 Audiodaten für das Ambiente

Ein Besonderer Wert wurde auf das Ambiente der einzelnen Level gelegt, um durch gezielte Audiodaten die Realitätstreue und Immersion zu steigern. In den Levels wie dem Meetingraum oder dem Hörsaal wurden Geräusche wie das Summen einer Klimaanlage oder beruhigende Hintergrundgeräusche verwendet, um eine authentische Atmosphäre zu erzeugen.

Im Hörsaal wurden zudem verschiedene Soundeffekte implementiert, die mit einer bestimmten Wahrscheinlichkeit abgespielt werden. So gibt es beispielsweise ruhige Hintergrundgeräusche für entspannte Situationen sowie lautere, wie etwa das Murmeln von Gesprächen. Diese akustischen Effekte wurden durch Audio-Cues gesteuert, die zufällig ausgewählt und abgespielt werden.

Zur präzisen Integration von Audiodaten in die Szenen wurden diese in Audio-Cues umgewandelt. Die Audio-Cues ermöglichen eine gezielte Platzierung und Steuerung von Soundeffekten innerhalb der virtuellen Umgebung, indem Parameter wie Hörreichweite und Lautstärke an die spezifischen Anforderungen der jeweiligen Szene angepasst werden. Diese Audio-Cues wurden anschließend durch einfaches Ziehen und Ablegen an den gewünschten Positionen im Level platziert.

5.4 Audiodaten in der Benutzeroberfläche

Auch wurden Sounds in die Benutzeroberfläche integriert, um die Betätigung von Schaltflächen auditiv zu unterstützen. Dies dient dazu, das Nutzungserlebnis zu verbessern, indem akustische Rückmeldungen das Gefühl der Interaktion verstärken und dem Benutzer eine zusätzliche Bestätigung geben, dass eine Schaltfläche erfolgreich betätigt wurde.

Die Zuordnung von Sounds zu den jeweiligen Schaltflächen erfolgt über das Detail-Panel der entsprechenden Widget-Klasse. Unter dem Abschnitt *Appearance* → *Style* kann ein Audioeffekt sowohl für den gedrückten als auch für den fokussierten Zustand der Schaltfläche hinzugefügt werden.

5.5 Optimierung der 3D-Audios

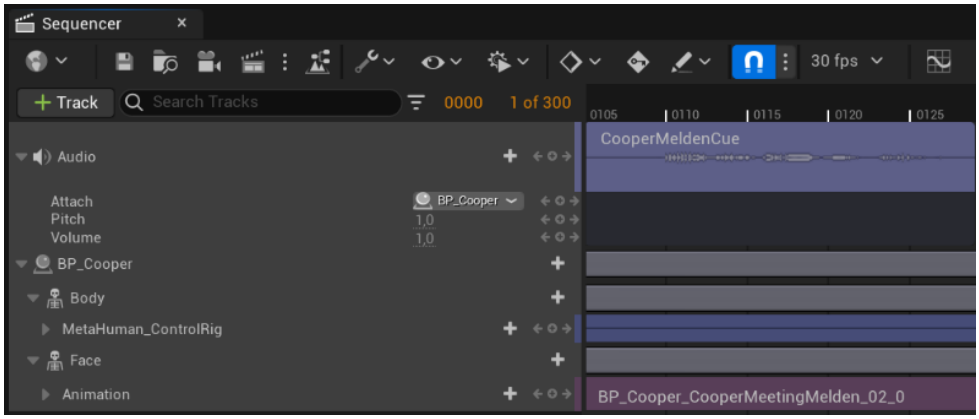

Abbildung 20: Integration der 3D-Audios in die Level-Sequenz für die Störanimation. Das Bild zeigt die vollständige Level-Sequenz, in der die 3D-Audios für die Störanimation vom MetaHuman eingebunden sind. Die Integration umfasst die Platzierung und Anpassung der Audiodaten innerhalb des Sequencers, um eine realistische und immersive Audioerfahrung zu gewährleisten. Dabei legt der Parameter „Attach“ die Koordinate des Audios fest.

Um die Audiodaten realistischer zu gestalten und somit die Immersion zu fördern, wurden sie in 3D-Audios umgewandelt, indem Sound-Attenuation verwendet wurde. Diese Technik ermöglicht es, den Audios spezifische Koordinaten in der virtuellen Umgebung zuzuweisen (siehe Abbildung 20), sodass sie eine räumliche Position erhalten. Über Parameter wie „Falloff Distance“, „Inner Radius“ oder „3D-Stereo Spread“ kann der Klang weiter optimiert werden. Zudem kann die Attenuation-Funktion von linear bis logarithmisch oder invers variiert werden, was den Lautstärkeabfall in Abhängigkeit von der Entfernung beeinflusst. In dieser Arbeit wurde die Attenuation-Funktion auf 'linear' eingestellt. Die 3D-Audios ermöglichen es

somit, die Lautstärke in Relation zur Entfernung zu modifizieren und diese an die Audiogeräte der Benutzer anzupassen. Wenn der Sound beispielsweise von der linken Seite kommt, wird der linke Lautsprecher entsprechend stärker beansprucht.

Die Parameter der Sound-Attenuation wurden in dieser Arbeit so gewählt, dass sie ein möglichst realistisches Klangbild erzeugen. Beispielsweise erscheinen Geräusche, die weiter entfernten MetaHumans zugewiesen sind, leiser als identische Geräusche, die von MetaHumans in unmittelbarer Nähe des Player-Actors stammen. Diese optimierten Audiodaten wurden vor allem bei den Störanimationen der MetaHumans verwendet, dazu tragen erheblich zur Erhöhung der Immersion bei und unterstützen die angestrebte Realitätsnähe der virtuellen Umgebung.

6. MetaHumans: Technische Details und ihre Herausforderungen

Wie im Kapitel 2.2 schon über die Technologie und Integration der MetaHumans berichtet wurde, wird in diesem Kapitel über die Besonderheiten, technische Details, sowie Herausforderungen und deren Lösungen der MetaHumans berichtet.

Im Abschnitt 6.1 wird der Fokus auf die Anpassung und Individualisierung von MetaHumans gelegt. Epic Games stellt Entwicklern hierfür den MetaHuman-Creator zur Verfügung, ein leistungsfähiges Tool, das die Erstellung und Feinabstimmung von MetaHumans nach spezifischen Anforderungen ermöglicht. In dieser Arbeit wurde der MetaHuman-Creator verwendet, um MetaHumans entsprechend den Anforderungen verschiedener Szenarien zu gestalten [2].

Der Abschnitt 6.2 behandelt die Herausforderungen bei der Umsetzung und Integration von MetaHumans in verschiedenen Szenarien, insbesondere im Hinblick auf die Performance. MetaHumans erfordern erhebliche Rechenleistung und Speicherressourcen, was eine gezielte Optimierung notwendig macht. In diesem Kapitel werden die Herausforderungen bei der effizienten Nutzung von MetaHumans in den Leveln aufgezeigt, sowie Lösungsansätze vorgestellt, um die Performance zu verbessern, etwa durch Speicheroptimierung und gezielte Anpassungen der Modelle und dabei ein flüssiges sowie immersives Simulationserlebnis zu ermöglichen.

6.1 Anpassung und Individualisierung

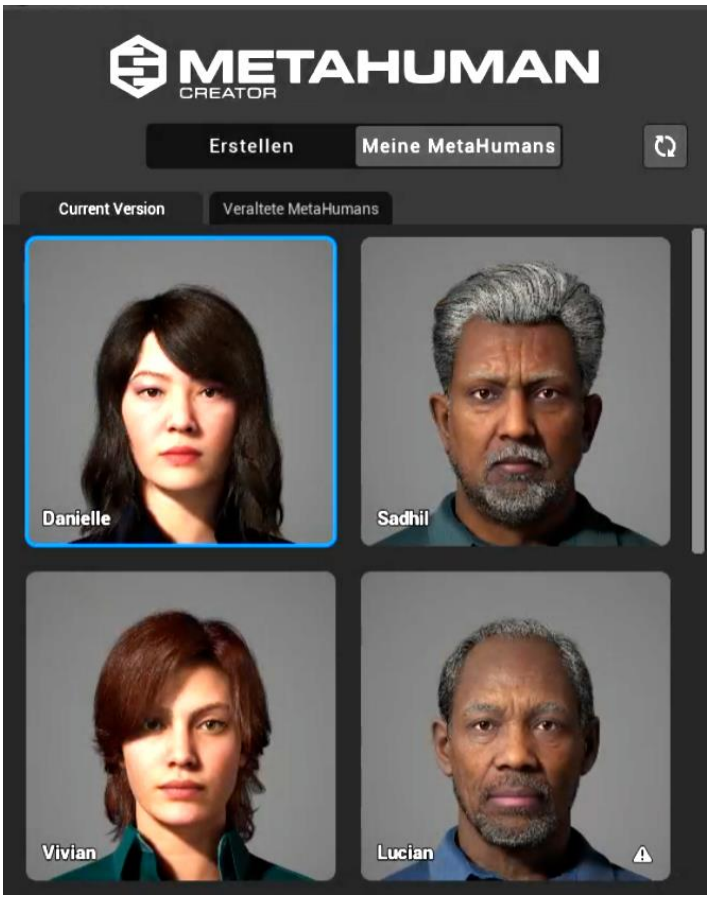

Abbildung 21: Ansicht des MetaHuman-Creators, in dem ein MetaHuman zur Konfiguration ausgewählt wird. Das Bild zeigt die Benutzeroberfläche, auf der alle verfügbaren MetaHumans aufgelistet sind, und ermöglicht die Auswahl und detaillierte Anpassung eines spezifischen MetaHumans für das Projekt.

Epic Games bietet mit dem MetaHuman-Creator ein leistungsstarkes Tool zur Individualisierung von MetaHumans an. Der MetaHuman-Creator ermöglicht es Entwicklern, MetaHumans nach spezifischen Anforderungen zu gestalten (siehe Abbildung 21). In der vorliegenden Arbeit wurde der MetaHuman-Creator verwendet, um die MetaHumans an die Anforderungen verschiedener Szenen anzupassen. Beispielsweise wurden im Meeting-Raum MetaHumans als erwachsene, elegant gekleidete Personen dargestellt, während im Klassenzimmer jugendliche Charaktere mit entsprechender Kleidung, wie Cargo-Hosen, T-Shirts und Sneakers, gestaltet wurden. Zur Anpassung an jugendliche Merkmale wurden bei männlichen MetaHumans Bart, Körpergröße und Mimik modifiziert, um ein jüngeres Erscheinungsbild zu erzeugen, während bei weiblichen MetaHumans auf Make-up verzichtet wurde.

Der MetaHuman-Creator bietet eine Vielzahl an Frisuren, die zur Differenzierung der MetaHumans genutzt wurden [2]. Die Vielfalt der Auswahlmöglichkeiten im MetaHuman-Creator ist bemerkenswert und ermöglicht auch das Mischen verschiedener MetaHumans, um daraus differenziertere MetaHumans zu erstellen. Zudem ermöglicht der MetaHuman-Creator die präzise Anpassung kleinster Details, wie etwa der individuellen Struktur der Hautporen oder der genauen Zahnstellung, was die außergewöhnliche Konfigurierbarkeit und den hohen Grad an Individualisierung unterstreicht.

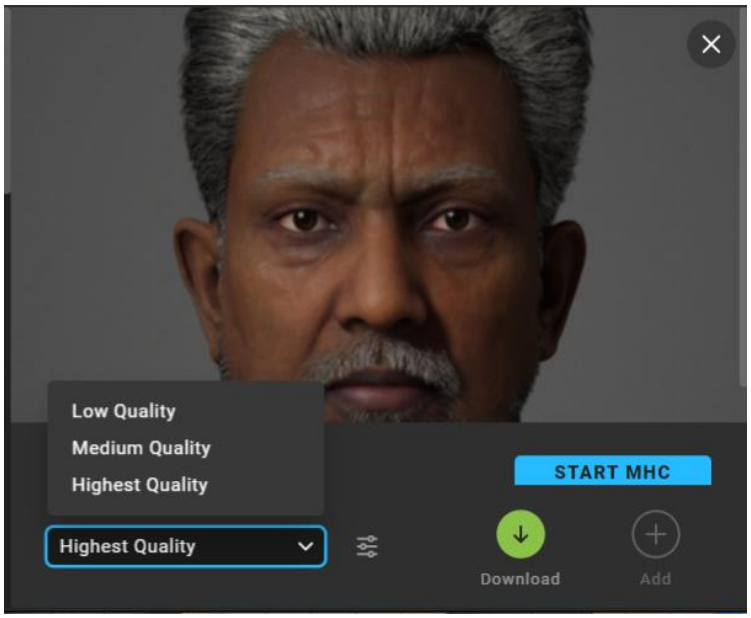

Abbildung 22: Auswahl eines MetaHumans über die Quixel Bridge, inklusive der verfügbaren Auflösungsstufen (2K, 4K, 8K) für den Import in Unreal Engine. Die Bildunterschrift zeigt, wie die Qualitätseinstellungen angepasst werden können, um die Anforderungen des Projekts zu erfüllen.

Nach der Konfiguration der MetaHumans im MetaHuman-Creator besteht die Möglichkeit, diese über die Unreal-Engine-Quixel-Bridge in ausgewählte Projekte zu importieren und weiterzuverwenden. Dabei können die MetaHumans in drei Auflösungsstufen heruntergeladen werden: in niedriger Auflösung (2K), mittlerer Auflösung (4K) und höchster Auflösung (8K) (siehe Abbildung 22) [17].

Auch hat man die Möglichkeit, das Rendern der MetaHumans durch ihr Level of Detail (LOD) festzulegen, welche im Detail-Panels der Blueprints der MetaHumans vorzufinden sind [2]. LOD bezeichnet eine Technik zur Optimierung der 3D-Darstellung, bei der unterschiedliche Detailstufen je

nach Entfernung zur Kamera verwendet werden. Dies reduziert die Rechenlast und verbessert die Rendering-Leistung, indem für entfernte Ansichten weniger detaillierte und für nahe Ansichten detailliertere Versionen geladen werden. Dabei verzichtet man beispielsweise bei Entfernung auf das Rendern der Face-Komponenten wie Augenbrauen oder Wimpern, welches die Immersion beeinträchtigt. Je höher die LOD-Stufen gewählt werden, desto weniger Details werden gerendert [18].

In dieser Arbeit wurden die LOD-Stufen 1 und "Forced LOD-Stufen -1" ausgewählt. LOD-Stufe 1 bietet eine mittlere Detailstufe, die für die meisten Anwendungen ausreichend ist, während "Forced LOD -1" eine höhere Detailstufe erzwingt und somit alle Komponenten des MetaHumans in guter Grafikqualität rendert. Diese Wahl gewährleistet, dass alle visuellen Details der MetaHumans bei nahen und entfernten Ansichten erhalten bleiben.

6.2 Herausforderungen und Lösungsansätze

Durch die ausgeprägte Realitätsnähe der MetaHumans ergaben sich in dieser Arbeit signifikante Performance-Probleme. Besonders in der höchsten Auflösungsstufe (8K) verursachten die MetaHumans einen hohen Ressourcenverbrauch, der zu spürbaren FPS-Drops und Rendering-Problemen führte. Diese Probleme machten die Echtzeit-Integration der MetaHumans praktisch unmöglich. Zudem benötigen die MetaHumans bis zu über 1 GB Speicher pro Einheit, was die Anzahl der einsetzbaren MetaHumans im vorliegenden Projekt erheblich begrenzte.

Das MetaHuman-Plugin war zum Zeitpunkt des Schreibens dieser Arbeit noch in der Entwicklungsphase und als experimentelles Plugin, innerhalb der Unreal Engine klassifiziert. Dies führte während der Entwicklung des Projektes oft zu Problemen bei der vollständigen Darstellung der Gesichtskomponenten der MetaHumans sowie zu häufigen Abstürzen der Unreal Engine. Diese Schwierigkeiten wurden durch den Wechsel auf eine ältere Version der Unreal-Engine 5, konkret Version 5.2, oder durch das Erstellen eines neuen Projekts und die Migration der Daten aus dem älteren Projekt in das aktuelle Projekt erfolgreich behoben.

Die Renderprobleme und die FPS-Drops wurden durch die Wahl der niedrigsten Auflösungsstufe (2K) sowie durch das Deaktivieren der Schattenwurf-Optionen für die MetaHumans gelöst [18]. Eine weitere Herausforderung bestand in der Integration der MetaHumans in den Hörsaal, da eine große Anzahl von MetaHumans zu FPS-Drops und Rendering-Problemen führte. Daher wurde letztlich entschieden, die Anzahl der MetaHumans zu reduzieren und nicht alle Plätze im Hörsaal mit MetaHumans zu besetzen. Zudem besteht die Möglichkeit, MetaHumans anhand ihres Level-of-Detail (LOD) zu optimieren. Hierbei könnte die Anzahl der LOD-Stufen erhöht werden, um weiter entfernte MetaHumans effizienter zu rendern, indem beispielsweise auf detaillierte Gesichtskomponenten verzichtet wird. Dies würde jedoch die Realitätsnähe und Immersion der Darstellung beeinträchtigen. Im Hörsaal wurde daher bevorzugt, die Anzahl der MetaHumans zu reduzieren, anstatt die LOD-Stufen zu erhöhen, da eine Erhöhung der LOD-Stufen die Immersion verringern und die große Anzahl an MetaHumans den Speicherbedarf erheblich steigern würde, was das Projekt unpraktikabel groß gemacht hätte.

7. Integration von Virtual Reality

In diesem Kapitel wird die Integration von Virtual Reality (VR) in die Unreal Engine untersucht, insbesondere wie sie den Entwicklungsprozess beeinflusst. Es wird auf die Implementierung der VR-Technologie eingegangen, einschließlich der Steuerungselemente und der Interaktion in der VR-Umgebung. Zudem wird die Synchronisation zwischen VR-Komponenten und der Unreal Engine behandelt, um eine nahtlose und immersive VR-Erfahrung zu gewährleisten.

7.1 Implementierung von VR in Unreal Engine

Die Unreal Engine bietet eine Vielzahl von Templates, darunter das Ego-Shooter-Template, das Third-Person-Template, das Top-Down-Template, das Vehicle-Template, das Handheld-AR-Template sowie das Virtual-Reality-Template (siehe Abbildung 1). Für die vorliegende Arbeit wurde das Virtual-Reality-Template verwendet, da es bereits wesentliche Funktionalitäten für Virtual-Reality-Anwendungen integriert. Dazu gehören beispielsweise ein Pop-up-Menu sowie die Möglichkeit, sich mithilfe des Controller-Sticks zu bewegen, ohne den Kopf physisch bewegen zu müssen. Dieses Template beschleunigte den Entwicklungsprozess dieser Arbeit bedeutend [20].

Trotz dieser vorinstallierten Funktionen erwies sich das Virtual-Reality-Template als unzureichend, um alle spezifischen Anforderungen dieses Projekts zu erfüllen. Daher waren zusätzliche Anpassungen und Optimierungen erforderlich, die im Kapitel 9.3 näher erläutert werden.

Um eine direkte Verbindung zwischen der VR-Brille und der Unreal Engine herzustellen, ist zunächst eine Verbindung zwischen dem Computer und der VR-Brille erforderlich, die in der Regel durch die herstellereigene Software der jeweiligen VR-Brille ermöglicht wird. Zusätzlich ist eine externe Software notwendig, nämlich „SteamVR“, die von Steam bereitgestellt wird [21]. Diese Software wird ebenfalls für die Verbindung zwischen der VR-Brille und dem Computer benötigt und ist essenziell, um die VR-Umgebung in der Unreal Engine nutzen zu können.

7.2 VR-Interaktion und Synchronisation

Die VR-Interaktion in dieser Arbeit ist folgendermaßen konzipiert: Die VR-Brille ermöglicht eine dreidimensionale Darstellung der Szenen in den jeweiligen Leveln. Der VR-Player wird passend in die Szene integriert, um beispielsweise die Simulation eines Vortrags in den entsprechenden Szenarien zu ermöglichen. Dabei kann der Benutzer durch die Bewegungen der VR-Brille den Raum visuell erkunden und sich entweder auf einzelne MetaHumans fokussieren oder das gesamte Szenario betrachten. Eine Bewegung des Spielers innerhalb des Raumes ist jedoch nicht möglich. Stattdessen steuert der Benutzer den Simulationsfluss mittels des Controllers. Durch das Drücken der Tasten Y oder B wird ein Pop-up-Menu aufgerufen, welches Optionen zum Fortsetzen der Simulation, zum Beenden oder zur Rückkehr zum Level-Auswahlbildschirm bietet.

Abbildung 23: Debug-Darstellung der Hände und Controller zur Visualisierung der 3D-Widget-Interaktion im Level-Select-Menu. Diese Abbildung zeigt die Debug-Linien für die Hände und Controller im Level „Level-Select“, die zur besseren Visualisierung der Interaktion mit den 3D-Widgets verwendet werden. Die Debug-Darstellungen erleichtern die Erkennung und Auswahl der 3D-Widgets.

Zur Verbesserung der Interaktion wurde die Darstellung von Debug-Linien für die Controller aktiviert (siehe Abbildung 23), um eine präzisere Auswahl der Menüpunkte im Level „Level-Select“ und im Spiel zu gewährleisten. Dies trägt dazu bei, dass der Benutzer sicherer agieren und seine Auswahl klar nachvollziehen kann. Zudem wird durch das haptische Feedback des Controllers, das standardmäßig durch Vibrationen der Hersteller integriert ist,

ein zusätzliches taktiles Erlebnis bereitgestellt. Audiosignale wie zur Bestätigung der Auswahl wurden eigenständig in die Anwendung integriert, um die Benutzererfahrung weiter zu verbessern, wie bereits im Kapitel 5 erläutert.

Die Darstellung der Hände des Spielers in der virtuellen Umgebung erfolgt durch die direkte Verbindung mit dem Controller, wodurch die Bewegungen der menschlichen Hand exakt nachgeahmt werden. Diese präzise Synchronisation zwischen den realen Handbewegungen des Benutzers und der virtuellen Darstellung fördert die Immersion und trägt zu einem nahtlosen Interaktionserlebnis bei.

7.3 Optimierung der VR-Funktionalitäten

Um die VR-Funktionalitäten und Interaktionen den spezifischen Anforderungen dieser Arbeit anzupassen, wurden zunächst die Debug-Linien der Controller für den linken und rechten Controller aktiviert. Dies erfolgte durch Anpassungen im VR-Pawn-Blueprint, wobei die Option „Show Debug“ in den WidgetInteractionLeft- und WidgetInteractionRight-Komponenten aktiviert wurde [20]. Die Debug-Linien wurden gezielt nur im Level „Level-Select“ sichtbar gemacht, um die Auswahl der Level zu erleichtern, während sie in den anderen Leveln selbst deaktiviert blieben, um die Immersion nicht zu beeinträchtigen. Eine Ausnahme bildet das Pop-up-Menu, bei dessen Aktivierung die Debug-Linien temporär sichtbar werden, um die Interaktion mit der Benutzer-Oberfläche zu erleichtern. Die Implementierung, die diese Funktionalitäten regelt, wird im Kapitel 9.3.3 detailliert erläutert.

Des Weiteren wurde die Bewegung über den Controller deaktiviert, welche im ursprünglichen Template-Level aktiviert war. Diese Änderung erfolgte durch das bewusste Auslassen dieser Funktion in den individuell konfigurierten Leveln innerhalb der jeweiligen Level-Blueprints.

Um die Interaktion der 3D-Widgets mit dem Controller-Input zu ermöglichen, wurden zusätzliche InputActions für die Trigger erstellt, darunter „NewInteract“, „IA_NewTriggerPressLeft“ und „IA_NewTriggerPressRight“. Diese InputActions modellieren die

Triggerbuttons des VR-Controllers und gewährleisten deren Funktionalität zur Interaktion durch die Trigger-Tasten des Controllers [3].

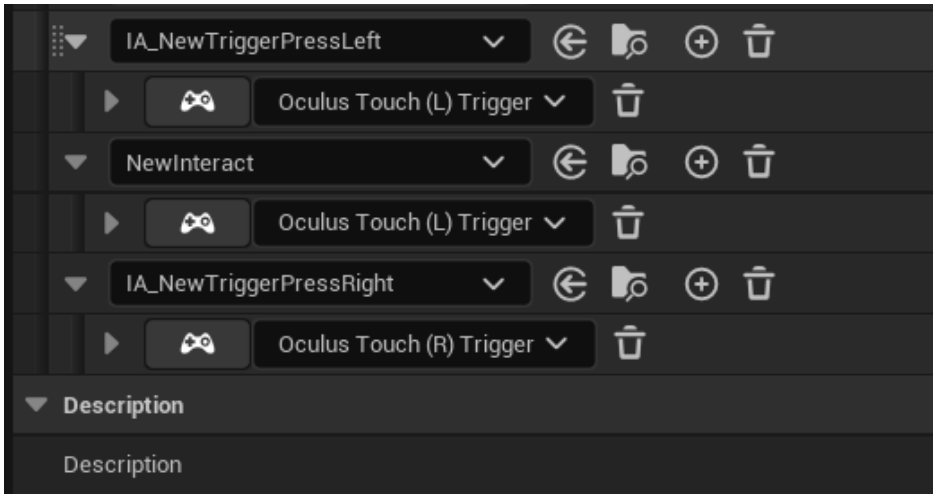

Abbildung 24: Modellierung der Tasten im Input Mapping Context (IMC_Default). Die Abbildung zeigt die Konfiguration der InputActions „NewInteract“, „IA_NewTriggerPressLeft“ und „IA_NewTriggerPressRight“ innerhalb des IMC_Default. Diese InputActions sind den Triggerbuttons des VR-Controllers zugeordnet und ermöglichen die Interaktion mit 3D-Widgets durch die Trigger-Tasten.

Die neuen Eingaben wurden den Input Mapping Contexts wie „IMC_Menu“ und „IMC_Default“ hinzugefügt und entsprechend konfiguriert, wie in der beigefügten Abbildung dargestellt. Dadurch konnte die Modellierung der Input-Actions im Event-Graph des VR-Pawn erfolgen. Eine detaillierte Erläuterung dieses Prozesses findet sich im Kapitel 9.3.3

8. Design der Benutzeroberfläche

In diesem Kapitel werden die Benutzeroberflächen des Projekts detailliert erläutert und der Entwicklungsprozess sowie die Entscheidungsgründe für die gewählte Benutzeroberfläche analysiert. Darüber hinaus wird die Implementierung der Benutzeroberfläche in Kapitel 9.3.3 näher ausgeführt und damit das Verständnis für die gestalterischen und technischen Aspekte vervollständigt.

8.1 Der Entwicklungsprozess der Benutzeroberfläche

Um eine Benutzeroberfläche in der Unreal Engine zu erstellen, wird zunächst ein Widget-Blueprint angelegt. Dieser ermöglicht sowohl das Design der Benutzeroberfläche als auch die Implementierung ihrer Funktionalitäten im Event-Graph [22]. Die Unreal Engine bietet eine breite Palette an Gestaltungsmöglichkeiten, einschließlich der Integration von Elementen wie Bildern, Panels, Checkboxen, Buttons, Scrollboxen, Spinboxen und vielen weiteren Komponenten.

Bei der Entwicklung der Benutzeroberfläche lag der Schwerpunkt auf Benutzerfreundlichkeit und Anpassung an die VR-Umgebung. Das Design wurde darauf ausgerichtet, eine einfache und intuitive Interaktion sicherzustellen, die den spezifischen Anforderungen einer VR-Anwendung gerecht wird.

Um die Interaktion der Benutzeroberfläche innerhalb der VR-Umgebung zu ermöglichen, wurden 3D-Widgets eingesetzt. Diese erlauben es den Benutzern, in der virtuellen Umgebung eine Level-Auswahl durchzuführen. Hierfür wurde ein spezielles Level namens "Level-Select" erstellt, dessen Details und Funktionsweise im Kapitel 8.1.1 ausführlich erläutert werden.

Ein 3D-Widget in Unreal Engine ist ein Interface-Element, das in einer dreidimensionalen Umgebung dargestellt wird [22]. Im Gegensatz zu traditionellen 2D-Widgets, die auf dem Bildschirm angezeigt werden, existieren 3D-Widgets in der virtuellen Welt des Levels und können dort positioniert und manipuliert werden (siehe Abbildung 25). Diese Widgets

können wie reguläre UI-Elemente verwendet werden, z.B. um Buttons, Schieberegler oder Textfelder

Abbildung 25: Platzierung und Visualisierung der 3D-Widgets (Level-Select-Buttons) im Level „Level-Select“ zur Interaktion mit dem VR-Controller.

zu erstellen, aber sie sind in die 3D-Umgebung integriert und reagieren auf die Benutzerinteraktionen in der virtuellen Welt. Dies ermöglicht eine immersive und intuitive Interaktion innerhalb der VR-Umgebung.

Um die 3D-Widgets im Level zu platzieren, wurde ein spezieller Actor-Blueprint erstellt, dem ein Widget zugewiesen wurde. Dadurch konnte das im Widget-Blueprint entworfene und implementierte Interface als 3D-Widget im Level integriert werden. Mithilfe eines Widget-Switchers wurde die Funktionalität ermöglicht, zwischen verschiedenen Menüs innerhalb des 3D-Widgets zu wechseln. So konnte beispielsweise zwischen dem Level-Select-Button, dem MetaHuman-Select-Menu und dem Störanimation-Konfigurations-Menu intern im jeweiligen 3D-Widget gewechselt werden. Die Darstellung der 3D-Widgets ist in Abbildung 25 ersichtlich.

8.1.1 Level-Select-Menu

Wie im Kapitel 8.1 beschrieben, wurde ein spezielles Level mit der Bezeichnung "Level-Select" entwickelt. Die zentralen Elemente dieses Levels bestehen aus drei 3D-Widgets, die jeweils ein anderes auswählbares Level repräsentieren: Hörsaal, Meetingraum und Klassenzimmer. In Abbildung 25 sind die 3D-Widgets dargestellt.

Jedes dieser Widgets fungiert als interaktiver Button, dem ein Bild des jeweiligen Levels zugewiesen ist, welches das entsprechende Level visuell

präsentiert. Durch die Interaktion mit einem dieser 3D-Widgets wird der Benutzer mithilfe des Widget-Switchers zum MetaHuman-Select-Widget weitergeleitet, das zusätzliche Auswahlmöglichkeiten bietet.

8.1.2 MetaHuman-Select-Menu

Abbildung 26: Darstellung des MetaHuman-Select-Menu. In dieser Ansicht werden verschiedene MetaHumans als auswählbare Optionen präsentiert. Der Benutzer kann einen MetaHuman auswählen, um ihn für Störanimationen im ausgewählten Level zu konfigurieren. Die MetaHumans werden in Form von Bild-Buttons dargestellt, die eine visuelle Rückmeldung zur getroffenen Auswahl geben. Über die Pfeil-Buttons kann zu weiteren MetaHumans navigiert werden.

Im MetaHuman-Select-Menu hat der Benutzer die Möglichkeit, einen MetaHuman auszuwählen, der potenzielle Störanimationen im zuvor gewählten Level ausführt. Die Auswahl erfolgt über Bild-Buttons, denen jeweils ein Bild des jeweiligen MetaHuman zugeordnet ist (siehe Abbildung 26). Durch das Anklicken eines dieser Buttons wird der Benutzer zum Störanimation-Konfigurations-Menu des ausgewählten MetaHumans weitergeleitet. Dieses Menü wird im folgenden Kapitel detailliert erläutert.

In diesem Menü hat der Benutzer zusätzlich die Möglichkeit, den MetaHuman-Select-Menu abzubrechen und zum Level-Select-Menu zurückzukehren. Außerdem kann das ausgewählte Level nach den Konfigurationen gestartet werden. Über einen Pfeil-Button kann der Benutzer zu weiteren MetaHumans navigieren, die nicht auf der aktuellen Seite des MetaHuman-Select-Menus angezeigt werden.

Die Gestaltung des MetaHuman-Select-Menus wurde bewusst so konzipiert, dass die Interaktion mit den VR-Controllern erleichtert wird. Durch die Darstellung der MetaHumans in Form von Bildern erhält der Benutzer eine visuelle Rückmeldung über die Auswahl des gewünschten MetaHumans und dessen Konfiguration. Diese Designwahl trägt zur Verbesserung der Benutzerfreundlichkeit bei und vermittelt dem Benutzer ein sicheres Gefühl bei der Interaktion.

8.1.3 Störanimation-Konfigurations-Menu

Abbildung 27: Störanimation-Konfigurations-Menu. Das Menü erscheint nach der Auswahl eines MetaHumans im MetaHuman-Select-Menu und ermöglicht die Konfiguration spezifischer Zeiten für die Störanimationen.

Im Störanimation-Konfigurations-Menu, wie in Abbildung 27 dargestellt, hat der Benutzer die Möglichkeit, spezifische Störanimationen zu aktivieren und die Zeiten festzulegen, zu denen diese Animationen stattfinden sollen. Mit einem Plus-Button können zusätzliche Zeitpunkte definiert werden, während der Minus-Button die zuletzt hinzugefügte Zeit aus der Scrollbox entfernt. Die Konfiguration ermöglicht die genaue Festlegung von Minuten und Sekunden für die Störanimationen. Der Häkchen-Button bestätigt die festgelegten Zeiten, während der Reset-Button alle ausgewählten Werte löscht. Der Zurück-Button unten links führt ohne Bestätigung der Werte zurück zum MetaHuman-Select-Menu. Mit dem Bestätigen-Button werden die Werte gespeichert und in der Game-Instance abgelegt, sodass sie beim

Start des jeweiligen Levels für den entsprechenden MetaHuman abgerufen und angewendet werden können, auch gelangt man durch das Betätigen des Bestätigen-Buttons wieder zum MetaHuman-Select-Menu.

9. Blueprint-System: Funktionsweise und Implementierung

Das Blueprint-System von Unreal Engine dient der Implementierung von Funktionalitäten, vergleichbar mit der Programmierung von Code. Dieses System ist so konzipiert, dass grundlegende Funktionen wie Schleifen (z. B. for-loops) oder beispielsweise das Casten auf andere Klassenobjekte bereits als vordefinierte Bausteine in Unreal Engine verfügbar sind [3] [24] [25]. Durch das Verbinden dieser Funktionen können komplexe Implementierungen erstellt werden. In diesem Kapitel wird der implementierte Teil des Projekts untersucht, und es wird erläutert, wie die Implementierungen in Abhängigkeit voneinander funktionieren und interagieren.

9.1 Grundlagen des Blueprint-Systems

Das Blueprint-System von Unreal Engine ist ein visuelles Skript- und Programmiersystem, welches es Nutzern ermöglicht, logische Abläufe und Funktionalitäten zu erstellen, ohne tiefgehende Programmierkenntnisse zu benötigen. Es existieren verschiedene Arten von Blueprints, darunter Actor-Blueprints, Level-Blueprints, Widget-Blueprints, Animation-Blueprints sowie MetaHuman-Blueprints. Diese Blueprints setzen sich aus Komponenten wie Meshes und Kollisionsboxen sowie Knotenpunkten (Nodes) zusammen, die miteinander verbunden werden, um komplexe Logiken und Abläufe zu gestalten [24] [25].

Ähnlich wie in traditionellen Programmiersprachen wie C++ bietet das Blueprint-System die Möglichkeit, mit Interfaces, Vererbung, Variablen, Zuweisungen und verschiedenen Datenstrukturen zu arbeiten. Diese basieren auf vordefinierten Funktionalitäten, die von Unreal Engine bereitgestellt werden. Zusätzlich besteht die Möglichkeit, statt Blueprints die Programmiersprache C++ zu verwenden, was insbesondere bei der Implementierung komplexer Algorithmen vorteilhaft ist, da man nicht auf primitive Funktionen beschränkt ist [25].

Die Steuerung der Implementierung innerhalb des Blueprint-Systems erfolgt über die Knoten und Verbindungen, die den Ablauf definieren. Analog zu

herkömmlichen Programmiersprachen gibt es auch hier Rückgabewerte (Return Values) und Methoden, die den Ablauf der Logik bestimmen.

9.2 Aufbau und Struktur der Blueprints

Der größte logische Anteil dieses Projekts entfällt auf die Blueprints der MetaHumans. In diesen Blueprints werden die Animationen mithilfe der Funktion „Create-Level-Sequence-Player“ abgespielt. Die Steuerung der Störanimationen erfolgt über traditionelle If-Bedingungen, die in Blueprints als „Branch“ bezeichnet werden. Diese Bedingungslogik, die auf spezifische Kriterien reagiert, wird im Kapitel 9.3.2 ausführlich erläutert.

Die Widgets-Blueprints steuern die verschiedenen UI-Komponenten. Dies umfasst unter anderem das Wechseln der verschiedenen Widgets innerhalb der 3D-Widgets sowie das Auslesen der Spinboxen im Störanimations-Konfigurations-Menu, das ebenfalls über Ereignisse verwaltet wird.

Im Level-Blueprint werden hauptsächlich die Debug-Funktionen des Controllers gesteuert, die aktiviert oder deaktiviert werden, wenn das Pop-up-Menu eingeblendet oder ausgeblendet wird.

Eine detaillierte Erläuterung des logischen Ablaufs dieser Blueprints erfolgt in Kapitel 9.3.

9.3 Implementierung von Logiken und Steuerungen

In diesem Unterkapitel wird die Implementierung sowie der logische Ablauf der verschiedenen Komponenten, sowie die Blueprints detailliert erläutert.

9.3.1 Die Game-Instance

Die Game-Instance in Unreal Engine dient als zentraler Speicherort für persistente Daten, die während der gesamten Laufzeit einer Anwendung erhalten bleiben sollen [3]. In diesem Projekt wird die Game-Instance genutzt, um die im Störanimations-Konfigurations-Menu für jeden ausgewählten MetaHuman festgelegten Werte zu speichern. Diese Werte werden anschließend im jeweiligen Level innerhalb der MetaHuman-Blueprints verarbeitet.

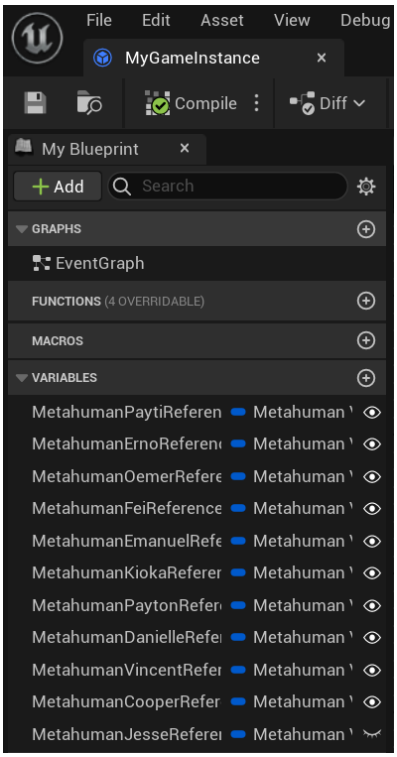

Abbildung 28: Darstellung der Variablen in der Game-Instance. Die Abbildung zeigt die verschiedenen Variablen, die in der Game Instance verwendet werden, um die MetaHuman-Referenzen und die konfigurierten zeitlichen Werte der Störanimationen zu speichern.

Jeder MetaHuman verfügt über eine eigene Referenz in der Game-Instance (siehe Abbildung 28). Um die spezifischen Variablen eines MetaHumans zu speichern, wurde eine Struct erstellt, die zwei Boolean-Variablen und zwei Float-Arrays enthält. Die Boolean-Variablen dienen zur Aktivierung der Störanimationen, während die Float-Arrays die im Störanimations-Konfigurations-Menu festgelegten Zeiten speichern, die den jeweiligen aktivierten Animationen zugewiesen werden.

Damit diese Funktionalität wirksam wird, muss die neu erstellte Game-Instance in den World Settings als aktuelle Game-Instance festgelegt werden. Die selbst definierte Game-Instance hat im Projekt den Namen „MyGameInstance“.

9.3.2 Animationssteuerung der MetaHumans

Wie bereits erläutert, werden die Animationen der MetaHumans durch die Funktion „Create-Level-Sequence-Player“ abgespielt [14]. Jeder MetaHuman besitzt mindestens zwei Variablen vom Typ „MovieSceneSequencePlayer“, denen die jeweiligen Animationen zugewiesen sind. Diese Animationen können durch die Verbindung zum

„Play-Knoten“ abgespielt werden. Die Anzahl der Variablen vom „Typ Movie-Scene-Sequence-Player“ variiert je nach Anzahl der möglichen Animationen des MetaHuman. Zudem gibt es eine Variable vom Typ „float“ namens „ElapsedTime“, die die verstrichene Zeit in Sekunden misst, sowie eine Funktion namens „ComputeElapsedTime“, die diese Zeit berechnet.

Implementierung der Animationssteuerung in den MetaHuman-Blueprints

1. **Zugriff auf die Game-Instance:**

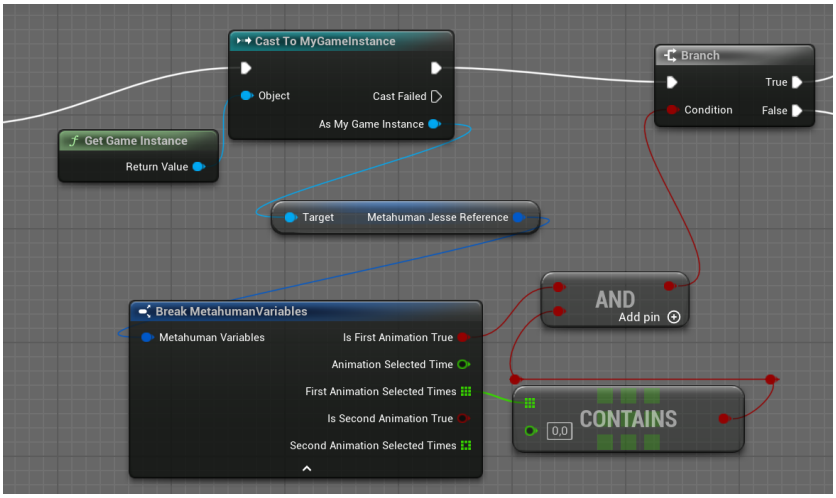

Abbildung 29: Blueprint-Diagramm zur Darstellung des Castings zur Game Instance und der Branch-Bedingung. Diese Abbildung illustriert den Ablauf eines Blueprints, in dem zunächst eine Game Instance referenziert wird. Anschließend wird mithilfe einer Branch-Bedingung überprüft, ob bestimmte Kriterien erfüllt sind, um die darauffolgenden Aktionen auszuführen.

Die Implementierung beginnt mit der Referenzierung der Game-Instance, um die für die Störanimationen festgelegten Werte auszulesen (siehe Abbildung 29). Dies erfolgt durch die Funktion „Cast to MyGameInstance“, um Zugriff auf die Game-Instance zu erhalten. Anschließend wird die Objektreferenz des jeweiligen MetaHuman mit dieser Game-Instance verknüpft. Die Funktion „Break-MetaHumanVariables“ wird aufgerufen, wobei MetaHumanVariables die im Kapitel 9.3.1 beschriebene Struktur darstellt. Hierdurch wird Zugriff auf die im Störanimations-Konfigurations-Menu ausgewählten Werte ermöglicht.

2. **Überprüfung der Störanimationen:** Die Steuerung des Abspielens der Störanimationen erfolgt über Branch-Knoten, die Bedingungen evaluieren. Die Bedingung basiert auf dem Boolean-Wert der Störanimation, der mit dem entsprechenden Wert aus der Game-Instance verglichen wird. Die

Werte im Array werden mittels der Funktion „Contains“ überprüft, die feststellt, ob der aktuelle Wert der Variable „ElapsedTime“ im Array enthalten ist. Diese Überprüfung liefert einen Boolean-Wert.

3. **Steuerung des Abspielens:** Der erhaltene Boolean-Wert wird durch eine AND-Verknüpfung mit dem entsprechenden Boolean der Game-Instance kombiniert. Diese Verknüpfung bildet die Bedingung für den Branch-Knoten, der das Abspielen der Störanimation steuert. Zu Beginn erfolgt die Abfrage der Störanimation bei „ElapsedTime = 0“. Darauffolgend werden die weiteren Abfragen der Störanimationen anhand des aktuellen Werts von „ElapsedTime“ durchgeführt. Weitere Branch-Knoten werden abhängig von der Anzahl der verfügbaren Störanimationen des jeweiligen MetaHuman hinzugefügt, um die weiteren Störanimationen zu steuern.

4. **Einsatz des Set-Timer-By-Event-Knotens:**

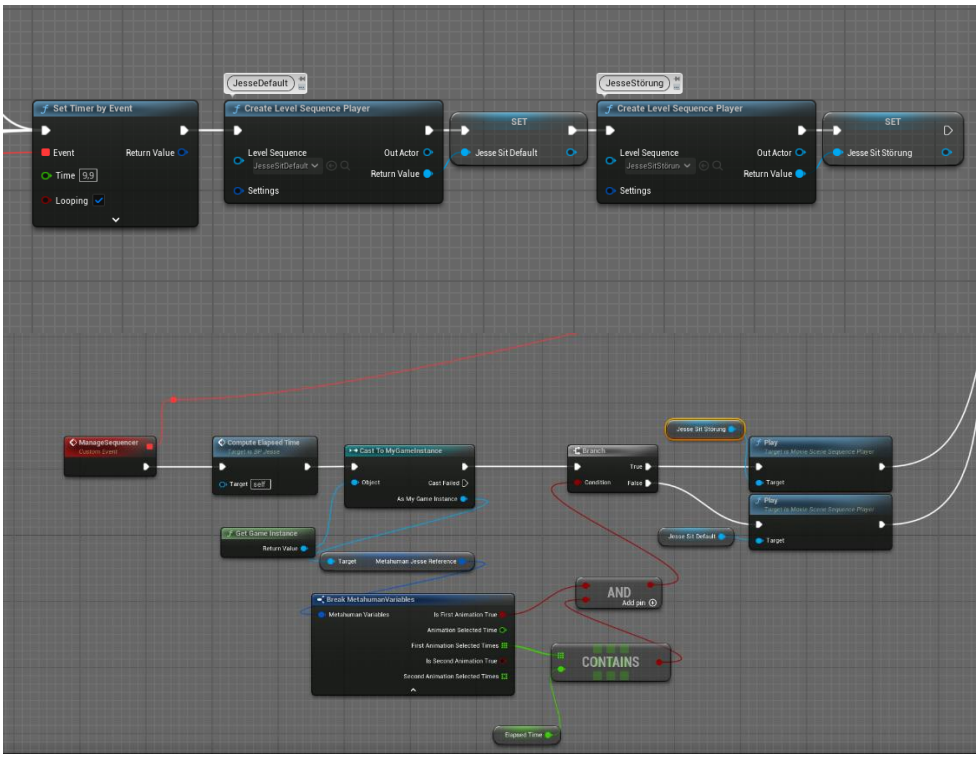

Abbildung 30: Blueprint-Diagramm zur Verwendung von „Set-Timer-by-Event“. Diese Abbildung zeigt die Implementierung eines Timers, der alle 10 Sekunden das Event „ManageSequencer“ auslöst. Dieses Event ist dafür verantwortlich, eine Sequenz von Animationen zu steuern, die in regelmäßigen Abständen ausgeführt werden sollen. Der Timer gewährleistet eine präzise zeitliche Abfolge der jeweiligen Animation.

Eine zentrale Rolle in der Implementierung spielt der Knoten „Set-Timer-By-Event“, der ein konfiguriertes Event in festgelegten Abständen erneut ausführt, um das Abspielen der Störanimationen zu steuern (siehe

Abbildung 30). Die Zeitabstände können im „Time-Feld“ des Knotens „Set-Timer-By-Event“ festgelegt werden. In der Regel beträgt dieser Abstand 10 Sekunden, wobei die Dauer der Animationen berücksichtigt wird. Anschließend ruft der „Set-Timer-By-Event-Knoten“ in zeitlichen Abständen das Event auf, welche die Abfragen von den Punkten 2 und 3 wiederholt und die jeweilige Animation abspielt.

5. **Erklärung der Wahl des Create-Level-Sequence-Players:** Die Wahl der Methode „Create-Level-Sequence-Player“ wurde getroffen, da die Animationen innerhalb der Level-Sequencers definiert sind. Diese Methode ermöglicht eine nahtlose Integration und Verwaltung der Animationen innerhalb der Level-Sequencers, wie beispielsweise das Abspielen.

Bei der Entwicklung dieser Implementierung wurde besonderes Augenmerk auf seine Einfachheit, Wartbarkeit, Erweiterbarkeit und Lesbarkeit gelegt.

9.3.3 Implementierung und Steuerung der Benutzeroberfläche

Die Widget-Blueprints, die zur Entwicklung der Benutzeroberfläche verwendet wurden, verfügen über einen Design-Modus und einen Event-Graph-Modus [23]. Im Design-Modus wird, wie bereits erwähnt, die Benutzeroberfläche gestaltet, während im Event-Graph-Modus die zugehörige Funktionalität implementiert wird. Für die Buttons wurden in dieser Arbeit überwiegend die On-Clicked-Events genutzt, das heißt, beim Klicken dieser Buttons wird die jeweilige im Event-Graph definierte Funktion ausgeführt.

Die meisten On-Clicked-Events dienten lediglich dem Wechseln der Widgets. Die Funktionalität des Widget-Switchers wurde bereits in Kapitel 8 erläutert. Die Implementierung dieser Funktionalität erwies sich als wenig komplex.

Eine anspruchsvollere Implementierung eines On-Clicked-Events erfolgte im Störanimations-Konfigurations-Menu der MetaHumans, insbesondere bei der Festlegung und Verwaltung der Zeitwerte für die Störanimationen.

Um die Spinboxen in die Scrollbox, in der sie definiert sind, einfügen zu können, wurde ein spezieller Widget-Blueprint namens „SpinboxWidget“ erstellt, der diese Spinboxen darstellt. Dieser Widget-Blueprint enthält

lediglich zwei Spinboxen und einen Text, der als Trennzeichen zwischen den Spinboxen dient.

Das Hinzufügen dieser Spinboxen wurde durch das On-Clicked-Event des „+“-Buttons realisiert. Dabei wird mithilfe der Funktion „Construct SpinboxWidget“ das erstellte Widget namens „SpinboxWidget“ erzeugt und der Scrollbox als Child-Element zugewiesen.

Das Entfernen der Spinboxen wurde durch das On-Clicked-Event des „–“-Buttons umgesetzt. Hierbei wird die primitive Funktion „Remove Child at“ verwendet, um das zuletzt hinzugefügte Spinbox-Widget aus der Scrollbox zu entfernen.

Das Störanimations-Konfigurations-Menu verfügt für jede Störanimation über eine lokale Float-Array-Variable sowie eine Boolean-Variable, die über die Benutzeroberfläche durch die Checkbox oder Spinbox verändert werden können. Beim Betätigen der Häkchen-Buttons der jeweiligen Störanimation wird folgende Implementierung ausgeführt (siehe Abbildung 31):

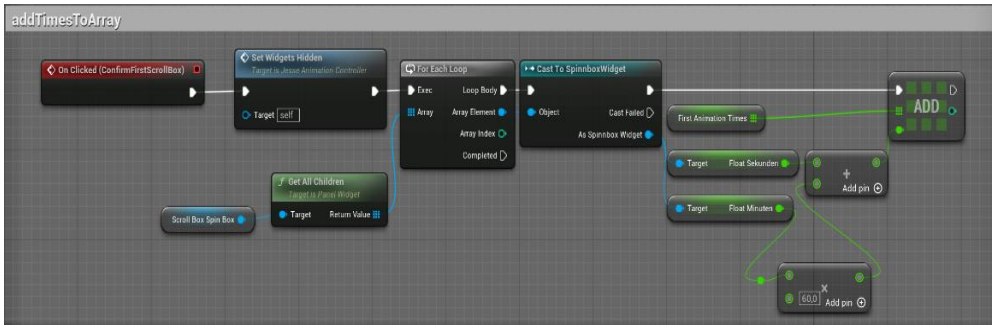

Abbildung 31: Blueprint-Diagramm der Implementierung, der durch das On-Clicked-Event des Häkchen-Buttons ausgelöst wird. Diese Abbildung zeigt den Prozess, bei dem die in die Spinbox eingegebenen Werte vom Benutzer durch Casten auf die Spinbox ausgelesen und in lokale Variablen gespeichert werden. Die Implementierung ermöglicht es, die vom Benutzer festgelegten Werte effektiv zu erfassen und weiterzuverarbeiten, um das Verhalten der VR-Anwendung entsprechend zu steuern.

1. **Iteration durch die Kinder der Scrollbox**: Mithilfe einer „for-each-loop“ wird durch alle Kinder der Scrollbox iteriert.
2. **Cast zum SpinboxWidget**: Um die Werte aus den Spinboxen auslesen zu können, wird in jeder Iteration ein Cast zum „SpinboxWidget“ mithilfe der Funktion „Cast to MySpinboxWidget“ durchgeführt.
3. **Auslesen und Verarbeiten der Werte**: Die Werte aus den Spinboxen werden ausgelesen, wobei zwischen Minuten und Sekunden unterschieden

wird. Die Minuten werden mit 60 multipliziert, um sie in Sekunden umzurechnen, und anschließend mit den in der Spinbox festgelegten Sekunden addiert. Diese berechneten Werte werden dann zur lokalen Array-Variable hinzugefügt, die die zeitlichen Werte für die Störanimationen enthält.

Beim Betätigen des Bestätigungs-Buttons werden die Werte aus den lokalen Variablen, wie dem Boolean oder dem Array, das die zeitlichen Werte für die Störanimationen enthält, der jeweiligen MetaHuman-Referenz in der Game Instance zugewiesen (siehe Abbildung 32). Dabei wird die folgende Implementierung ausgeführt:

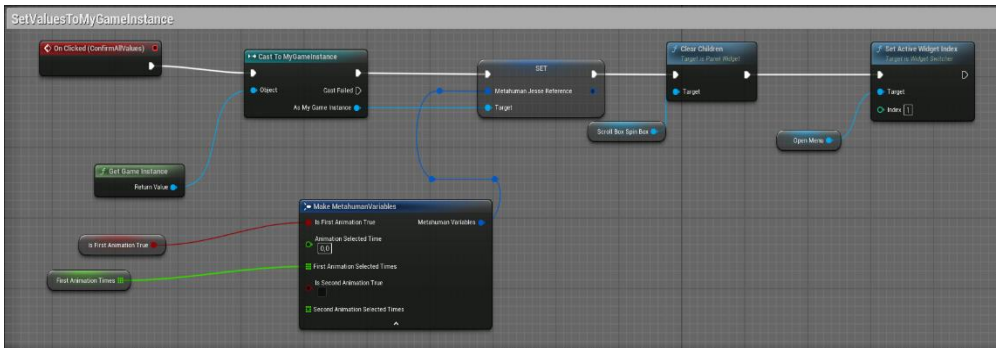

Abbildung 32: Blueprint-Diagramm der Implementierung, der durch das On-Clicked-Event des Bestätigungs-Buttons ausgelöst wird. Diese Abbildung illustriert den Prozess, bei dem die in den lokalen Variablen gespeicherten Werte, die zuvor durch den Häkchen-Button festgelegt wurden, in die globale GameInstance referenziert werden. Die Implementierung stellt sicher, dass die vom Benutzer konfigurierten Werte korrekt in der Game-Instance gespeichert werden, um die entsprechende MetaHuman-Referenz zu aktualisieren.

1. **Zugriff auf die Game-Instance**: Um Zugriff auf die Game-Instance zu erhalten, wird die Funktion „Cast to MyGameInstance“ aufgerufen. Anschließend können die Variablen in der Game-Instance bearbeitet werden, indem die entsprechende MetaHuman-Referenz mit der „Set-Methode“ aufgerufen und die lokalen Variablen den entsprechenden Werten in der Game-Instance zugewiesen werden.

2. **Bereinigung und Navigation**: Daraufhin werden alle Kinder der Scrollbox mithilfe der Funktion „Clear All Children“ entfernt. Anschließend wird das MetaHuman-Select-Menu durch Aufrufen der Funktion „Set Active Widget (Index = 1)“ aktiviert. Diese Funktion basiert auf dem Widget-Switcher.

Der Reset-Button im Störanimations-Konfigurations-Menu entfernt die Werte aus den lokalen und globalen Arrays. Dies erfolgt durch die folgende Implementierung (siehe Abbildung 33):

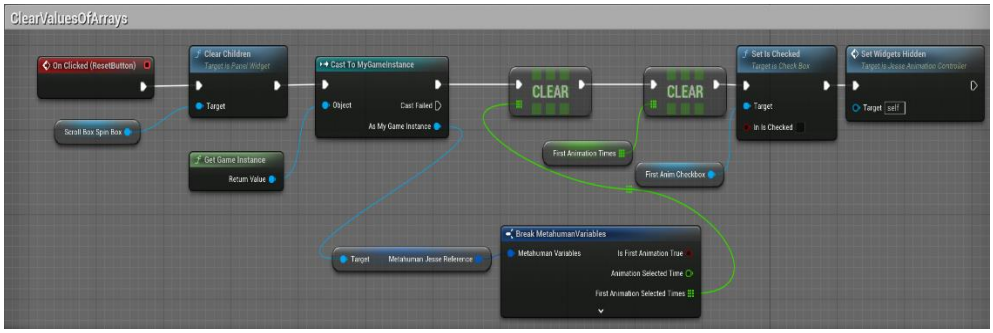

Abbildung 33: Blueprint-Diagramm der Implementierung, der durch das On-Clicked-Event des Reset-Buttons ausgelöst wird. Diese Abbildung verdeutlicht den Ablauf, bei dem der Reset-Button die Werte sowohl im lokalen als auch im globalen Array für die Störanimations-Zeitwert-Arrays zurücksetzt.

1. **Entfernen der Scrollbox-Kinder**: Zunächst werden alle Kinder der Scrollbox mithilfe der Funktion „Clear Children“ entfernt, wobei die Scrollbox als Ziel festgelegt ist.
2. **Zugriff auf die MetaHuman-Referenz**: Durch das Casten zur Game Instance mithilfe der Funktion „Cast to MyGameInstance“ wird die entsprechende MetaHuman-Referenz mit der „Get-Methode“ aufgerufen.
3. **Löschen der Array-Werte**: Anschließend werden alle Werte des Arrays durch die Funktion „Clear“ gelöscht. Diese Funktion wird sowohl für den lokalen als auch für den globalen Array ausgeführt.
4. **Zurücksetzen der Checkboxen und Sichtbarkeit der Scrollbox**: Die Checkboxen der Störanimationen werden auf „false“ gesetzt und die Scrollbox wird unsichtbar gemacht. Nach dem erneuten Setzen der Checkboxen auf „true“ wird die Scrollbox wieder sichtbar.

Eine sehr ähnliche Implementierung, wurde auch für das On-Clicked-Event des „Abbrechen-Buttons“ des MetaHuman-Select-Menu angewendet. Die Implementierung stellt sicher, dass zuvor festgelegte Werte ordnungsgemäß zurückgesetzt werden, um eine unbeabsichtigte Übertragung zwischen verschiedenen Levels zu verhindern. Beispielsweise werden im „Hörsaal“-Level vorgenommene Einstellungen für die MetaHumans bei Betätigung des „Abbrechen-Buttons“ zurückgesetzt. Dies verhindert, dass die Einstellungen

fälschlicherweise im „Klassenzimmer-Level“ übernommen werden. Durch die Reset-Funktion des „Abbrechen-Buttons“ werden die Referenzen der MetaHumans zurückgesetzt, wodurch eine unerwünschte Persistenz der Werte in anderen Levels vermieden wird.

Die Interaktion der VR-Controller mit den 3D-Widgets wurde im Event-Graph des VR-Pawns implementiert [20]. Da die Input-Actions im Event-Graph zu Events transformiert werden können, wurden die im Kapitel 7 erwähnten neu erstellten Input-Actions im VR-Pawn zu Events umgewandelt. In diesen Events wurde den VR-Controllern die Funktionalität eines linken Mausklicks zugewiesen, sodass die VR-Controller-Trigger wie linke Maustasten interagieren konnten (siehe Abbildung 34).

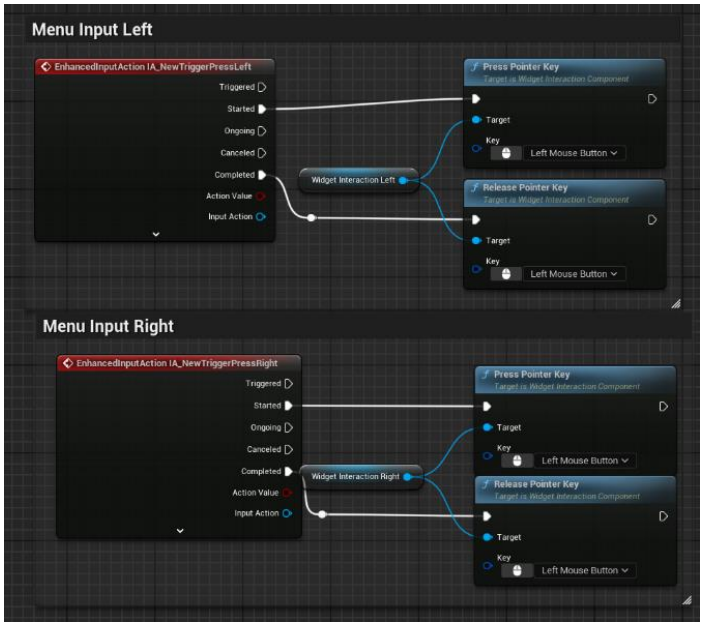

Abbildung 34: Blueprint-Diagramm der Transformation von Input-Actions im VR-Pawn zu Events. Diese Abbildung illustriert den Prozess, bei dem die definierten Input-Actions in den VR-Pawn umgewandelt werden, um als Events zu fungieren. Diese Events sind so konfiguriert, dass sie die Funktionalität einer linken Maustaste übernehmen.

Die Interaktion mit der linken Maustaste wurde bereits von der Unreal Engine implementiert, sodass die Zuweisung dieser Funktionalität es den VR-Controllern ermöglichte, mit den 3D-Widgets zu interagieren [3].

Um die Debug-Linien der VR-Controller zu verwalten, wie im Kapitel 8 bereits erwähnt, wurde eine Erweiterung zur bestehenden Implementierung des VR-Templates hinzugefügt. Diese Implementierung wurde in das bereits vorhandene Pop-Up-Menu des Templates integriert (siehe Abbildung 35)

[20]. Die hinzugefügte Implementierung verweist auf den VR-Pawn und aktiviert die Debug-Linien des Controllers. Das Deaktivieren der Debug-Linien wurde ebenfalls in die bereits vorhandene Implementierung des Pop-Up-Menus integriert.

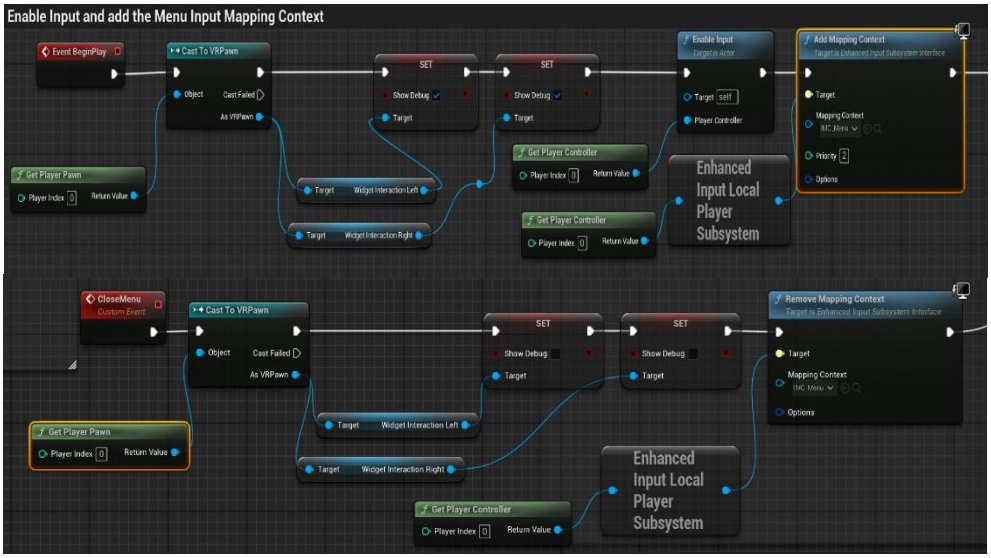

Abbildung 35: Blueprint-Diagramm der Integration der Debug-Linien-Implementierung in das VR-Template. Diese Abbildung zeigt die Erweiterung der bestehenden Implementierung des VR-Templates um die Funktion zur Steuerung der Sichtbarkeit der Debug-Linien über das Pop-up-Menu. Die hinzugefügte Logik ermöglicht es, die Debug-Linien zu aktivieren oder zu deaktivieren, je nachdem, ob das Pop-up-Menu aufgerufen wird.

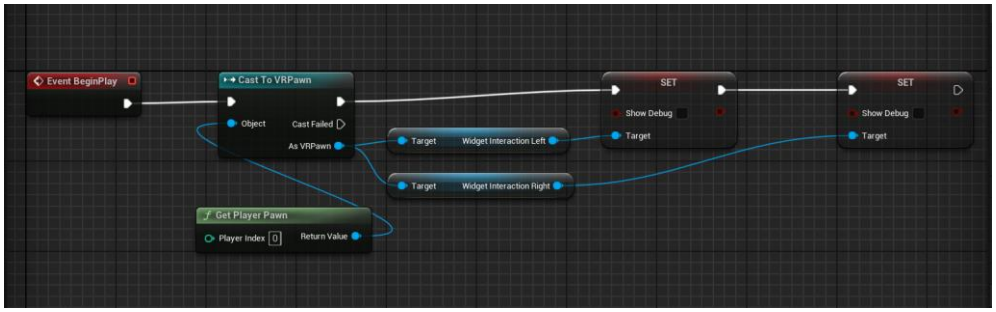

Abbildung 36: Blueprint-Diagramm der Implementierung zur Steuerung der Sichtbarkeit der Debug-Linien in den Levels. Diese Abbildung veranschaulicht die Implementierung in den Level-Blueprints, die sicherstellt, dass die Debug-Linien während des Vortrags in den Levels standardmäßig deaktiviert sind. Sie werden nur dann sichtbar, wenn das Pop-up-Menu aktiviert wird.

Um sicherzustellen, dass die Debug-Linien beim Start des Levels nicht sichtbar sind, wurde im Level-Blueprint des jeweiligen Levels eine Implementierung hinzugefügt, die die Debug-Linien deaktiviert, da diese im Level „Level-Select“ aktiviert sind und man diese Aktivierung beim Levelstart ausschalten möchte (siehe Abbildung 36). Zunächst wurde der VR-Pawn zugewiesen, wodurch Zugriff auf die Widgets „InterActionLeft“ und „InterActionRight“ sowie die Checkboxen der Debug-Linien bestand. Durch

das Casten konnte auf die Checkboxen zugegriffen und diese deaktiviert werden, sodass die Debug-Linien nicht mehr sichtbar waren.

9.4 Das Testen der Funktionalitäten

In diesem Projekt wurden die Implementierungen nicht durch traditionelle Unit-Tests geprüft, wie es in herkömmlichen Programmiersprachen üblich ist. Stattdessen erfolgte die Überprüfung der Funktionalitäten durch manuelle Tests, bei denen beispielsweise die möglichen Eingabekombinationen, wie das Ausfüllen der Spinboxen, das Betätigen von Buttons, sowie das Auslösen der Reset- und Abbrechen-Buttons getestet wurden. Bei der Durchführung dieser Tests wurden erwartete Störanimationen in den entsprechenden Szenen und Leveln überprüft. Entdeckte Fehler wurden durch Print-Strings und Debugging identifiziert und behoben, um sicherzustellen, dass alle Testfälle korrekt abgedeckt und die Implementierungen fehlerfrei sind.

10. Ergebnisse und Evaluation

In diesem Kapitel werden die Ergebnisse dieser Bachelorarbeit präsentiert, diskutiert und evaluiert. Es wird überprüft, inwieweit die definierten Anforderungen erfüllt wurden und welche Herausforderungen während des Entwicklungsprozesses aufgetreten sind. Die wesentlichen Ergebnisse, einschließlich der erstellten virtuellen Levelumgebungen und der Integration der MetaHumans, werden zudem anhand von Abbildungen visualisiert. Diese Abbildungen bieten einen zusätzlichen Einblick in die Umsetzungsdetails der verschiedenen Level und Funktionen.

Abbildung 37: Endgültige Ausgestaltung der virtuellen Umgebungen – Hörsaal, Meetingraum und Klassenzimmer.

Die Erstellung der virtuellen Levelumgebungen war erfolgreich, da alle definierten Anforderungen erfüllt wurden. Es konnte eine realitätsnahe Umgebung geschaffen werden, sowohl visuell als auch physisch, durch den Einsatz von MetaHumans, 3D-Audio und Ambient-Audio (siehe Abbildung 37). Auch die Beleuchtung spielte eine entscheidende Rolle bei der Realisierung einer immersiven Umgebung.

Durch die Entwicklung realitätsnaher Animationen, insbesondere mithilfe von Werkzeugen wie Live-Link und dem MetaHuman-Control-Rig, konnte ebenfalls die Anforderung der Immersion erfüllt werden. Eine zentrale Herausforderung war dabei, die Animationen möglichst lebensnah zu gestalten und passende Störanimationen für die jeweiligen Level zu entwickeln, wie im Kapitel 4 erläutert wird. Die spezifischen Merkmale der einzelnen Level trugen ebenfalls wesentlich zur Gesamtimmersion bei.

Die Interaktion in der VR-Umgebung, unterstützt durch die entwickelten Features, trug wesentlich zur Immersion bei und ermöglichte eine realitätsgetreue Simulation eines Vortragstrainers mit menschenähnlichen Figuren (MetaHumans). Die Implementierung der Steuerung der MetaHuman-Animationen verlief erfolgreich. Die ausgewählten Animationen wurden korrekt den MetaHumans zugeordnet und erwartungsgemäß in den jeweiligen Leveln abgespielt.

Abbildung 38: Darstellung des Levels „Level-Select“ mit integrierten 3D-Widgets zur Auswahl der Level und Konfiguration der Animationen. Diese Benutzeroberfläche unterstützt die intuitive Auswahl und Anpassung der Level sowie die Steuerung der Animationen, um die Benutzerfreundlichkeit zu fördern.

Die Bedienung der 3D-Widgets mittels VR-Controller erwies sich als intuitiv und flüssig. Durch die Zuweisung der Trigger-Buttons als linke Maustasten konnten die VR-Controller nahtlos mit der virtuellen Umgebung interagieren, was die Nutzererfahrung deutlich verbesserte (siehe Abbildung 38).

Die ursprünglich definierten Anforderungen wurden vollumfänglich erfüllt. Besonders hervorzuheben ist die flexible Steuerung der Animationen (siehe

Abbildung 26), die eine dynamische Anpassung an unterschiedliche Szenarien ermöglicht. Im Vergleich zu bestehenden Lösungen bietet die entwickelte Implementierung eine erhöhte Anpassungsfähigkeit, was sie zu einer vielversprechenden Option für zukünftige Anwendungen macht.

Die Performance der Anwendung war stabil; es traten keine signifikanten Latenzen auf, die die Nutzererfahrung beeinträchtigt hätten. Erste Rückmeldungen von Testnutzern betonten die intuitive Bedienbarkeit und die klare Strukturierung der Benutzeroberfläche. Darüber hinaus trug die durch diese Arbeit entwickelte Immersion erheblich zur Effektivität des Projekts als Vortragstrainer bei.

11. Zusammenfassung und Ausblick

In diesem Kapitel sollen zunächst die erreichten Ziele diskutiert und abschließend ein Ausblick auf mögliche, weiterführende Arbeiten gegeben werden. Die vorliegende Arbeit verfolgte das Ziel, eine immersive VR-Umgebung zu entwickeln, in der MetaHumans realitätsnah animiert und gesteuert werden können, um eine Simulation eines Vortragstrainers zu ermöglichen. Ein zentrales Ziel war es zudem, den Benutzern die Möglichkeit zu bieten, die Szenen individuell über das Benutzerinterface, insbesondere das Störanimations-Konfigurations-Menu, zu konfigurieren. Diese Anforderungen wurden erfolgreich umgesetzt. Es gelang, eine interaktive und visuell ansprechende Umgebung zu erschaffen, die durch den Einsatz moderner Technologien wie dem MetaHuman-Control-Rig und Live-Link eine hohe Immersion bietet.

Durch die flexible Konfiguration der Level-Szenarien konnten Benutzer ihre Umgebung und die Störanimationen an ihre spezifischen Bedürfnisse anpassen, was den praktischen Nutzen der Anwendung erheblich steigerte. Die Implementierung der Steuerung der MetaHumans und die nahtlose Integration der VR-Controller ermöglichten eine intuitive Bedienung, die sowohl in der Entwicklung als auch in den ersten Tests überzeugte. Die Immersion und die Flexibilität des Systems, insbesondere bei der Anpassung der Animationen an verschiedene Szenarien, stellen einen besonderen Erfolg dieser Arbeit dar.

Ausblick auf zukünftige Arbeiten

Trotz der erreichten Ziele bietet die Arbeit auch Ansatzpunkte für weiterführende Forschungen und Verbesserungen. Die VR-Technologie, die MetaHumans sowie die zugrunde liegende Unreal Engine sind Technologien, die sich ständig weiterentwickeln. Diese kontinuierlichen Fortschritte können die Möglichkeiten dieser Anwendung in der Zukunft weiter fördern und ausbauen.

Eine interessante Erweiterung wäre die Implementierung von Sprachanalysen, die es ermöglichen, die Stimmlage des Benutzers zu analysieren. Dies könnte Feedback geben, ob der Benutzer eine bestimmte

Wortwahl wiederholt, ob er laut genug spricht oder ob er sich durch die Animationen beeinflussen lässt. Solche Analysen könnten zu noch individuelleren und tiefgreifenderen Auswertungen führen und die Trainingssimulation auf ein neues Niveau heben.

Ein weiterer vielversprechender Ansatz könnte die Integration von KI-basierten Dialogsystemen wie ChatGPT in die MetaHumans sein [26]. Dies würde es ermöglichen, interaktive Gespräche zwischen den MetaHumans und den Benutzern zu führen, was die Immersion und den Realitätsgrad der Simulation erheblich steigern könnte. Durch die Verknüpfung mit solchen fortschrittlichen Tools könnten MetaHumans in der Lage sein, auf Benutzereingaben in natürlicher Sprache zu reagieren und dynamische, kontextabhängige Interaktionen zu ermöglichen. Dies eröffnet neue Möglichkeiten für Trainingsszenarien, bei denen nicht nur die körperliche Präsentation, sondern auch die verbale Interaktion und Reaktion des Benutzers geübt werden kann.

Zukünftige Arbeiten könnten sich darüber hinaus auf die Verfeinerung der Animationen und eine noch realistischere Darstellung der MetaHumans konzentrieren. Eine Erweiterung der VR-Umgebung um zusätzliche Interaktionsmöglichkeiten, wie z. B. Gestensteuerung oder Sprachbefehle, könnte die Immersion weiter erhöhen und neue Anwendungsbereiche erschließen.

Ein weiterer vielversprechender Bereich wäre die Optimierung der Performance, um auch auf weniger leistungsstarken Geräten eine flüssige Darstellung zu gewährleisten. Schließlich könnte die Entwicklung einer umfassenden Testmethodik, ähnlich wie bei traditionellen Softwareprojekten, die Qualitätssicherung in zukünftigen Projekten deutlich verbessern.

12. Abbildungsverzeichnis

13. Literaturverzeichnis

[1] „Wikipedia,“ 24 juni 2024. [Online]. Available: https://de.wikipedia.org/wiki/Virtuelle_Realit%C3%A4t. [Zugriff am 24 juni 2024].

[2] U. Engine, „MetaHuman-for-Unreal-Engine,“ [Online]. Available: https://dev.epicgames.com/documentation/en-us/metahuman/metahuman-for-unreal-engine. [Zugriff am 1 9 2024].

[3] Unreal Engine, „Unreal-Engine 5.2 Documentation,“ [Online]. Available: https://dev.epicgames.com/documentation/de-de/unreal-engine/unreal-engine-5-2-documentation?application_version=5.2. [Zugriff am 24 8 2024].

[4] U. Engine, „Unreal Engine,“ [Online]. Available: https://www.unrealengine.com/de. [Zugriff am 2 9 2024].

[5] Unreal Engine, „Unreal Engine market-place,“ [Online]. Available: https://www.unrealengine.com/marketplace/en-US/store. [Zugriff am 25 8 2024].

[6] Unreal-Engine, „Unreal-Engine animating with Live-Link,“ [Online]. Available: https://dev.epicgames.com/documentation/en-us/metahuman/animating-metahumans-with-livelink-in-unreal-engine. [Zugriff am 29 8 2024].

[7] Unreal Engine, „animating MetaHumans with control-rig in Unreal Engine,“ [Online]. Available: https://dev.epicgames.com/documentation/en-us/metahuman/animating-metahumans-with-control-rig-in-unreal-engine. [Zugriff am 28 8 2024].

[8] Unreal Engine, „Unreal Engine market-place industrial-office,“ [Online]. Available: https://www.unrealengine.com/marketplace/en-US/product/industrial-office-04. [Zugriff am 13 5 2024].

[9] Unreal Engine, „Unreal Engine market-place university-classroom,“ [Online]. Available: https://www.unrealengine.com/marketplace/en-US/product/university-classroom-scene-assets. [Zugriff am 29 5 2024].

[10] Unreal Engine, „light-types-and-their-mobility-in-unreal-engine,“ [Online]. Available: https://dev.epicgames.com/documentation/en-us/unreal-engine/light-types-and-their-mobility-in-unreal-engine?application_version=5.2. [Zugriff am 2 9 2024].

[11] Unreal Engine , „global-illumination-in-unreal-engine,“ [Online]. Available: https://dev.epicgames.com/documentation/en-us/unreal-engine/global-illumination-in-unreal-engine?application_version=5.2. [Zugriff am 2 9 2024].

[12] Unreal Engine , „shadowing-in-unreal-engine,“ [Online]. Available: https://dev.epicgames.com/documentation/en-us/unreal-engine/shadowing-in-unreal-engine?application_version=5.2. [Zugriff am 2 9 2024].

[13] Unreal-Engine, „unreal-engine-facial-capture-with-live-link,“ [Online]. Available: https://dev.epicgames.com/community/learning/tutorials/lEYe/unreal-engine-facial-capture-with-live-link. [Zugriff am 2 9 2024].

[14] Unreal-Engine, „cinematic-workflow-guides-and-examples-in-unreal-engine,“ [Online]. Available: https://dev.epicgames.com/documentation/en-us/unreal-

engine/cinematic-workflow-guides-and-examples-in-unreal-engine?application_version=5.2. [Zugriff am 2 9 2024].

[15] Unreal-Engine, „working-with-audio-in-unreal-engine," [Online]. Available: https://dev.epicgames.com/documentation/en-us/unreal-engine/working-with-audio-in-unreal-engine?application_version=5.2. [Zugriff am 2 9 2024].

[16] Unreal Engine, „cinematic-audio-track-in-unreal-engine," [Online]. Available: https://dev.epicgames.com/documentation/en-us/unreal-engine/cinematic-audio-track-in-unreal-engine?application_version=5.2. [Zugriff am 2 9 2024].

[17] Unreal Engine, „downloading-and-exporting-metahumans-to-unreal-engine-5," [Online]. Available: https://dev.epicgames.com/documentation/en-us/metahuman/downloading-and-exporting-metahumans-to-unreal-engine-5-and-maya. [Zugriff am 2 9 2024].

[18] Unreal Engine, „technical-specifications-for-metahumans," [Online]. Available: https://dev.epicgames.com/documentation/en-us/metahuman/technical-specifications-for-metahumans. [Zugriff am 2 9 2024].

[19] Unreal Engine, „performance-and-scalability-settings for MetaHumans," [Online]. Available: https://dev.epicgames.com/documentation/en-us/metahuman/technical-specifications-for-metahumans/performance-and-scalability-settings. [Zugriff am 2 9 2024].

[20] Unreal Engine, „vr-template-in-unreal-engine," [Online]. Available: https://dev.epicgames.com/documentation/en-us/unreal-engine/vr-template-in-unreal-engine?application_version=5.2. [Zugriff am 2 9 2024].

[21] Unreal-Engine, „developing-for-steamvr-in-unreal-engine," [Online]. Available: https://dev.epicgames.com/documentation/en-us/unreal-engine/developing-for-steamvr-in-unreal-engine?application_version=5.2. [Zugriff am 2 9 2024].

[22] Unreal Engine, „getting-started-with-umg-for-unreal-engine," [Online]. Available: https://dev.epicgames.com/documentation/en-us/unreal-engine/getting-started-with-umg-for-unreal-engine?application_version=5.2. [Zugriff am 2 9 2024].

[23] Unreal Engine, „umg-widget-interaction-components-in-unreal-engine," [Online]. Available: https://dev.epicgames.com/documentation/en-us/unreal-engine/umg-widget-interaction-components-in-unreal-engine?application_version=5.2. [Zugriff am 2 9 2024].

[24] M. R. Sewell, „Blueprints Visual Scripting for Unreal Engine: The Faster, Smarter Way to Build Games and Interactive Experiences," in *Visual Scripting for Unreal Engine*, Indianapolis, IN, USA:, Addison-Wesley Professional , 2017.

[25] Unreal Engine , „unreal-engine-programming-and-scripting," [Online]. Available: https://dev.epicgames.com/documentation/en-us/unreal-engine/unreal-engine-programming-and-scripting?application_version=5.2. [Zugriff am 2 9 2024].

[26] Unreal Engine, „connecting-metahuman-to-chatgpt-success," [Online]. Available: https://forums.unrealengine.com/t/connecting-metahuman-to-chatgpt-success/760356/16. [Zugriff am 2 9 2024].

14. Danksagung

Diese Arbeit wäre ohne die Unterstützung und das Vertrauen vieler Menschen nicht möglich gewesen. Mein tiefster Dank gilt meinem Betreuer, Prof. Dr.-Ing. Thorsten Thormählen für seine unerschütterliche Unterstützung, seine wertvollen Anregungen und sein kontinuierliches Engagement. Durch seine Expertise und konstruktiven Kritiken hat er maßgeblich zum Erfolg dieser Arbeit beigetragen.

Besonderer Dank gebührt auch meiner Familie, die mir durch ihre Liebe und Geduld die Kraft gegeben hat, diese Arbeit zu vollenden. Ihre unermüdliche Unterstützung und ihr Glaube an mich waren eine konstante Quelle der Motivation.

Ein ebenso herzlicher Dank gilt meinen Freunden und Kommilitonen, die mir nicht nur in fachlichen Fragen zur Seite standen, sondern auch in schwierigen Momenten Mut gemacht haben. Ihr Zuspruch und eure Bereitschaft, immer wieder als Testnutzer zu fungieren, haben mir sehr geholfen.

Nicht zuletzt möchte ich den Entwicklern von Unreal Engine und den Teams hinter den innovativen Technologien danken, die in dieser Arbeit zum Einsatz kamen. Ihre bahnbrechende Arbeit hat den Grundstein für viele der hier umgesetzten Ideen gelegt.

Diese Bachelorarbeit ist nicht nur das Ergebnis monatelanger harter Arbeit, sondern auch ein Zeugnis der Unterstützung, die ich von vielen Seiten erfahren durfte. Dafür bin ich zutiefst dankbar.