Fakultät Elektrotechnik Feinwerktechnik
Informationstechnik

Entwicklung eines Third-Person-XR-Plattformers mit dualer Eingabeinteraktion

Bachelorarbeit im Studiengang Media Engineering

vorgelegt von

David Zargartalebi

Matrikelnummer 3572858

Sommersemester 2025 - Abgabedatum: 10. August 2025

Erstgutachter:	Prof. Dr. rer. nat. Matthias Hopf
Zweitgutachter:	Prof. Dr. Ralph Lano

ohm Technische Hochschule Nürnberg

Hinweis: Diese Erklärung ist mit Originalunterschrift (nicht gescannt) in das Papierexemplar der Abschlussarbeit fest einzubinden. Eine Spiralbindung ist nicht zulässig. Das digitale Exemplar enthält einen Scan der Erklärung mit Unterschrift.

Prüfungsrechtliche Erklärung der/des Studierenden

Angaben des bzw. der Studierenden:

Name: Zargartalebi Vorname: David Matrikel-Nr.: 3572858

Fakultät: efi Studiengang: Bachelor Media Engineering

Semester: 10

Titel der Abschlussarbeit:

Entwicklung eines Third-Person-XR-Plattformers mit dualer Eingabeinteraktion

Ich versichere, dass ich die Arbeit selbständig verfasst, nicht anderweitig für Prüfungszwecke vorgelegt, alle benutzten Quellen und Hilfsmittel angegeben sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe. Außerdem versichere ich, dass der Hauptteil des Papierexemplares und des digitalen Exemplares identisch sind. Das digitale Exemplar enthält, falls gefordert, lediglich weitere Anlagen.

Erlangen, 24.07.2025, D.Zargartalebi

Ort, Datum, Unterschrift Studierende/Studierender

Erklärung der/des Studierenden zur Veröffentlichung der vorstehend bezeichneten Abschlussarbeit

Die Entscheidung über die vollständige oder auszugsweise Veröffentlichung der Abschlussarbeit liegt grundsätzlich erst einmal allein in der Zuständigkeit der/des studentischen Verfasserin/Verfassers. Nach dem Urheberrechtsgesetz (UrhG) erwirbt die Verfasserin/der Verfasser einer Abschlussarbeit mit Anfertigung ihrer/seiner Arbeit das alleinige Urheberrecht und grundsätzlich auch die hieraus resultierenden Nutzungsrechte wie z.B. Erstveröffentlichung (§ 12 UrhG), Verbreitung (§ 17 UrhG), Vervielfältigung (§ 16 UrhG), Online-Nutzung usw., also alle Rechte, die die nicht-kommerzielle oder kommerzielle Verwertung betreffen.

Die Hochschule und deren Beschäftigte werden Abschlussarbeiten oder Teile davon nicht ohne Zustimmung der/des studentischen Verfasserin/Verfassers veröffentlichen, insbesondere nicht öffentlich zugänglich in die Bibliothek der Hochschule einstellen.

Hiermit ☑ genehmige ich, wenn und soweit keine entgegenstehenden Vereinbarungen mit Dritten getroffen worden sind,

☐ genehmige ich **nicht,**

dass die oben genannte Abschlussarbeit durch die Technische Hochschule Nürnberg Georg Simon Ohm, ggf. nach Ablauf einer mittels eines auf der Abschlussarbeit aufgebrachten Sperrvermerks kenntlich gemachten Sperrfrist

von 0 Jahren (0 - 5 Jahren ab Datum der Abgabe der Arbeit),

der Öffentlichkeit zugänglich gemacht wird. Im Falle der Genehmigung erfolgt diese unwiderruflich; hierzu wird der Abschlussarbeit ein Exemplar im digitalisierten PDF-Format an die Betreuer übermittelt. Bestimmungen der jeweils geltenden Studien- und Prüfungsordnung über Art und Umfang der im Rahmen der Arbeit abzugebenden Exemplare und Materialien werden hierdurch nicht berührt.

Erlangen, 24.7.2025, D.Zargartalebi

Ort, Datum, Unterschrift Studierende/Studierender

Datenschutz: Die Antragstellung ist regelmäßig mit der Speicherung und Verarbeitung der von Ihnen mitgeteilten Daten durch die Technische Hochschule Nürnberg Georg Simon Ohm verbunden. Weitere Informationen zum Umgang der Technischen Hochschule Nürnberg mit Ihren personenbezogenen Daten sind unter nachfolgendem Link abrufbar: https://www.th-nuernberg.de/datenschutz/

Hinweis: In dieser Arbeit wird überwiegend das generische Maskulinum verwendet. Die in dieser Arbeit verwendeten Personenbezeichnungen beziehen sich jedoch - sofern nicht anders kenntlich gemacht - stets auf alle Geschlechter.

Kurzdarstellung

Erweiterte Realität (XR, auch Cross Reality oder xReality genannt) bietet innovative Interaktionsmöglichkeiten in der Spieleentwicklung, erfordert jedoch auch spezifische Designansätze, um eine intuitive und komfortable Nutzungserfahrung zu gewährleisten. Diese Arbeit beschäftigt sich mit der Konzeption, Umsetzung und Evaluation eines Third-Person-XR-Plattformers mit paralleler Controller- und Gestensteuerung.

Im theoretischen Teil der Arbeit werden grundlegende Begriffe aus den Bereichen der digitalen Realitäten erläutert. Es folgt eine Darstellung bestehender Design-Heuristiken und Industriestandards. Diese bilden die Grundlage für das Game Design, in dem unter anderem Spielziele, Regeln, Interaktionsmechaniken, Dramaturgie sowie Inspirationsquellen analysiert und definiert werden.

Darauf aufbauend wird ein spielbarer Prototyp mit der Unity-Engine und dem Meta XR SDK entwickelt. Die im Game Design entworfenen Spielmechaniken und die definierten Design-Heuristiken werden in konkrete funktionale und nicht-funktionale Anforderungen übersetzt und im Entwicklungsprozess umgesetzt. Der Wechsel zwischen augmentierter und virtueller Umgebung wird dabei Bestandteil des Spiels und dessen Handlungsverlaufs sein.

Zur abschließenden Evaluation wird ein Usability-Test mit Beobachtung und Nachbefragung sowie ein standardisierter Fragebogen eingesetzt, um qualitative und quantitative Daten zu erheben. Ziel ist es, die Spielbarkeit und Nutzungsfreundlichkeit des Prototyps im Kontext der definierten Designprinzipien zu bewerten.

Diese Arbeit leistet einen Beitrag zur Gestaltung benutzerfreundlicher XR-Anwendungen, indem praxisnahe Erkenntnisse zur Anwendung von Usability-Heuristiken im Rahmen eines spielerischen, immersiven Erlebnisses geliefert werden.

Abstract

Extended Reality (XR, also known as Cross Reality or xReality) offers innovative interaction possibilities in game development but also requires specific design approaches to ensure an intuitive and comfortable user experience. This thesis focuses on the conception, implementation, and evaluation of a third-person XR platformer with parallel controller and gesture-based input.

The theoretical part introduces fundamental concepts from the field of digital realities. This is followed by a presentation of existing design heuristics and industry standards, which serve as the foundation for the game design. Within this, gameplay goals, rules, interaction mechanics, narrative elements, and sources of inspiration are analyzed and defined.

Based on this, a playable prototype is developed using the Unity engine and the Meta XR SDK. The mechanics outlined in the game design and the defined heuristics are translated into concrete functional and non-functional requirements and implemented during development. The transition between augmented and virtual environments becomes an integral part of the game and its narrative structure.

For the final evaluation, a usability test involving observation and post-session interviews, as well as a standardized questionnaire, is conducted to collect qualitative and quantitative data. The aim is to assess the playability and usability of the prototype in the context of the defined design principles.

This thesis contributes to the development of user-friendly XR applications by providing practical insights into the application of usability heuristics within an engaging and immersive experience.

Inhaltsverzeichnis

1. Einleitung

Technologien wie Virtual, Augmented und Mixed Reality sind in den letzten Jahren zunehmend präsent in verschiedenen Lebens- und Arbeitsbereichen geworden. Dies reicht von der Industrie und Medizin bis hin zur Unterhaltungsbranche. Insbesondere im Bereich von Videospielen ermöglichen die verschiedenen Formen der digitalen Realitäten immersive und interaktive Erlebnisse. [1]

Besonders Head-Mounted Displays (HMDs) wie die Meta Quest 3, die Apple Vision Pro oder Microsofts HoloLens haben neue Interaktionsformen hervorgebracht, beispielsweise durch Hand-Tracking, Tiefensensorik oder die Integration von Raumklang. Das Vorantreiben der technologischen Entwicklung durch verschiedene Hersteller bringt jedoch auch Probleme mit sich:

So gibt es eine inkonsistente Nutzung der Begriffe AR, VR, MR und XR. Während Microsoft und Meta ihre HMDs beispielsweise als „Mixed-Reality"-Headsets vermarkten, spricht Apple von „Augmented Reality". Diese Mehrdeutigkeit von Begriffen zeichnet sich auch in der akademischen Literatur ab. [2]

Die technologischen Unterschiede zwischen den Herstellern führen außerdem zu Herausforderungen bei der Gestaltung konsistenter Nutzungserfahrungen. Experten bemängeln die fehlende Vereinheitlichung von plattformübergreifenden Interaktionsprinzipien [3] und weisen darauf hin, dass diese Entwicklungen auch Anpassungen im Verständnis von UX-Design erfordern. Etablierte Muster und Design-Guidelines sind weitgehend auf Anwendungen für 2D-Bildschirme ausgelegt und decken die Anforderungen für XR-Anwendungen nur bedingt ab. [4]

Diese Bachelorarbeit widmet sich der Entwicklung eines XR-Prototyps für die Meta Quest 3, bei dem etablierte und für XR entwickelte Design-Heuristiken sowie Best Practices und Industriestandards aufgegriffen und in einer praktischen Anwendung umgesetzt werden. Ziel ist die Konzeption, Entwicklung und Evaluation eines Third-Person-Plattformer-Spiels, das eine hybride Steuerung durch Controller und Handgesten bietet sowie AR- und VR-Elemente miteinander verbindet. Der Spieler steuert dabei einen Charakter mit einem einhändigen Controller durch eine Spielwelt, die sich um ihn herum aufbaut. Mit der freien Hand kann aktiv in die Umgebung eingegriffen werden, etwa durch das Verschieben von Plattformen oder das Auslösen von Aktionen über Hand-Posen.

Ein zentrales Designelement ist der Übergang zwischen realer und virtueller Welt, der durch ein Portal dargestellt wird. Der Wechsel zwischen AR- und VR-Umgebung wird so in einem erzählerischen Kontext ermöglicht, um ein immersives Spielerlebnis zu schaffen.

Die Arbeit verfolgt dabei folgende Ziele:

- Untersuchung und Anwendung von Usability-Heuristiken und XR-spezifischen Designprinzipien in einem spielerischen Kontext.
- Entwicklung eines funktionsfähigen XR-Prototyps mit dualer Eingabeinteraktion und dynamischem Wechsel zwischen AR- und VR-Umgebungen.
- Evaluation der Anwendung hinsichtlich Usability, Spielbarkeit und User Experience.

In der Arbeit wird dabei wie folgt vorgegangen: Der **theoretische Rahmen** analysiert bestehende Usability-Heuristiken in Bezug auf XR und greift zusätzlich XR-spezifische Designprinzipien und industrielle Standards auf. Im **Game Design** wird daraufhin die Spielidee ausgearbeitet, eine Zielgruppe definiert und die formalen und dramaturgischen Elemente des Spiels festgelegt. Die **Entwicklung** umfasst die technische Umsetzung. Hierbei werden die zu entwickelnden funktionalen und nicht-funktionalen Anforderungen aus dem Game Design und den dargestellten Design-Heuristiken abgeleitet. Die **Evaluierung** der Anwendung wird abschließend mittels explorativer Usability- und Playtests durchgeführt. Hierbei soll die Anwendung anhand der zuvor definierten Heuristiken und Anforderungen bewertet werden.

Der Prototyp wird für die Meta Quest 3 in der Unity-Engine entwickelt. Dabei kommt das Meta-XR-All-In-One-SDK zum Einsatz, das bereits grundlegende XR-Funktionalitäten bereitstellt. Durch die Kombination von Usability-Grundlagen, Game Design, praktischer Entwicklung und Evaluierung soll ein Beitrag zur nutzerzentrierten Gestaltung zukünftiger XR-Erfahrungen geleistet werden.

2. Theoretische Grundlagen

„Extended Reality“ (XR), zu Deutsch „erweiterte Realität“, wird häufig als Oberbegriff für die immersiven Technologien „Augmented Reality“ (AR, „angereicherte Realität“), „Mixed Reality“ (MR, „gemischte Realität“) und „Virtual Reality“ (VR, „virtuelle Realität“) verwendet [1] [5]. Während AR und VR relativ einheitlich definiert sind, gibt es insbesondere bei den Begriffen MR und XR unterschiedliche Verwendungen. Das Verhältnis der verschiedenen Realitätsformen zueinander wird oft unterschiedlich dargestellt. Dies zeigt sich sowohl in der Industrie, beispielsweise bei den Herstellern von Head-Mounted Displays (HMDs), als auch in der Fachliteratur [2]. Da diese Arbeit eine umfassende XR-Betrachtung vornimmt, ist es zunächst erforderlich, die damit verbundenen Begrifflichkeiten zu klären.

2.1. Virtual Reality

„Als virtuelle Realität [...] wird die Darstellung und gleichzeitige Wahrnehmung einer scheinbaren Wirklichkeit und ihrer physikalischen Eigenschaften in einer in Echtzeit computergenerierten, interaktiven virtuellen Umgebung bezeichnet.“ [6] Dabei wird die reale Umgebung vollständig verdeckt [2], während die virtuelle Umgebung hauptsächlich über visuelle Ausgabemedien dargestellt wird. Allerdings können auch weitere Sinne angesprochen werden, zum Beispiel durch akustische Signale oder haptisches Feedback. VR wird daher als multisensorisch bezeichnet. [1]

Virtuelle Welten werden in der Regel mithilfe von Virtual-Reality-Headsets dargestellt. Dabei wird das stereoskopische, also räumliche, Sehvermögen des Menschen genutzt, indem zwei leicht versetzte Bilder erzeugt und dem jeweiligen Auge zugeführt werden. Eine weitere Form der VR stellt das „Cave Automatic Virtual Environment“ (CAVE) dar. Hierbei handelt es sich um einen Raum, in dem durch Projektionen auf Wände eine virtuelle Umgebung entsteht. In dieser Arbeit wird CAVE jedoch nicht ausführlicher behandelt. [6] [7]

Zwei zentrale Eigenschaften von VR sind Immersion und Interaktion. Immersion (lat. immersio = Eintauchung) beschreibt das Erleben einer virtuellen Umgebung als real. Je stärker das Bewusstsein des Nutzers für die Künstlichkeit der Stimuli in den Hintergrund tritt, desto höher ist der Grad an Immersion. Man spricht in diesem Zusammenhang auch

von (Tele-)Präsenz. Durch Interaktionen mit der virtuellen Umgebung steigert sich das Gefühl, ein Teil dieser Welt zu sein, erheblich. [8] [5]

Das Interaktionsniveau variiert je nach Gerät. Neben den bereitgestellten Interaktionsmöglichkeiten, wie Hand- und Augenverfolgung oder Steuerung per Controller, sind die Freiheitsgrade („Degrees of Freedom“, DoF) von Bedeutung. Diese beziehen sich auf das Headset und gegebenenfalls auf die Controller. Bei drei Freiheitsgraden (3DoF) ist lediglich eine Rotationsverfolgung möglich: Der Nutzer kann sich durch Kopfbewegungen umsehen oder mit den Controllern auf virtuelle Elemente zeigen und diese auswählen. Bei sechs Freiheitsgraden (6DoF) ist neben der Rotationsverfolgung zusätzlich eine Translations- bzw. Positionsverfolgung möglich: Der Nutzer kann sich frei im virtuellen Raum bewegen und die Controller als virtuelle Hände verwenden. [2] [9]

2.2. Augmented Reality

„Bei Augmented Reality [...] handelt es sich um eine Erweiterung der Realität. Dabei verbindet AR die reale und virtuelle Umgebung, indem virtuelle Objekte oder Informationen in das Sichtfeld projiziert werden und somit eine neue Wahrnehmung der Realität geschaffen wird.“ [1] Die reale, physikalische Umgebung bleibt dabei stets Teil der Nutzungserfahrung, was den primären Unterschied zu VR darstellt [2].

Während VR hauptsächlich über HMDs dargestellt wird, ist AR auch auf allgegenwärtigen Geräten, wie Smartphones, weit verbreitet [2]. Die digitalen Inhalte werden hier als Überlagerung des Kamerabildes dargestellt. Nutzer können dann über den Bildschirm mit diesen Inhalten interagieren. Bekannte Beispiele hierfür sind das Spiel *Pokémon Go* oder die *Google Maps Live-View*-Funktion. [5] Neuere AR-Geräte nutzen HMDs, um die Technologie näher an den Körper zu bringen. Spezialisierte Hardware, wie Tiefensensoren, und die Unterstützung von Hand- und Augenverfolgung ermöglichen dabei neue Interaktionsmöglichkeiten, ähnlich zu VR. [2]

2.3. Mixed Reality und Extended Reality

Die Begriffe Mixed Reality und Extended Reality werden sowohl in der Industrie als auch in der Fachliteratur unterschiedlich definiert [2]. Häufig wird AR synonym mit MR verwendet, während andere Quellen AR als eine Unterkategorie von MR betrachten [10]. Extended Reality wird meist als Oberbegriff für alle digitalen Realitätsformen genutzt, jedoch wird der Begriff selbst kontrovers diskutiert. So wird XR auch als „Cross Reality“ oder „xReality“ bezeichnet [1] [2]. In der Literatur lassen sich vier grundlegende Perspektiven zur Klassifikation digitaler Realitätsformen unterscheiden [2]:

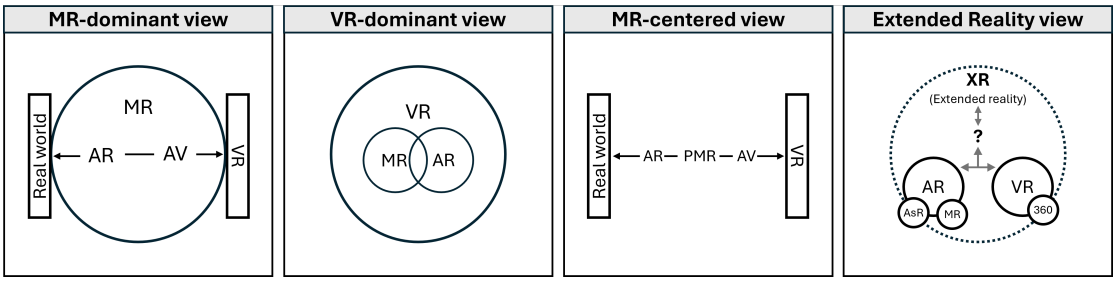


Abbildung 2.1.: Vier Ansichten digitaler Realitäten (eigene Darstellung nach [2], Fig.2.).

1. **MR-dominant view**: MR ist ein Oberbegriff und umfasst alle Erfahrungen, in denen reale und virtuelle Objekte kombiniert werden. Subkategorien sind Augmented Reality (AR, hier als virtuelles Overlay auf der realen Welt definiert) und Augmented Virtuality (AV, reale Objekte werden auf virtuelle Inhalte gelegt).

2. **VR-dominant view**: VR ist das Hauptmedium, in dem AR als Unterkategorie gesehen wird. MR dient dabei als Verbindung zwischen AR und VR.

3. **MR-centered view**: MR ist hier kein Oberbegriff, sondern ein spezieller Realitätstyp zwischen AR und AV. AR ist näher an der realen Welt, während AV sich der virtuellen Realität annähert.

4. **Extended Reality view**: XR (hier als Extended Reality definiert) dient als Sammelbegriff für AR und VR. MR wird als eine Art Mischform von AR und VR beschrieben, häufig ohne klare Definition. So wird es zum Beispiel als synonym für „realistisches AR“ oder als Umgebung, in der digitale und physische Objekte gemeinsam existieren und miteinander interagieren, beschrieben. Diese Ansicht wird hauptsächlich durch die Industrie geprägt. [2]

2.4. Das xReality Framework

Aufgrund der unterschiedlichen Klassifikationen schlagen Rauschnabel et al. in ihrem Artikel *„What is XR? Towards a Framework for Augmented and Virtual Reality“* eine aktualisierte Ansicht, das xReality Framework, vor. Dieses Framework dient der einheitlichen Definition und Klärung zentraler Begriffe und Konzepte. Es wurde mit Experten aus der Industrie, Fachleuten im Bereich AR und VR, sowie umfassender akademischer Literatur und industriellen Publikationen erstellt und lässt sich folgendermaßen veranschaulichen [2]:

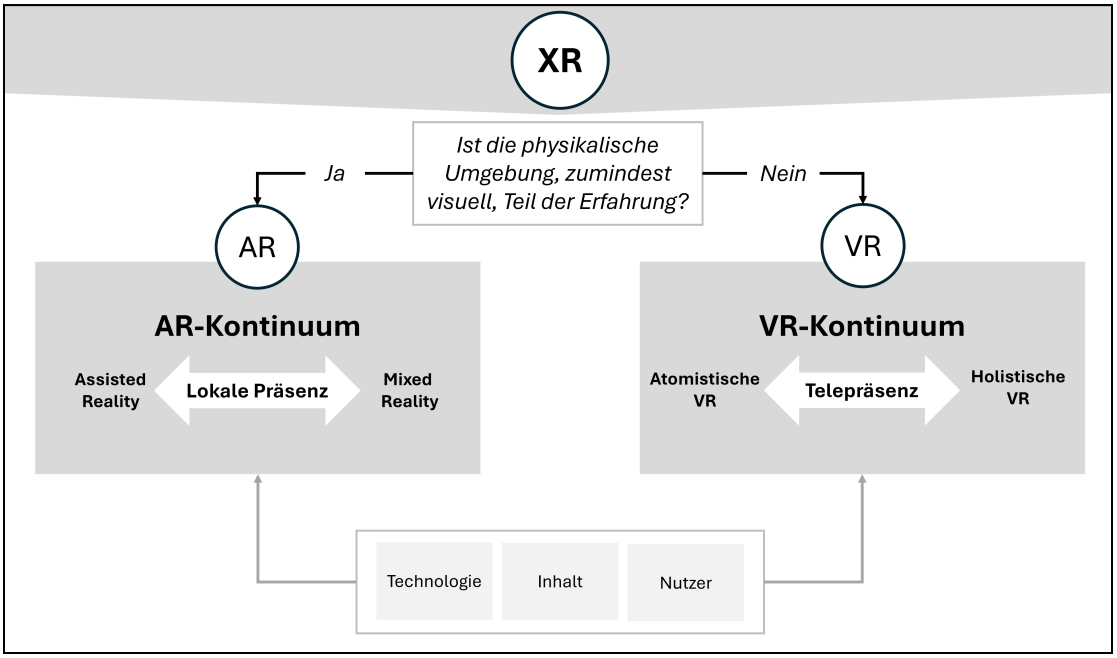


Abbildung 2.2.: Das xReality Framework (eigene Darstellung nach [2], Fig.4.).

Das Framework definiert das „X“ in XR als Platzhalter für alle Formen digitaler Realitäten und bezeichnet XR somit als xReality. Der Begriff „erweiterte Realität“ würde per Definition VR ausschließen, da es sich hierbei um eine vollständig ausgetauschte Realität handle. AR und VR werden strikt getrennt: Ist die physikalische Umgebung, zumindest visuell, Teil der Nutzungserfahrung, handelt es sich um AR - ist sie, zumindest visuell, vollständig ausgetauscht, um VR. Hardware oder Inhalte sind für diese Unterscheidung irrelevant, da moderne Geräte beide Technologien unterstützen. Deshalb ist es auch möglich, innerhalb eines Erlebnisses zwischen AR und VR zu wechseln, jedoch befindet sich ein Nutzer immer in einem der beiden Bereiche.

Das AR-Kontinuum unterteilt sich in „Assisted Reality“ und „Mixed Reality“, die sich durch den Grad der wahrgenommenen lokalen Präsenz unterscheiden. Assisted Reality überlagert Inhalte klar erkennbar und als künstlich wahrgenommen, während Mixed Reality diese realitätsnah integriert. Das VR-Kontinuum umfasst „Atomistische VR“ und „Holistische VR“. Atomistische VR fokussiert sich auf pragmatische Nutzung, während holistische VR durch starke Immersion und Telepräsenz charakterisiert ist. [2]

Das xReality Framework wird für die weitere Arbeit als grundlegende Definition für die digitalen Realitätsformen verwendet.

3. Stand der Technik/Forschung/Wissenschaft

Dieser Abschnitt ordnet das Thema der Arbeit anhand wissenschaftlicher Erkenntnisse ein. Etablierte sowie speziell für XR entwickelte Design-Guidelines, die zur Entwicklung der Applikation herangezogen wurden, werden erläutert. Zudem wird die Auswahl der Zielplattform für die Entwicklung und ihre Funktionsweise veranschaulicht.

3.1. Etablierte Usability-Heuristiken in Bezug auf XR

Bereits 1994 veröffentlichte Jakob Nielsen zehn allgemeine Grundsätze für das Interaktionsdesign [11]. Diese weit verbreiteten und bekannten [12] sogenannten „Heuristiken“ sind grob gefasste Regeln zur Beurteilung der Gebrauchstauglichkeit einer Benutzeroberfläche [13] [14]. Nielsen betonte im Jahr 2020, dass sich die Formulierungen der Definitionen zwar leicht angepasst hätten, die Heuristiken seit 1994 aber unverändert und weiterhin relevant geblieben seien. Er prognostizierte zudem, dass sie auch auf künftige Generationen von Benutzeroberflächen angewendet werden können. [13] Die anhaltende Bedeutung dieser Heuristiken, auch für Videospiele und XR-Anwendungen, wird in weiteren Quellen bestätigt [4] [15] [12].

Die zehn Heuristiken von Nielsen lauten wie folgt:

1. **Visbility of System Status**: *The design should always keep users informed about what is going on, through appropriate feedback within a reasonable amount of time.*
2. **Match Between the System and the Real World**: *The design should speak the users' language. Use words, phrases, and concepts familiar to the user, rather than internal jargon. Follow real-world conventions, making information appear in a natural and logical order.*
3. **User Control and Freedom**: *Users often perform actions by mistake. They need a clearly marked „emergency exit“ to leave the unwanted action without having to go through an extended process.*
4. **Consistency and Standards**: *Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform and industry conventions.*

5. **Error Prevention**: *Good error messages are important, but the best designs carefully prevent problems from occurring in the first place. Either eliminate error-prone conditions, or check for them and present users with a confirmation option before they commit to the action.*

6. **Recognition Rather than Recall**: *Minimize the user's memory load by making elements, actions, and options visible. The user should not have to remember information from one part of the interface to another. Information required to use the design (e.g. field labels or menu items) should be visible or easily retrievable when needed.*

7. **Flexibility and Efficiency of Use**: *Shortcuts — hidden from novice users — may speed up the interaction for the expert user so that the design can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.*

8. **Aesthetic and Minimalist Design**: *Interfaces should not contain information that is irrelevant or rarely needed. Every extra unit of information in an interface competes with the relevant units of information and diminishes their relative visibility.*

9. **Help Users Recognize, Diagnose, and Recover from Errors**: *Error messages should be expressed in plain language (no error codes), precisely indicate the problem, and constructively suggest a solution.*

10. **Help and Documentation**: *It's best if the system doesn't need any additional explanation. However, it may be necessary to provide documentation to help users understand how to complete their tasks.* [11] [13]

An diesen Heuristiken orientiert sich diese Arbeit bei der Konzeption und Entwicklung des Prototyps sowie der Nutzertest-Evaluation. An dieser Stelle werden einige, für das XR-Design besonders interessante, Heuristiken vertieft:

Die zweite Heuristik, die Übereinstimmung zwischen System und realer Welt, ist für XR und insbesondere für immersive VR-Erlebnisse von besonderer Bedeutung. Der Aufbau auf bestehenden mentalen Modellen erleichtert es den Nutzern, die Interaktionsmöglichkeiten in XR-Systemen vorherzusehen. Diese Modelle beeinflussen die Erwartungen der Nutzer an das System und basieren auf früheren realen sowie digitalen Erfahrungen. Dies ist besonders relevant, da viele Nutzer nur wenig oder gar keine Erfahrung mit VR haben und sich daher auf bereits bekannte Muster stützen. [15] [16] Auch Meta hebt in den Design-Guidelines für MR-Anwendungen hervor, dass es für die Gestaltung immersiver Erlebnisse wichtig ist, von der physischen Welt abzuleiten. Dies umfasst unter anderem intuitives Greifen, realistische Physik sowie natürliche Größen- und Tiefendarstellungen.

Interaktionen sollten so gestaltet werden, dass sie dem natürlichen Verhalten von Menschen mit Objekten und ihrer Umgebung entsprechen. Außerdem kann es in manchen Fällen sinnvoll sein, eine 2D-Benutzeroberfläche im virtuellen Raum zu platzieren, da Nutzer diese bereits von Smartphones, Tablets und Computern kennen. [17]

Die vierte Heuristik, die Wahrung von Konsistenz und Einhaltung von Standards, erfordert ebenfalls eine genauere Betrachtung. Die fehlende Vereinheitlichung von Interaktionsprinzipien bei XR-Systemen erschwert eine konsistente plattformübergreifende Bedienung. Bei etablierten Technologien wie Smartphones ist es beispielsweise branchenweit Standard, dass eine „Pinch-Geste“ zum Vergrößern von Inhalten dient. [3] Trotz der fehlenden Industrie-Standards (externe Konsistenz) sollte deshalb zumindest innerhalb einer Anwendung und ihrer Produktfamilie eine einheitliche Bedienung gewahrt werden (interne Konsistenz), da dies die Benutzerfreundlichkeit erhöht. [13] Daher orientiert sich diese Arbeit bei der Entwicklung des Prototyps an den Design-Leitlinien von Meta [17], um eine konsistente Nutzungserfahrung für erfahrene Meta-Quest-Nutzer sicherzustellen.

Die fünfte Heuristik, die Fehlervermeidung, spielt in der Hinsicht bei XR eine besondere Rolle, dass eine fehlerhafte Nutzung neben üblichen Problemen auch physische Verletzungen nach sich ziehen kann. Einige Anwendungen erfordern körperliche Bewegungen, wie Laufen, Greifen, Bücken und Umsehen. Sicherheitsfeatures, beispielsweise virtuelle Begrenzungen, die den Spielbereich kennzeichnen und Nutzer warnen, wenn sie diesen verlassen, sollten während der Nutzung stets aktiv sein. Nutzer sollten zudem darauf hingewiesen werden, sich in einem ausreichend großen Raum zu bewegen und während des Spielens Vorsicht walten zu lassen, um Unfälle zu vermeiden. [15] Nutzer dürfen durch die Anwendung nie zu Handlungen verleitet werden, die im Konflikt mit ihrer physischen Umgebung stehen [17].

Die siebte Heuristik, die Flexibilität und Effizienz der Nutzung, ist ebenfalls essenziell für eine komfortable XR-Erfahrung. Insbesondere die Personalisierung und Anpassung an individuelle Bedürfnisse [13] sollte möglich sein. Multimodale Interaktionsmöglichkeiten können dazu beitragen, dass Nutzer Methoden wählen können, die ihren Präferenzen entsprechen oder eventuelle körperliche Einschränkungen ausgleichen. [18] Bei der Nutzung von VR-Anwendungen kann außerdem körperliches Unwohlsein durch die sogenannte VR-Krankheit auftreten [19]. Es gibt verschiedene Maßnahmen, die diesem entgegenwirken, die aber auch die Immersion einschränken können. Nutzer sollten daher die Möglichkeit haben, die Anwendung bei Bedarf so einzustellen, dass die Symptome verringert werden. [20] [21] Genauer wird dies im Kapitel „VR-Krankheit“ erläutert.

3.2. Speziell für XR entwickelte UX-Guidelines

Einige Autoren betonen die Notwendigkeit, neue Guidelines für spezifische Anwendungen zu erstellen, da bestehende Richtlinien oft zu allgemein sind, um die Benutzerfreundlichkeit eines bestimmten Systems zu evaluieren [4]. Aufgrund neuer Interaktionsmöglichkeiten und der Art und Weise, wie digitale Informationen im dreidimensionalen Raum erlebt werden, sind speziell auf XR zugeschnittene UX-Design-Richtlinien erforderlich [4] [12].

Vi et al. entwickelten ein Set aus elf UX-Richtlinien für die Gestaltung von HMD-XR-Anwendungen. Hierfür wurden 68 verschiedene wissenschaftliche und industrielle Ressourcen analysiert, darunter die Arbeit von Endsley et al. [12], die sich bereits mit der Entwicklung von AR-Heuristiken befasst. Die Heuristiken von Vi et al. fundieren dennoch hauptsächlich auf VR-Erfahrungen, da es zum Zeitpunkt ihrer Erstellung nur eine limitierte Verfügbarkeit von XR-Geräten gab und die VR-Community und die damit verbundene Quellenlage größer waren. Der Fokus liegt außerdem auf Nutzungskomfort, der in vielen zugrundeliegenden Studien besprochen wurde und somit als große Herausforderung beim Design von XR-Anwendungen identifiziert wurde. [4]

Die Autoren sehen weiteren Forschungsbedarf, um die entwickelten Konzepte in der Praxis zu erproben und deren Einfluss auf die Gestaltung von XR-Anwendungen zu untersuchen. Aufgrund dessen sollen in dieser Arbeit neben den etablierten Design-Heuristiken von Nielsen auch die Richtlinien von Vi et al. zur Entwicklung und Evaluation des XR-Prototyps genutzt werden. [4]

Die elf UX-Guidelines für XR von Vi et al. lauten wie folgt:

1. **Organize the Spatial Environment to Maximize Efficiency**: *XR is inherently spatial. Use space as an organizational tool to create an environment that is comfortable to use and minimizes the amount of conscious thinking a user has to do to accomplish his or her goals.*

2. **Create Flexible Interactions and Environments**: *Provide users with the capability to customize the application to their personal preferences and comforts. Build in options that cater to a range of users that take into account different experience levels and physical considerations.*

3. **Prioritize User's Comfort**: *Provide users with the capability to customize the application to their personal preferences and comforts. Build in options that cater to a range of users that take into account different experience levels and physical considerations.*

4. **Keep it Simple: Do not Overwhelm the User**: *The more there is, the less the user remembers. Create simple and relevant elements in an environment that do not distract the user from what is important.*

5. **Design Around Hardware Capabilities and Limitations**: *The way users interact and explore the environment will be greatly dependent on the system they are using. Always keep the capabilities of the hardware in mind when crafting XR experiences.*

6. **Use Cues to Help Users Throughout Their Experience**: *Create signifying cues to help users to get started, provide additional information, guide user's attention, and simplify choice within the application.*

7. **Create a Compelling XR Experience**: *XR allows users to be immersed in the virtual environment. Enhance their senses through visuals, audio, and narrative elements that captivate them in the experience.*

8. **Build upon Real World Knowledge**: *Help users to understand how to use the application by designing the interactions, objects, and environments around existing knowledge of the real world.*

9. **Provide Feedback and Consistency**: *Use feedback to generalize perception of events and interactions. Additionally, feedback should be consistent such that users can build an understanding of what they can and cannot do within the application.*

10. **Allow Users to Feel in Control of the Experience**: *Users are vulnerable when they are immersed in XR environments. It is important for the application to establish a feeling of trust with the user by making sure that they feel in control during the whole experience.*

11. **Allow for Trial and Error**: *As much as possible, allow actions to be reversible and set up protections around potential mistakes made by users. This will help relieve user's anxiety and promote exploration of the application.* [4]

3.3. VR-Krankheit

Die sogenannte VR-Krankheit „[...] ist eine Form von Übelkeit, die mit dem Eintauchen in eine computergenerierte Umgebung auftritt." [19] Die VR-Krankheit ist ähnlich zu der Reise- oder Bewegungskrankheit (engl.: „Motion-Sickness") [22]. Diese wird in der Fachsprache auch als Kinetose bezeichnet und tritt auf, wenn bei Bewegungen die Informationen der Sinnesorgane des Menschen widersprüchlich sind oder diese nicht mit den Erwartungen des Nutzers übereinstimmen. In Konflikt stehen dabei Gleichgewichtssinn

(vestibuläre Wahrnehmung), Körperwahrnehmung (somatosensorische Wahrnehmung) und die visuelle Wahrnehmung. Viele Menschen leiden auf Schiffen oder beim passiven Transport in Fahrzeugen unter den Symptomen der Kinetose. [23]. Häufig auftretend sind Schwindelgefühl, Müdigkeit, Kopfschmerzen und Übelkeit bis hin zu Erbrechen [24].

Diese Symptome treten auch bei der VR-Krankheit auf, die sich jedoch insofern von der Bewegungskrankheit abgrenzt, dass Betroffene sich nicht tatsächlich in Bewegung befinden müssen [23]. In VR-Umgebungen wird eine Scheinbewegung (Vektion) wahrgenommen, wenn der Nutzer sich physisch nicht bewegt, jedoch Bewegungen durch visuelle Hinweise über das HMD erlebt. Die visuellen Informationen stehen dann im Konflikt zu denen des Gleichgewichtssinns und der Körperwahrnehmung, was Unwohlsein verursachen kann. [20] Technische Aspekte, wie unangepasste Bewegungen, ein großes Sichtfeld, Eingabeverzögerungen und niedrige Bildraten können ebenfalls zu Beschwerden beitragen [22]. Auch bei Video-See-Through (VST) AR-Systemen können Motion-Sickness-Symptome auftreten. Hierbei spielen zusätzlich geometrische Ungenauigkeiten und Verzerrungen in der Umgebungsanzeige eine Rolle. [25]

Um die Auswirkungen der VR-Krankheit zu reduzieren, können (optionale) Gegenmaßnahmen bereitgestellt werden. Zum einen sollte der Bewegungsstil anpassbar sein. Teleportationen statt sanftem Gleiten durch die Spielwelt können die VR-Krankheit reduzieren, allerdings auch zu Orientierungsverlust führen und die Immersion beeinträchtigen. Auch der Drehungsstil kann den Komfort beeinflussen. Physische Drehungen eines Nutzers entsprechen dem natürlichen Verhalten des Menschen, weshalb es hier nicht zu Konflikten in der Sinneswahrnehmung kommt. Bei Anwendungen, in denen die Drehung z.B. durch Thumb-Sticks des Controllers ermöglicht wird, gibt es zwei Optionen: ruckartige und sanfte Drehungen. Ruckartige Drehungen minimieren die VR-Krankheit. Hierbei dreht sich die Kamera immer in einem festen Winkel, was jedoch ebenfalls zu Orientierungsverlust führen kann. Bei sanften Drehungen ist die Drehgeschwindigkeit relativ zur Eingabeintensität, was zu einem stärkeren Missverhältnis zwischen visueller Bewegung und physischer Beschleunigung führt. Die Drehgeschwindigkeit sollte durch den Nutzer anpassbar sein. Auch Vignetten, also die Verdunkelung der Anzeigenränder, können helfen. Das Sichtfeld kann so bei Bewegungen eingeschränkt werden, damit sich der Nutzer stärker auf den mittleren Bereich konzentriert. [21]

3.4. Meta Quest 3 als Zielplattform für die Entwicklung

3.4.1. Auswahl der Plattform

Der im Rahmen dieser Arbeit erstellte Prototyp wird für die Meta Quest 3 (kurz: Quest 3) entwickelt. Die genutzte Entwicklungsumgebung erlaubt es theoretisch, die Anwen-

dung auch für andere Systeme der Quest-Reihe bereitzustellen. Einige Funktionen des Prototyps sind jedoch für die Quest 3 optimiert, ein Test auf anderen Plattformen wurde nicht durchgeführt. Das HMD wurde als primäre Plattform ausgewählt, da es im Gegensatz zum Vorgängermodell verbesserte Hardware, darunter auch verbesserte Video-See-Through AR-Features bietet. [26]. Zudem werden Controller und Gestensteuerung unterstützt, was ebenfalls essenziell für die technische Umsetzung dieser Arbeit ist [9] [27].

3.4.2. Funktionsweise der Meta Quest 3

Die Quest 3 ist ein HMD von Meta, das in Verbindung mit einem PC, mithilfe der integrierten Hardware jedoch auch autark verwendet werden kann [26].

Die beiden separaten LCD-Displays haben eine Auflösung von 2.064 x 2.208 Pixeln pro Auge mit einer Pixeldichte von 1.218 PPI und einer Aktualisierungsrate von bis zu 120 Hz [26] [27]. Die für die Optik verwendeten Pancake-Linsen ermöglichen eine höhere Abbildungsqualität und einen kompakteren Aufbau als bei den Vorgängermodellen [26]. Der Linsenabstand kann mithilfe eines Rads an der Außenseite der Brille an den Augenabstand der Nutzer angepasst werden. Sie bietet ein Sichtfeld von 110 Grad horizontal und 96 Grad vertikal. [27]

Für die Leistung der Brille sorgt ein Snapdragon XR2 Gen 2-Prozessor von Qualcomm, der extra für XR-Systeme entwickelt wurde [27] [28]. Vier Infrarot-Kameras erfassen die Umgebung und ermöglichen so ein 6 DoF Inside-Out-Tracking, womit das HMD ohne externe Basisstationen betrieben werden kann. Die Ansicht der Umgebung und damit verbundene AR- bzw. MR-Features werden durch zwei weitere RGB-Farbkameras mit Tiefenprojektion ermöglicht. [27] [29] Es handelt sich bei der Quest 3 also um ein geschlossenes Headset mit Video-See-Through-AR.

Die Quest 3 wird mit Meta Quest Touch Plus-Controllern geliefert. Diese bieten taktiles Feedback und ebenfalls 6 DoF zur Nachverfolgung im 3D-Raum. Die Nachverfolgung der Controller hat dabei eine Präzision von weniger als einem Millimeter und kann bei Verdeckung oder schlechtem Licht kurzzeitig mit Hilfe von maschinellem Lernen geschätzt werden [9]. Die Navigation wird durch Handtracking ergänzt, indem Sensoren in Verbindung mit maschinellem Lernen die Gesten der Nutzer erfassen. [27] [29]

Das Betriebssystem der Quest 3 ist das Meta Horizon OS. Dieses basiert auf Android und stellt eine 3D-UI bereit, die mit Gesten, Controllern oder Sprache bedient werden kann. Apps werden in 2D-Fenstern angezeigt, die im 3D-Raum platziert werden können. Außerdem wird durch das OS das Guardian-System bereitgestellt, mit Hilfe dessen man die Grenzen des Spielbereichs festlegen kann, um die physische Sicherheit der Nutzer zu gewährleisten. [30]

4. Game Design

In diesem Kapitel wird die Spielidee erläutert. Es folgt eine Analyse bestehender Spiele mit ähnlicher Funktionsweise und deren Einfluss auf das eigene Game Design. Des Weiteren werden die formalen Elemente des Designs (Spielertyp, Spieltyp, Ziele, Prozeduren, Regeln, Ressourcen, Konflikte, Grenzen) und die emotionalen Elemente (Herausforderung, Prämisse, Charaktere, Geschichte, Aufbau der Welt) dargestellt. Das im Rahmen dieser Arbeit entworfene Spiel trägt im Folgenden den Arbeitstitel „XR-Jump“.

4.1. Grundlegende Spielidee

Die Grundidee des Spiels war es, durch die Kombination besonderer Features ein Konzept zu entwickeln, das sowohl Spaß macht als auch hinsichtlich der Usability untersucht werden kann. Dabei stehen drei besondere Merkmale im Fokus:

Das Spiel kombiniert zwei Eingabemethoden - den klassischen Controller und die direkte Handinteraktion. Der Spieler hält in einer Hand einen Controller, während er mit der freien Hand aktiv in die Spielwelt eingreifen und mit ihr interagieren kann.

Die duale Eingabeinteraktion wird kombiniert mit einer Third-Person-Perspektive. Die Spielwelt baut sich um den Spieler herum auf, während dieser die Rolle einer beweglichen Kamera einnimmt. Aus dieser Perspektive steuert er den Charakter durch die Welt und kann gleichzeitig mit der Umgebung interagieren.

Das Spiel nutzt außerdem eine Kombination von VR und AR. Während einige Abschnitte in einer VR-Umgebung stattfinden, integrieren andere die reale Umgebung als Teil der Spielwelt. Dadurch entsteht ein XR-Erlebnis mit fließenden Übergängen zwischen AR und VR.

Um diese Mechaniken sinnvoll zu kombinieren, bietet sich ein 3D-Plattformer (Jump 'n' Run) als Genre an. Die Level bauen sich dabei um den Spieler auf, während der Charakter per Controller gesteuert wird. Durch Eingriffe mit der freien Hand kann der Spieler Hindernisse überwinden und das Ziel erreichen. Zudem ermöglicht das Konzept, einzelne Abschnitte gezielt in AR oder VR anzusiedeln.

4.1.1. Prüfung der technischen Machbarkeit

Bevor das Spielkonzept weiter ausgearbeitet wurde, sollte zunächst getestet werden, ob die Umsetzung im Rahmen einer Bachelorarbeit technisch realisierbar ist. Dazu wurde mit der Unity-Engine ein erster Software-Prototyp entwickelt. Da dieser lediglich grundlegende Funktionen enthielt, die in der späteren Entwicklung aufgegriffen und verbessert werden, wird seine Entstehung an dieser Stelle nicht weiter behandelt.

Der Prototyp implementiert bereits die duale Steuerung in einer simplen Testumgebung, die aus mehreren Plattformen besteht. Der Charakter kann mit dem Controller gesteuert werden, während der Spieler mit der freien Hand bewegliche Blöcke verschieben und einfache Brücken bauen kann. Eine AR-Integration wurde in dieser frühen Phase noch nicht umgesetzt, sodass es sich um ein reines VR-Erlebnis handelt.

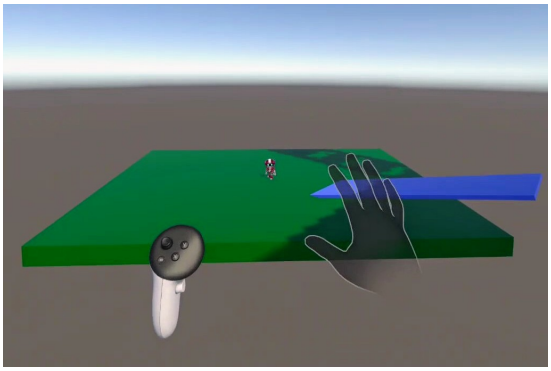


Abbildung 4.1.: Erster Prototyp - Controller und Handinteraktion (Screenshot aus der Unity-Engine).

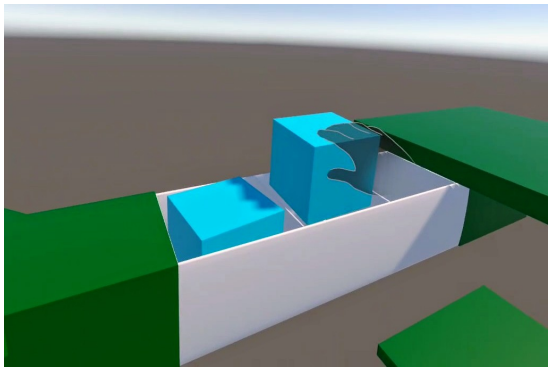


Abbildung 4.2.: Erster Prototyp - Eingreifen in die Spielwelt (Screenshot aus der Unity-Engine).

Der Prototyp bestätigt die grundsätzliche Umsetzbarkeit der Spielidee und ermöglichte erste informelle „Hallway-Tests“ [31] mit drei Testpersonen. Alle Teilnehmenden konnten das Spielprinzip intuitiv erfassen und das erste Test-Level erfolgreich abschließen. Zudem gaben sie an, sich vorstellen zu können, dass das Konzept die Grundlage für ein unterhaltsames Spiel bildet.

4.2. Inspirationsquellen und ähnliche Spiele

Um den Entwurf des Spiels zu fundieren, wurden ähnliche Spiele analysiert, um sich von ihnen inspirieren zu lassen und deren Ansätze im Game Design mit dem eigenen Spielkonzept zu verknüpfen. Dabei wurden nicht nur XR-Spiele untersucht, sondern auch konventionelle Plattformer und deren spezifische Herangehensweise an das Genre.

4.2.1. Level-Design in Super Mario 3D-Spielen

Die „Super Mario“-Reihe von Nintendo umfasst sowohl 2D- als auch 3D-Plattformer und prägt das Genre seit der Veröffentlichung von „Super Mario Bros.“ im Jahr 1985 maßgeblich [32].

Insbesondere in den neueren 3D-Ablegern der Reihe, wie „Super Mario 3D World“, fällt auf, dass diese eine Vielzahl an Features und Spielmechaniken bieten, ohne dabei überladen zu wirken oder ausgedehnte Tutorials zu benötigen. Dies ist vor allem auf Kōichi Hayashida zurückzuführen, der als Level-Design-Director, Co-Direktor oder Director mehrerer 3D-Mario-Titel eine vierstufige Methode für das Level-Design entwickelte [33].

Diese Methode wurde von „Kishōtenketsu“ inspiriert, einer rhetorischen und narrativen Struktur aus japanischen Werken, die aus vier Abschnitten besteht:

1. *Ki*, die Einleitung zur Vorstellung des Themas,
2. *Shō*, die Entwicklung zur Weiterführung und Vertiefung des Themas,
3. *Ten*, die Wende als unerwartetes Element, das nur indirekt einen Bezug oder eine Verbindung mit dem Thema hat,
4. *Ketsu*, der Schluss der alle Elemente zusammenführt und eine Schlussfolgerung zieht. [34] [33]

Analog dazu erfolgt die Implementierung der Spielmechaniken:

1. Ein neues Konzept wird zu Beginn eines Levels eingeführt. Die Mechanik ist intuitiv erkennbar und vorhersehbar, sodass der Spieler sie ohne ein Tutorial erfassen kann. Zudem hat er die Möglichkeit, die Mechanik in einer sicheren Umgebung zu erproben (Siehe Abbildung 4.3. (a)).
2. Mit fortschreitendem Level wird die Mechanik weiterentwickelt, indem sie schwieriger wird oder mit anderen Mechaniken kombiniert wird (Siehe Abbildung 4.3. (b)).
3. Die dritte Stufe beinhaltet eine Wendung oder einen Twist. Dies kann eine besonders anspruchsvolle Herausforderung sein oder erfordert vom Spieler, die Mechanik auf eine unkonventionelle Weise zu nutzen (Siehe Abbildung 4.3. (c)).
4. Zum Abschluss eines Levels wird die Mechanik in einer finalen Herausforderung, wie einem Bosskampf oder der ikonischen Fahnenmast-Sequenz, erneut aufgegriffen. Hierbei kann der Spieler sein erlerntes Wissen unter Beweis stellen (Siehe Abbildung 4.3. (d)). [33]

(a) Einleitung der Mechanik: Alle ?-Platten müssen aktiviert werden, um weiterzukommen.

(b) Weiterentwicklung: Der Spieler muss zusätzlich durch ein Labyrinth navigieren.

(c) Twist: Ein besonders starker Gegner und bewegliche Plattformen kommen hinzu.

(d) Abschluss: Alle Plattformen können aktiviert werden, um das Level perfekt abzuschließen

Abbildung 4.3.: Entwicklung eines Level-Designs in *Super Mario 3D World + Bowser's Fury* (eigene Screenshots von der Nintendo Switch, © Nintendo [35]).

Dieses Vorgehen beim Level-Design schafft praktische, wiederverwendbare Strukturen, die im weiteren Spielverlauf erneut genutzt werden können. Zusätzlich enthält jedes Level optionale sammelbare Gegenstände, die weitere Twists und zusätzliche Herausforderungen bieten. Dies führt zu einem dynamischen Schwierigkeitsgrad, bei dem der Spieler selbst entscheiden kann, welche Herausforderungen er auf dem Weg zum Ziel meistern möchte. [33] [36]

Einige „Super Mario"-Spiele implementieren zudem eine zentrale Kernmechanik, die sich durch das gesamte Spiel zieht. Diese beeinflusst alle weiteren Spielelemente und wird mit anderen Mechaniken kombiniert, wodurch ein zusammenhängendes Spielerlebnis entsteht. In „Super Mario Odyssey" kann Mario beispielsweise seine Mütze werfen, um Gegner oder Objekte zu übernehmen. Dies unterstützt ihn je nach Situation beim Klettern, Kämpfen oder Lösen von Rätseln. [37]

4.2.2. Moss: Third-Person VR Adventure mit Plattformer-Elementen und dualer Eingabeinteraktion

Moss ist ein Action-Adventure von Polyarc, das 2018 für PlayStation VR erschien und später für weitere Plattformen, darunter die Oculus Quest, portiert wurde. Der Spieler

bewegt dabei die Hauptfigur mithilfe des Controllers durch die kleine Spielwelt, die sich in einer Art Diorama vor dem Spieler aufbaut und an vielen Stellen mit Jump 'n' Run-Elementen versehen ist. Zudem werden die Bewegungen des Controllers im Raum erkannt, um die Position der Hände nachzuverfolgen. Diese werden in Form einer blau leuchtenden Energiekugel dargestellt, mit deren Hilfe der Spieler in die Welt eingreifen kann, um beispielsweise Türen zu öffnen oder Blöcke zu bewegen. [38]

Corinne Scrivens, Principal Artist bei Polyarc, erklärt in einem Talk auf der Game Developers Conference (GDC), wie Design-Entscheidungen für Moss getroffen wurden. Dabei geht sie insbesondere auf die Herausforderungen der Story- und Weltgestaltung von Spielen dieser Art ein und beschreibt, wie das Entwicklerteam diese gelöst hat:

Zum einen war es wichtig, eine Welt zu schaffen, die für den Spieler nachvollziehbar ist und mit der er sich identifizieren kann. Es musste also erklärt werden, warum die Spielwelt im Vergleich zum Spieler so klein beziehungsweise der Spieler in der Welt besonders groß erscheint. In Moss ist die Hauptfigur die Mäusekriegerin Quill, und die Spielwelt stellt die Heimat der Mäuse dar. Dies ermöglicht eine gedankliche Brücke zu einer kleinen Spielwelt, die dadurch authentisch wirkt.

Um zur Glaubwürdigkeit der Spielwelt beizutragen und die Motivation des Spielers zu erhöhen, Quill auf ihrem Abenteuer zu unterstützen, sollten zudem bedeutungsvolle Beziehungen zwischen dem Spieler und dem Hauptcharakter geschaffen werden. Dafür war es essenziell, dass der Spieler nicht nur als Kamera mit Interaktionsmöglichkeiten fungiert, sondern auch eine eigene Figur innerhalb der Spielwelt verkörpert. Die Geschichte von Moss erklärt daher, dass der Spieler ein nicht physischer, geisterhafter Begleiter der Hauptfigur ist. Dies fördert eine Spieler-Held-Beziehung, in der die Heldin Quill zwar eigenständig agiert, jedoch weiterhin auf die Zusammenarbeit mit dem Spieler angewiesen ist, um das Ziel zu erreichen. Zudem wird so erklärt, warum Quill ständig vom Spieler begleitet wird, was die Immersion des Spiels verstärkt.

Darüber hinaus wurde in dem GDC-Talk erläutert, wie das Medium VR die Art der Geschichtenerzählung beeinflusst. Da die Kamera im Spiel der Kopf des Spielers ist, können traditionelle Zwischensequenzen, wie sie in Nicht-VR-Spielen häufig eingesetzt werden, problematisch sein. Ein Schnitt würde den Spieler quasi teleportieren und dadurch zu Orientierungslosigkeit führen, während Kamerabewegungen unter Umständen Übelkeit hervorrufen können. Deshalb ließ sich das Entwicklerteam vom Storytelling auf Theaterbühnen inspirieren. Die Aufmerksamkeit des Spielers wird gezielt durch Licht, Bewegung, Farbe und räumliches Audio gelenkt. Zudem kann eine Blickerfassung implementiert werden, sodass bestimmte Aktionen erst ausgelöst werden, wenn der Spieler tatsächlich an die vorgesehene Stelle blickt. [39]

4.2.3. Einfluss auf das eigene Game Design

In Anlehnung an das Level-Design der Super Mario 3D-Spiele sollen bei XR-Jump Spielmechaniken in einer ähnlichen Weise implementiert werden. Die Kernmechanik des Spiels besteht in der Handinteraktion mit der Umgebung. Für die im Rahmen des Prototyps erstellten Level sollen Mechaniken entwickelt werden, die nur in Kombination mit der Handinteraktion funktionieren und mithilfe des vierstufigen Systems von Kōichi Hayashida in das Gameplay integriert werden. Dies ermöglicht es, trotz begrenzter Produktionskapazitäten mehrere Spielmechaniken zu implementieren, die für Abwechslung sorgen. Darüber hinaus können verschiedene Interaktionsmöglichkeiten im Hinblick auf die im dritten Kapitel vorgestellten Design-Heuristiken erprobt werden.

Zudem sollen Elemente des Game Designs von Moss in das eigene Spiel übernommen werden. Die Nachvollziehbarkeit der kleinen Spielwelt soll durch Story-Elemente unterstützt werden, die gleichzeitig die Rolle des Spielers verdeutlichen. Dieser soll nicht nur als interaktive Kamera agieren, sondern eine eigene Figur innerhalb der Spielwelt darstellen, um die Immersion zu erhöhen. Auch die Art und Weise, wie in Moss die Aufmerksamkeit des Spielers auf wichtige Ereignisse gelenkt wird, kann für das eigene Spiel adaptiert werden.

Eine Diorama-Ansicht der Spielwelt, wie bei Moss, steht mit dem eigenen Spielkonzept jedoch im Konflikt, da diese aufgrund der Position der Kamera bzw. des Spielers eine direkte Handinteraktion erschwert. Deshalb soll der Spieler dem Charakter folgen, was Ähnlichkeiten zur Kamerasteuerung in den Super Mario 3D-Spielen aufweist, sich aber auch schon bei VR-Plattformern wie „Lucky's Tale“ [40] oder „Ven VR Adventure“ [41] etabliert hat. Anpassbare Kameramodi sollen jedoch dazu beitragen, das Spielerlebnis zu individualisieren und der VR-Krankheit vorzubeugen.

4.3. Zielgruppe

Der entwickelte Prototyp richtet sich primär an Casual- bis Core-Gamer ab einem Alter von etwa zwölf Jahren, die über eine grundlegende Vertrautheit mit Gamepad-Steuerungen verfügen. Diese Vorerfahrung ist insbesondere im Hinblick auf die teilweise komplexe, duale Eingabeinteraktion - bestehend aus Controller- und Gestensteuerung - von Vorteil.

Um das Spiel dennoch niederschwellig zu gestalten und ein möglichst breites Publikum anzusprechen, wurde bewusst auf eine intuitive Ausgestaltung der Spielmechaniken geachtet. Darüber hinaus sollen narrative Elemente und eine stimmungsvolle Spielwelt auch Spieler ohne tiefere technische Vorkenntnisse ansprechen und motivieren, sich mit den Mechaniken auseinanderzusetzen.

Das Spiel richtet sich somit sowohl an neugierige Einsteiger im XR-Bereich als auch an erfahrene Spieler.

4.4. Formale Elemente

Nach Tracy Fullertons „Game Design Workshop - A Playcentric Approach to Creating Innovative Games“ bilden die formalen Elemente die strukturelle Grundlage eines Spiels. Diese lassen sich in die Kategorien Spieler, Ziele, Prozeduren, Regeln, Ressourcen, Konflikte, Grenzen und das Ergebnis unterteilen und sollen im Folgenden dargestellt werden.[42, S. 60]

4.4.1. Spieler

Das entwickelte Spiel unterstützt ausschließlich einen einzelnen Spieler und folgt damit dem Spieler-Interaktionsmuster „Einzelspieler versus Spiel“. In Abwesenheit weiterer Spieler müssen daher Strukturen geschaffen werden, die Konflikte erzeugen, welche vom Spieler gelöst werden müssen. [42, S. 63].

Trotz des Singleplayer-Formats übernimmt der Spieler zwei unterschiedliche Rollen: Zum einen steuert er einen Charakter aus der Third-Person-Perspektive, mit dem klassische Plattformer-Aufgaben (z.B. Springen, Laufen) erfüllt werden. Zum anderen agiert er als „helfende Hand“, die aus der Egoperspektive in die Spielwelt eingreifen kann.

Diese duale Steuerung ist aus Usability-Sicht besonders interessant, da sie hinsichtlich Komfort und Komplexität eine Herausforderung darstellen kann. Daher wird bei der Implementierung der Spielmechaniken besonderer Wert darauf gelegt, den Spieler weder zu überfordern noch unter Druck zu setzen.

Zunächst ist es jedoch essenziell, die Motivation zum Spielen aufrecht zu erhalten. Das „Gamer Motivation Model“ von Quantic Foundry, basierend auf der Analyse von über 500.000 Spielern, identifiziert sechs Hauptmotivationen: Action, Social, Mastery, Achievement, Immersion und Creativity. [43][42, S. 62]

XR-Jump soll die Spieler auf mehreren Ebenen ansprechen: Immersive Story-Elemente und das XR-Medium fördern das Eintauchen in die Spielwelt. Kreativitätsorientierte Spieler werden durch die Erkundung der verschiedenen Interaktionsmöglichkeiten angesprochen. Die klassischen Plattformer-Elemente bieten Anreiz für Spieler mit einem Fokus auf „Mastery“, während leistungsorientierte Spieler (Fokus auf „Achievement“) durch das Sammeln von Collectibles zusätzlich motiviert werden können.

4.4.2. Ziele

Die Ziele des Spiels definieren, was der Spieler im Rahmen der Spielregeln zu erreichen versucht. Bestenfalls sind diese herausfordernd, jedoch stets erreichbar.[42, S. 72]

Das übergeordnete, explizite Ziel von XR-Jump ist das Freispielen und Abschließen der einzelnen Level. Um dieses langfristige Ziel zu erreichen, müssen fortlaufend kurzfristige Teilziele bewältigt werden. Dazu zählen das Überwinden von Hindernissen sowie das Erforschen und Nutzen der einzelnen Interaktionselemente.

Das Spiel ermöglicht es dem Nutzer zusätzlich, selbst gewählte, optionale Ziele zu verfolgen. Hierzu gehören das Erzielen eines neuen Highscores, das Unterbieten der eigenen Bestzeit oder das Sammeln aller Collectibles innerhalb eines Levels.

Darüber hinaus vermittelt die narrative Ebene ein implizites, übergeordnetes Ziel. Dieses dient - unabhängig vom formalen Spielsystem - der zusätzlichen Motivation zum Durchspielen des Spiels.

4.4.3. Prozeduren

Die Prozeduren bezeichnen die Spielmethoden und Aktionen, die der Spieler durchführen kann, um die Ziele zu erreichen [42, S. 78]. Der Vollständigkeit halber werden in diesem Abschnitt auch Prozeduren aufgeführt, die im Laufe der Entwicklung entstanden sind, jedoch ursprünglich nicht im Game Design erarbeitet wurden.

4.4.3.1. Starten des Spiels und UI-Aktionen

Das Spiel beginnt im Hauptmenü, das aus einer im Raum schwebenden 2D-Benutzeroberfläche besteht und verschiedene Buttons bereitstellt. Diese UI-Elemente werden ausschließlich durch Handinteraktionen aktiviert, indem der Spieler sie mit dem Zeigefinger berührt. Auf diese Weise soll der Spieler bereits im Hauptmenü das mentale Modell verinnerlichen, dass alle direkten Interaktionen mit der Umgebung über die Hand erfolgen.

Meta empfiehlt in seinem Designleitfaden die Verwendung von 2D-Benutzeroberflächen, da Nutzer diese bereits von anderen Geräten kennen [17].

Über den „New Game“-Button startet ein neues Spiel, während über „Continue“ ein bestehender Spielstand geladen werden kann. Im „Settings“-Menü lassen sich verschiedene Kameramodi auswählen.

Während des gesamten Spiels kann der Spieler durch Gedrückthalten des sekundären Buttons am Controller („Y“ links oder „B“ rechts) ein Menü aufrufen. Je nach aktuellem

Kontext kann er über UI-Buttons das Level neu starten, zur Levelauswahl oder zum Hauptmenü zurückkehren.

In einem laufenden Level wird zusätzlich ein Canvas eingeblendet, das Informationen zum Spielzustand anzeigt (z.B. verbleibende Leben, aktueller Score, verstrichene Zeit, gesammelte Gegenstände).

Der Spieler kann jederzeit entscheiden, welche Hand für die Controller-Steuerung und welche für die Handinteraktion verwendet werden soll. Dazu muss er lediglich den jeweiligen Controller in die gewünschte Hand nehmen.

4.4.3.2. Zentrieren des XR-Rigs

An bestimmten Punkten, vor allem zu Spielbeginn und bei Szenenwechseln, muss die Position des XR-Kamera-Rigs zentriert werden. Diese Aktion wird durch das Betriebssystem der Quest 3 bereitgestellt und erfolgt immer mit der rechten Hand.

Wenn der rechte Controller genutzt wird, erfolgt die Zentrierung durch Gedrückthalten des „Meta-Buttons“. Wird der linke Controller verwendet, muss der Spieler eine bestimmte Geste mit der rechten Hand ausführen, um die Zentrierung auszulösen.

4.4.3.3. Bewegung des Charakters

Die Charaktersteuerung erfolgt über den Controller:

- **Laufen**: über den Thumbstick. Die Eingabestärke beeinflusst direkt die Laufgeschwindigkeit.
- **Springen**: über den Trigger-Button. Wird dieser gehalten, springt der Charakter höher.
- **Dash**: durch die Greif-Taste, die den Charakter kurz in Bewegungsrichtung beschleunigt. In Kombination mit dem Sprung ergibt sich ein „Air-Dash“, um größere Abstände zu überwinden.

Diese Buttonbelegung wurde gewählt, da Trigger und Greif-Taste die einzigen Tasten sind, die bei gleichzeitiger Nutzung des Thumbsticks bequem erreichbar sind.

Der Charakter sammelt Objekte automatisch beim Durchlaufen ein. Checkpoints werden durch Überqueren aktiviert und dienen als Respawn-Punkte.

4.4.3.4. Bewegung des Spielers

Die Bewegung des Spielers erfolgt durch die physische Bewegung im Raum sowie durch Kopf- und Körperrotation, was durch das Headset-Tracking erfasst wird.

Zusätzlich bewegt sich der Spieler automatisch in einem festen Abstand zum Charakter mit. Zur Reduktion von Vektionen (wahrgenommenen Scheinbewegungen) kann der Spieler zwischen verschiedenen Kameraoptionen wählen und so das Spielerlebnis an die eigenen Präferenzen anpassen [20]:

Der „Immersive-Modus“ lässt den Spieler in einer flüssigen Bewegung dem Charakter folgen. Optional kann hier eine Vignette aktiviert werden, um das Sichtfeld bei Fremdbewegung des Spielers zu verkleinern. Statt der Vignette ist es auch möglich, das „Dash-Movement“ zu aktivieren, bei dem man dem Charakter in ruckartigen Bewegungen folgt, sobald er sich eine gewisse Distanz vom Spieler entfernt hat. Im Gegensatz zum Teleport bietet der Dash die Möglichkeit, durch den sanfteren Übergang der Orientierungslosigkeit vorzubeugen. [21]

Der „Komfort-Modus“ teleportiert den Spieler an vorgefertigte Positionen im Level, sobald der Charakter gewisse Bereiche betritt. Dieser Modus beinhaltet keine direkte Fremdbewegung durch die Bewegung des Charakters. Dies reduziert Bewegungsübelkeit, kann aber zu Orientierungslosigkeit führen. [21]

Um das Erlebnis noch zugänglicher zu machen, kann das Spiel weitgehend sowohl im Stehen als auch im Sitzen gespielt werden.

4.4.3.5. AR-VR Übergang

Nach dem Intro wechselt der Spieler durch ein Portal von der realen Welt (via Video-See-Through AR) in die virtuelle Welt. Am Spielende erfolgt die Rückkehr auf demselben Weg.

Diese Übergänge sind die einzigen Punkte im Spiel, an denen der Spieler sich physisch im Raum bewegen muss. Der Übergang erfolgt durch tatsächliches Durchschreiten des Portals.

4.4.3.6. Interaktions-Taste

Wenn sich der Charakter in der Nähe eines Interaktionspunkts befindet, wird der Spieler über ein UI-Element zum Drücken des primären Buttons („X“ links oder „A“ rechts) aufgefordert. Diese Taste wird verwendet für:

- Start und Freischalten eines Levels,
- Rückkehr zur Levelauswahl am Levelanfang/-ende ohne Menü.

4.4.3.7. Handinteraktion: Direktes Greifen

Ein zentrales Spielelement ist das direkte Greifen von Objekten via Hand-Tracking. Zum Greifen müssen mindestens zwei Finger den Rand eines Objekts berühren oder das Objekt in der Handfläche eingeschlossen werden. Dies ermöglicht ein realistisches Greifgefühl, bei dem man frei entscheiden kann, wie ein Objekt gegriffen werden soll. Zum Loslassen genügt das Öffnen der Hand bzw. das Entfernen der Finger. Manche Objekte unterliegen der Gravitation, andere schweben frei im Raum.

Im Spiel enthalten sind folgende direkte Greif-Aktionen:

1. Würfel platzieren, um Türen zu öffnen.
2. Plattformen hochklappen, um Sprungelemente für den Charakter zu schaffen.
3. Platzieren schwebender Plattformen um Brücken für den Charakter zu bauen.
4. Stapeln von Objekten, um Kletterhilfen zu schaffen.
5. Würfel in Halterungen setzen, um Überquerungen zu ermöglichen.

4.4.3.8. Handinteraktion: Indirektes Greifen

Um auch entfernte Objekte manipulieren zu können, gibt es das indirekte Greifen: Der Spieler zielt mit ausgestreckter Hand auf das Objekt und presst Daumen und Zeigefinger zusammen, um es zu greifen. Das Objekt wird losgelassen, wenn die Finger wieder getrennt werden. Dies ist von Vorteil, um den interaktiven Bereich zu erweitern.

Im Spiel enthalten sind folgende indirekte Greif-Aktionen:

1. Bewegung von Objekten auf der X-Achse, um einen begehbaren Pfad für den Charakter zu schaffen.
2. Bewegung von Objekten auf der Y-Achse, also Hebeelemente, um Sprünge über Abgründe zu ermöglichen.

4.4.3.9. Handinteraktion: Hovern

Bestimmte Spielaktionen können allein durch das Platzieren der Hand an einer bestimmten Stelle ausgelöst werden.

Im Spiel gibt es zwei Hover-Aktionen:

1. Öffnen des XR-Portals im Intro durch Hovern in einer leuchtenden Kugel.
2. Aktivieren eines Kraftfelds, das den Charakter nach oben katapultiert, wenn die Hand über das Feld gehalten wird.

4.4.3.10. Handinteraktion: Hand-Posen

Einige Objekte reagieren auf spezifische Hand-Posen, also bestimmte Gesten. Diese Interaktionsform ist besonders interessant, da sie sich gut mit der gleichzeitigen Charaktersteuerung kombinieren lässt, was Multitasking erfordert.

Folgende Aktionen können im Spiel durch Posen ausgelöst werden:

1. Bewegliche Plattformen aktivieren mit ausgestreckter, nach vorne gerichteter Hand; stoppen bei nach oben gerichteter Hand.
2. „Daumen hoch“ / „Daumen runter“ verändern den Zustand klappbarer Plattformen (oben/unten). Mehrere dieser Plattformen hintereinander müssen so durch abwechselnde Gesten ihren Zustand ändern, damit der Charakter z.B. einen Abgrund bewältigen kann.

Die Entwicklung der Handinteraktionen stellt im Hinblick auf die Usability eine besondere Herausforderung dar. Dabei müssen mehrere Aspekte berücksichtigt und anschließend getestet werden: Wie präzise funktioniert die Interaktion mit der Hand? Interaktionspunkte sollten eindeutig erkennbar sein und durch Feedbackmechanismen - wie visuelle oder auditive Signale - unterstützt werden. Zudem ist entscheidend, ob ein Lerneffekt beim Spieler eintritt und wie gut sich die einzelnen Mechaniken intuitiv vermitteln lassen.

4.4.4. Regeln

Regeln definieren Spielobjekte und erlaubte Aktionen des Spielers. Sie können explizit erklärt oder implizit im Spiel enthalten sein, etwa indem unerlaubte Handlungen technisch nicht umsetzbar sind oder aktiv verhindert werden. Regeln dienen unter anderem dazu, Schlupflöcher im Spielsystem zu schließen und zu vermeiden, dass das Spiel in einen Zustand gerät, in dem kein Fortschritt mehr möglich ist [42, S. 78 ff.].

4.4.4.1. Regeln zum Spielablauf

Der Ablauf des Spiels ist durch eine lineare Struktur geregelt:

- Um das erste Level spielen zu können, muss zunächst das Intro abgeschlossen werden.
- Weitere Level werden jeweils erst freigeschaltet, wenn das vorherige erfolgreich absolviert wurde.
- Das Spiel gilt als abgeschlossen, wenn alle Level durchgespielt wurden.

Um ein Level abzuschließen, muss der sogenannte „Fokus-Stein“ am Ende des Levels erreicht und eingesammelt werden.

Zu Beginn hat der Charakter fünf Leben. Fällt er in den Abgrund, verliert er ein Leben und wird zum letzten aktivierten Checkpoint zurückgesetzt. Diese Checkpoints sind im Level verteilt. Sind alle Leben aufgebraucht oder kehrt der Spieler über das Menü zur Levelauswahl bzw. ins Hauptmenü zurück, muss das Level von vorne begonnen werden. In diesem Fall geht der bisherige Fortschritt im aktuellen Level verloren. Sammelt der Charakter 50 Kristalle, gewinnt er ein zusätzliches Leben hinzu. Kristalle und verbliebene Leben werden beim Übergang in das nächste Level beibehalten. Diese Regelungen werden im Spiel nicht explizit erklärt, sondern sollen sich dem Spieler durch das Gameplay erschließen.

4.4.4.2. Regeln zur Bewegung des Charakters

- Die Bewegungsgeschwindigkeit des Charakters ist nach oben begrenzt. Langsameres Gehen ist durch geringere Eingabestärke möglich.
- Ein Sprung ist nur dann ausführbar, wenn sich der Charakter am Boden befindet.
- Die Sprungkraft ist skalierbar: Durch Gedrückthalten der Sprungtaste kann ein stärkerer Sprung ausgelöst werden - bis zu einem festgelegten Maximalwert.
- Der Dash hat eine festgelegte Reichweite und einen Cooldown, um kontinuierliches Dashen zu verhindern.
- Ein Dash ist sowohl am Boden als auch in der Luft möglich.

4.4.4.3. Regeln zur Bewegung des Spielers

Der Spieler kann sich jederzeit durch physische Bewegung im realen Raum umsehen und seine Position verändern. Innerhalb eines Levels folgt die Spielerperspektive zusätzlich automatisch dem Charakter in einem bestimmten Abstand. Darüber hinaus kann der Spieler die Position des XR-Rigs jederzeit zurücksetzen, um zur vorgesehenen Grundposition im Raum zurückzukehren.

4.4.4.4. Regeln zur Handinteraktion

- Der Charakter selbst kann durch Handinteraktionen weder bewegt noch gegriffen oder auf andere Weise beeinflusst werden.
- Nur bestimmte, besonders gekennzeichnete, Objekte können durch Handinteraktionen manipuliert werden.
- Jedes Objekt unterstützt ausschließlich eine der drei Interaktionsarten: direktes Greifen, indirektes Greifen oder Hand-Posen.
- Die Beweglichkeit von Objekten ist auf bestimmte Spielbereiche beschränkt, um ihren Einsatz außerhalb des vorgesehenen Kontextes zu verhindern.
- Greifbare Objekte, die am Boden liegen, unterliegen der Gravitation, auch nach dem Loslassen.
- In der Luft schwebende, greifbare Objekte behalten ihre Position bei, wenn sie losgelassen werden.
- Werden bewegliche Objekte außerhalb des für den Spieler zugänglichen Bereichs platziert (z.B. in einen Abgrund geworfen), kehren sie automatisch an ihre Ursprungsposition zurück.
- Objekte, die durch Hand-Posen gesteuert werden, reagieren nur dann auf Gesten, wenn sich der Charakter in unmittelbarer Nähe befindet.

4.4.5. Ressourcen

Ressourcen sind Elemente, die zum Erreichen der Ziele erforderlich sind, dem Spieler jedoch nur begrenzt zur Verfügung gestellt werden, um eine zusätzliche Herausforderung zu schaffen [42, S. 84 ff.].

Die wichtigste Ressource in XR-Jump sind die Leben des Charakters. Zu Beginn stehen dem Spieler fünf Leben zur Verfügung. Verliert der Charakter sämtliche Leben durch

wiederholtes Herabfallen in den Abgrund, wird das Level zurückgesetzt und der Spieler startet erneut vom Anfang.

Um neue Leben zu erhalten, dient eine zweite Ressource: Kristalle. Diese sind in allen Leveln verteilt und werden automatisch eingesammelt, sobald sich der Spieler ihnen nähert. Für jeweils 50 gesammelte Kristalle erhält der Spieler ein zusätzliches Leben. Die Kristalle dienen außerdem der Orientierung innerhalb der Level, indem sie so angeordnet werden, dass sie den Weg weisen.

Eine weitere Ressource sind Collectibles in Form von Münzen. In jedem Level sind drei dieser Münzen versteckt oder nur über besonders herausfordernde Passagen erreichbar. Sie bieten keinen direkten Fortschritt oder Vorteil im Spielverlauf, sollen jedoch zu einem variablen Schwierigkeitsgrad beitragen. Sie sprechen insbesondere Spieler an, die durch Exploration, Vervollständigung oder die Beherrschung der Spielmechaniken motiviert sind.

Fokus-Steine, die jeweils am Ende eines Levels gesammelt werden müssen, sind erforderlich, um das nächste Level freizuschalten. Damit stellen sie eine essenzielle Ressource zum Fortschritt im Spiel dar.

Zusätzlich können in jedem Level Punkte gesammelt werden, die zu einem Highscore aufsummiert werden:

- Jeder Kristall bringt 10 Punkte.
- Jedes Collectible bringt 1000 Punkte.
- Pro Sekunde Spielzeit wird ein Punkt vom Gesamtscore abgezogen.

Obwohl es keine klassische Zeitbegrenzung gibt, wird die Zeit so zu einer indirekten Ressource. Sie soll Anreize erzeugen, ein Level möglichst schnell abzuschließen, ohne dabei auf das Einsammeln von Kristallen und Collectibles zu verzichten - was ein Gleichgewicht zwischen Schnelligkeit und Gründlichkeit erfordert.

4.4.6. Konflikte

Konflikte entstehen, wenn der Spieler versucht, seine Ziele im Rahmen der Regeln und Grenzen des Spiels zu erreichen. Sie werden bewusst in das Design integriert, um direkte Lösungswege zu verhindern und somit spielerische Herausforderungen zu schaffen. Dies geschieht, indem die verfügbaren Prozeduren absichtlich ineffizient für das Erreichen der Ziele gestaltet werden, sodass der Spieler bestimmte Fähigkeiten entwickeln und einsetzen muss, um Fortschritt zu erzielen.[42, S. 90]

In XR-Jump ergeben sich Konflikte vorwiegend aus den verschiedenen Hindernissen innerhalb der Spielwelt. Viele Passagen erfordern präzise Bewegung, um abgeschlossen zu werden. In anderen Abschnitten müssen Handinteraktionen eingesetzt werden, um Objekte zu manipulieren und dem Charakter den Weg freizumachen.

Zusätzlich gibt es auch konzeptuelle Hindernisse, also solche, die durch Regeln entstehen, die das Verhalten des Spielers einschränken [42, S. 91].

Dazu gehört der lineare Spielablauf, bei dem stets das vorherige Level abgeschlossen sein muss, um das nächste freizuschalten. Die Einschränkung, dass der Charakter nicht direkt gegriffen und somit nicht an eine andere Stelle „transportiert“ werden kann, wurde bewusst in das Spiel integriert. Auch die Charakter-Bewegung führt zu konzeptuellen Hindernissen, indem die Höhe des Sprungs oder die Reichweite des Dashs limitiert werden.

Diese konzeptuellen Einschränkungen sollen Abkürzungen verhindern und den Spieler dazu herausfordern, sich mit den Spielmechaniken auseinanderzusetzen und diese gezielt einzusetzen.

4.4.7. Grenzen

Grenzen trennen die Inhalte des Spiels von dem, was nicht zum Spiel gehört. Sie sind ein essenzielles Element des Game Designs, da sie das Spielprinzip maßgeblich strukturieren und beeinflussen. [42, S. 92]

In XR-Jump lassen sich Grenzen in zwei Kategorien unterteilen: Grenzen für den steuerbaren Charakter und Grenzen für den Spieler selbst.

Grenzen für den Charakter ergeben sich vor allem aus dem Level- und Welt-Design. Der Spielbereich wird durch natürliche Gegebenheiten wie Abgründe oder unüberwindbare Hindernisse eingeschränkt. Wo diese nicht ausreichen, werden unsichtbare Kollisionselemente (Colliders) eingesetzt, um z.B. Abkürzungen zu vermeiden oder zu verhindern, dass nach Abschluss eines Levels noch Ressourcen eingesammelt werden. Diese künstlichen Grenzen sind so in das Spielgeschehen integriert, dass sie möglichst unauffällig und nicht störend wirken.

Grenzen für den Spieler ergeben sich aus der physischen Umgebung. Theoretisch ist der Spieler im Tracking-Bereich der Quest 3 nur durch das Boundary-System bzw. den real verfügbaren Platz eingeschränkt. Obwohl sich der Spieler in den Leveln stets gemeinsam mit dem Charakter bewegt, wäre es prinzipiell möglich, sich innerhalb des verfügbaren Raums vom Charakter zu entfernen. Daher sind die Spielabschnitte so gestaltet, dass immer klar erkennbar ist, welche Zonen zum interaktiven Spielbereich gehören. Außerhalb dieser Zonen erstrecken sich „De-facto-Grenzen“: visuell uninteressante

Bereiche ohne Inhalt, die keinen Anreiz zur Erkundung bieten und somit als natürliche Barriere wirken. Auf diese Weise soll der Fokus des Spielers auf das eigentliche Spielgeschehen gelenkt werden, ohne durch harte Begrenzungen das Gefühl räumlicher Freiheit zu stören.

4.4.8. Ergebnis

Ein zentrales Merkmal von Spielen ist, dass der Ausgang ungewiss bleibt. Dies erzeugt Spannung und hält die Aufmerksamkeit des Spielers aufrecht. Im Interaktionsmuster „Einzelspieler versus Spiel“ ist der Ausgang typischerweise durch Sieg oder Niederlage definiert.[42, S. 96]

In XR-Jump hängt das Ergebnis innerhalb eines Levels davon ab, ob das Ende erreicht wird, ohne alle Leben zu verlieren. Scheitert der Spieler daran, wird er zum Levelanfang zurückgesetzt. Ein endgültiges „Game Over“ existiert jedoch nicht, da das Level jederzeit erneut gestartet werden kann.

Das übergeordnete Ergebnis des Spiels ist, alle Level erfolgreich abgeschlossen zu haben. Auch wenn dieser Ausgang durch die Struktur des Spiels vorhersehbar ist, sorgen die narrativen Elemente für zusätzliche Spannung und inhaltliche Ungewissheit. Somit wird eine zweite Ebene geschaffen, bei der der Ausgang des Spiels nicht allein durch den formalen Spielablauf, sondern auch durch die Entwicklung der Geschichte bestimmt wird.

4.5. Dramaturgische Elemente

Dramaturgische Elemente zielen darauf ab, den Spieler emotional anzusprechen und den Ausgang des Spiels interessanter zu gestalten. Sie stellen eine Verbindung zum Gameplay her und verleihen diesem einen Kontext, indem sie die formalen Elemente überlagern oder in eine bedeutungsvolle Geschichte integrieren. Die dramaturgischen Elemente lassen sich unter anderem kategorisieren in Herausforderung, Prämisse, Charakter, Geschichte und Spannung, sowie dem Aufbau der Welt.[42, S. 101 ff.]

Im Rahmen dieser Arbeit erhalten dramaturgische Elemente nach eigener Auffassung eine Relevanz, da sie auch zentrale Aspekte der Nutzererfahrung in XR beeinflussen können. Sie können die Ausbildung mentaler Modelle unterstützen, die Übereinstimmung mit der realen Welt fördern und dadurch die intuitive Verständlichkeit der Interaktionen verbessern - was ein zentrales Ziel der verwendeten Heuristiken darstellt. Die dramaturgische Gestaltung soll somit nicht nur ästhetisch, sondern auch funktional wirken.

4.5.1. Herausforderung

Herausforderung in Spielen bedeutet nicht bloß, dem Spieler schwierige Aufgaben zu stellen. Wäre dies der alleinige Zweck, unterschieden sich Herausforderungen im Spiel kaum von denen im realen Leben. Stattdessen sollten sie als lohnende Aufgaben gestaltet werden, deren Erfüllung befriedigend ist und ein Gefühl von Errungenschaft und Freude hervorruft. Die Herausforderung ist sehr individuell und bestimmt durch die Fähigkeiten des Spielers. Zudem verändert sich die wahrgenommene Herausforderung im Spielverlauf dynamisch, da Aufgaben, die zu Beginn noch schwierig erscheinen, mit wachsender Spielerfahrung an Komplexität verlieren.[42, S. 102]

Der Psychologe Mihaly Csikszentmihalyi beschreibt dieses Verhältnis in seiner „Flow"-Theorie: Beim Beginn einer Aktivität ist das Fähigkeitsniveau meist niedrig. Ist die Herausforderung dann zu groß, entsteht Frustration. Wird der Aktivität weiter nachgegangen, steigert sich das Fähigkeitsniveau - würde die Herausforderung dauerhaft gleich bleiben, stellt sich Langeweile ein. Der optimale Zustand - der sogenannte Flow - tritt dann ein, wenn die Anforderungen und die Fähigkeiten des Spielers im Gleichgewicht bleiben. [44] Genau diesen Zustand versucht man durch das Game Design zu erreichen, indem die Herausforderungen dynamisch an die Erfahrung des Spielers angepasst werden [42, S. 103].

In XR-Jump soll Flow unter anderem durch das von Kōichi Hayashida entwickelte Level-Design-Konzept erreicht werden (siehe Kapitel 4.2.1). Dabei werden neue Spielmechaniken zunächst in einer sicheren Umgebung eingeführt und anschließend zunehmend komplexer eingesetzt. Die Herausforderung steigt dabei allmählich und nachvollziehbar. [33]

Klar vorgegebene Ziele und Feedback unterstützen dabei die Entstehung von Flow [42, S. 103] [44]. In XR-Jump soll der Spieler stets wissen, was zu tun ist: Der lineare Aufbau und Story-Elemente weisen den Weg zum nächsten Abschnitt. Positives und negatives Feedback sind eindeutig: Fällt der Charakter in den Abgrund, verliert er ein Leben und muss die Passage erneut absolvieren. Erreicht er einen Checkpoint oder das Ende des Levels, ist der Fortschritt klar erkennbar.

Ein weiteres Element, zur Unterstützung des Flow-Zustands ist der uneingeschränkte Fokus auf das Spielgeschehen. Dies kann unter anderem durch audiovisuelle Reize unterstützt werden [42, S. 103] [44]. In XR-Jump könnte das XR-Medium diesen Fokus zusätzlich verstärken: Vor allem in den vollständig in VR verankerten Abschnitten kann der Spieler durch das HMD von externen Reizen abgeschirmt werden, wodurch ein stärkerer Fokus auf das Spielgeschehen erreicht werden könnte.

4.5.2. Prämisse

Die Prämisse legt die erzählerische Ausgangssituation eines Spiels innerhalb eines bestimmten Settings fest. Ohne sie wären viele Spiele zu abstrakt für den Spieler, um beim Spieler emotionale Resonanz zu erzeugen. Die Prämisse definiert Zeit und Ort, Hauptcharaktere, Ziele sowie Aktionen, die die Handlung vorantreiben.[42, S. 109 ff.]

XR-Jump beginnt in der realen Umgebung des Spielers, beispielsweise im Wohnzimmer, unterstützt durch die AR-Funktionen. Plötzlich öffnet sich ein magisches Portal, durch das ein kleiner Zauberer in die reale Welt tritt. Er ist auf der Suche nach einem Fokus-Stein, den er für seine magischen Experimente benötigt. Der Spieler hilft dem Zauberer, den Stein zu erreichen, woraufhin er vom Zauberer zum Dank in seine mittelalterliche Fantasiewelt eingeladen wird. Beim Betreten der Welt des Zauberers schließt sich das Portal jedoch unerwartet. Der Spieler ist gefangen. Nun müssen Spieler und Zauberer gemeinsam neue Fokus-Steine finden, um das Portal erneut zu öffnen und die Rückkehr in die reale Welt zu ermöglichen.

4.5.3. Charaktere

Charaktere und deren Handlungen sind zentrale Mittel zur Vermittlung der Spielgeschichte. Durch Identifikation mit einer Spielfigur können deren Erlebnisse emotional verinnerlicht und besser nachvollzogen werden. Dies erleichtert es dem Spieler, sich in die Entscheidungen und Lösungswege des Charakters einzufühlen. [42, S. 113]

In XR-Jump gibt es zwei Protagonisten:

Felix, der Zauberer, wird durch seine humorvolle und offene Persönlichkeit charakterisiert. Er stammt aus einer winzigen Parallelwelt und erscheint im Vergleich zum Spieler nur ca. zehn Zentimeter groß. Obwohl der Spieler Felix steuert, ist dieser kein leerer Avatar, sondern eine eigenständige Figur mit eigener Motivation, Geschichte und Persönlichkeit. Um die Balance zwischen spielerischer Kontrolle und narrativer Empathie zu wahren [42, S. 114], übernimmt Felix in Dialogen mit dem Spieler eine aktive Rolle, was den Plattformer-Elementen eine erzählerische Tiefe verleiht. Die Motivation von Felix zu Beginn des Spiels ist es, Fokus-Steine für seine Experimente zu sammeln. Später möchte er mit dem Spieler zusammen das verschlossene Portal zur realen Welt erneut öffnen.

Der Spieler selbst stellt den zweiten Protagonisten dar - mit der Besonderheit, dass er sich in der Spielwelt selbst verkörpert. Dies wird durch die Prämisse des Spiels ermöglicht, indem XR-Jump in der realen Umgebung des Spielers beginnt und so eine direkte Verbindung zur eigenen Person herstellt. Die Persönlichkeit des Spielers ist dabei nicht veränderbar, sondern ergibt sich aus seiner realen Identität. Daraus ergibt sich

eine erzählerische Struktur, in der der Spieler lediglich durch Aktionen auf Felix reagiert, beispielsweise durch die Unterstützung per Handinteraktion. Der Spieler nimmt in der Geschichte also ebenfalls eine bedeutende Rolle ein: Er hat ein klares Ziel, nämlich die Rückkehr in die eigene Welt. Diese Motivation soll so zum zentralen Antrieb für die gesamte Spielerfahrung werden.

4.5.4. Geschichte

Die Geschichte in XR-Jump verläuft linear, das Gameplay hat also keinen Einfluss auf den Handlungsverlauf. Der Spieler muss lediglich die einzelnen Level abschließen, um zum nächsten Punkt in der Handlung zu gelangen. Die grundlegende Handlung ist bereits durch die Prämisse vorgegeben. Sie dient dabei nicht als tiefgehende Story, sondern als unterstützender Rahmen für das Spielgeschehen.

Gelingt es dem Spieler gemeinsam mit Felix, alle Fokus-Steine zu sammeln, also sämtliche Level erfolgreich zu absolvieren, öffnet sich das Portal zur realen Welt erneut, und der Spieler kann nach Hause zurückkehren.

Spannung kann durch gezielt eingesetzte Unklarheiten innerhalb der Erzählung entstehen. So bleibt beispielsweise lange offen, ob der Spieler letztlich in seine eigene Welt zurückkehren kann. Auch die Entscheidung, dem Zauberer zu vertrauen und durch das Portal in die fremde Welt zu treten, kann das Interesse an der Handlung verstärken.

Diese Geschichte soll mehrere Funktionen erfüllen: Sie kann eine nachvollziehbare Einführung in die Mechaniken und in die Spielwelt ermöglichen. Sie verbindet AR- und VR-Elemente und soll durch eine klare übergeordnete Zielsetzung motivieren. Gleichzeitig erlaubt sie eine emotionale Bindung zum Begleitercharakter, verdeutlicht die Rolle des Spielers selbst und bietet eine gewisse emotionale Tiefe im Spielverlauf.

4.5.5. Aufbau der Welt

Das Spiel beginnt und endet in der realen Welt, also in der jeweils aktuellen Umgebung des Spielers. Die eigentliche Spielwelt, eine Parallelwelt namens „Wigglewick“, ist im Verhältnis zur realen Welt stark verkleinert dargestellt. Diese gestalterische Entscheidung unterstützt das zentrale Spielprinzip, bei dem der Spieler durch seine Größe in der Welt mittels Handinteraktionen direkt mit der Spielwelt interagieren kann. Dadurch wird das Interaktionskonzept nicht nur spielmechanisch ermöglicht, sondern auch plausibel erklärt.

Wigglewick ist als mittelalterlich inspirierte Fantasiewelt konzipiert, die sich aus Wasserflächen und schwebenden Inseln zusammensetzt. Die Levelauswahl befindet sich auf

einer Gruppe solcher Inseln und stellt gleichzeitig die Heimat des Zauberers dar. Hier befindet sich unter anderem sein Haus sowie mehrere Portale, die als Zugangspunkte zu den einzelnen Level dienen. Dies ist außerdem der Bereich, in dem der Spieler nach dem Übergang in die Welt des Zauberers ankommt.

Der aktuelle Prototyp des Spiels umfasst neben der Levelauswahl zwei weitere Levelabschnitte: Das erste Level führt über mehrere schwebende Inseln zur Spitze eines großen Berges. Das zweite Level ist in einem wassergefüllten Tal zwischen zwei Bergketten angesiedelt und unterscheidet sich im Aufbau und in der Atmosphäre vom ersten Level.

Ein detailliertes World-Building wurde im Rahmen dieser Arbeit bewusst zurückgestellt, da es weder für die vergleichsweise einfache Handlung noch für das Spielgenre eine zentrale Rolle spielt. Aus diesem Grund enthält die Welt keine tiefergehenden geschichtlichen oder kulturellen Hintergründe.

5. Entwicklung des Prototyps

5.1. Anforderungsanalyse

Zu Beginn der Entwicklungsphase ist es erforderlich, die Anforderungen an das zu entwickelnde Spiel zu erfassen. Grundlage dafür bilden die im Game Design konzipierten formalen und dramaturgischen Elemente, die in Kombination mit den von Nielsen und Vi et al. herangezogenen Design-Heuristiken (siehe Kapitel 3.1 und 3.2) zu konkreten Anforderungen für den Prototyp führen [4] [11].

5.1.1. Zielsetzung des Systems

Ziel ist die Entwicklung eines XR-Plattformers, bei dem der Spieler eine Spielfigur mithilfe eines Controllers steuert und gleichzeitig mit der freien Hand direkt in die Spielwelt eingreifen kann.

Der Prototyp richtet sich an die im Game Design definierte Zielgruppe, primär bestehend aus Casual- bis Core-Gamern. Gleichzeitig soll der Zugang zum Spiel möglichst niederschwellig gestaltet sein, um auch ein breiteres Publikum anzusprechen.

Das Interaktionsdesign orientiert sich an den erarbeiteten Heuristiken und Design-Prinzipien. Dadurch soll eine intuitive und komfortable Steuerung gewährleistet sowie potenziellen Beeinträchtigungen wie der VR-Krankheit gezielt vorgebeugt werden.

Die technische Umsetzung konzentriert sich in erster Linie auf die formalen Elemente des Game Designs, die durch dramaturgische Aspekte ergänzt und unterstützt werden, um ein konsistentes und immersives Spielerlebnis zu schaffen.

5.1.2. Funktionale Anforderungen

Funktionale Anforderungen beschreiben, welche Leistungen das System erbringen muss, um die zentralen Ziele seiner Entwicklung zu erfüllen [45].

Einige funktionale Anforderungen wurden erst im Verlauf des Entwicklungsprozesses identifiziert und ergänzt. Im Folgenden werden die finalen funktionalen Anforderungen

dargestellt, um eine nachvollziehbare Grundlage für die weitere technische Umsetzung zu schaffen.

Tabelle 5.1.: Funktionale Anforderungen

Nr.	**Anforderung**	**Beschreibung**
F1	Charaktersteuerung per Controller	Die Spielfigur kann aus der Third-Person-Perspektive mit einem einhändigen Controller durch die Spielwelt gesteuert werden.
F2	Bewegung des Spielers	Der Spieler kann sich physisch im Raum bewegen. Das XR-Rig folgt der Spielfigur, wobei zwischen unterschiedlichen Komfortmodi gewählt werden kann.
F3	Handinteraktion: Greifen	Objekte können mit der freien Hand direkt oder aus der Ferne gegriffen und bewegt werden.
F4	Handinteraktion: Posen	Durch definierte Handposen können bestimmte Aktionen ausgelöst oder Objekte beeinflusst werden.
F5	XR-optimierte Benutzeroberfläche	Interaktive UI-Elemente wie Menüs sind auf die Interaktion im XR-Kontext ausgelegt.
F6	Zentrierung des XR-Rigs	Das XR-Rig kann an der vorgesehenen Position ausgerichtet werden.
F7	AR/VR-Übergang	Ein Portal-Mechanismus ermöglicht den Übergang zwischen realer (AR) und virtueller Welt (VR) sowie zurück.
F8	Speicherfunktion	Spielstand und Fortschritt (z.B. absolvierte Level, Highscores) werden persistent gespeichert und können wiederhergestellt werden.
F9	Sammelobjekte	Der Charakter kann verschiedene Ressourcen wie Kristalle, Münzen und Fokus-Steine einsammeln.
F10	Lebenssystem	Der Charakter verfügt über ein Leben-System, in dem Leben gewonnen oder verloren werden können.
F11	Dialogsystem	Dialoge mit vertonter Sprachausgabe und synchronisierten Motion-Capture-Animationen werden unterstützt.
F12	Animationssystem	Bewegungen der Spielfigur und anderer interaktiver Elemente werden animiert dargestellt.
F13	Levelarchitektur	Levelaufbau mit Triggern, Checkpoints und Spawnpunkten.

Nr.	Anforderung	Beschreibung
F14	Weltstruktur	Die Spielwelt ist in Abschnitte unterteilt, deren Sichtbarkeit und Ladezustand durch eine Logik gesteuert wird.

5.1.3. Nicht-funktionale Anforderungen

Nicht-funktionale Anforderungen beschreiben die qualitativen Eigenschaften eines Systems. Sie betreffen Aspekte wie Nutzbarkeit, Leistungsfähigkeit, Zuverlässigkeit, Sicherheit oder Attraktivität der Anwendung. Sie sind notwendig, da Nutzer erwarten, dass funktionale Anforderungen auf eine bestimmte Art und Weise und mit einem bestimmten Grad an Qualität funktionieren. [45]

Tabelle 5.2.: Nicht-funktionale Anforderungen

Nr.	Anforderung	Beschreibung
NF1a	Usability: Übereinstimmung mit der realen Welt	Interaktionen orientieren sich an bekannten mentalen Modellen und wirken dadurch intuitiv.
NF1b	Usability: Konsistenz	Es wird sowohl auf interne Konsistenz innerhalb der Anwendung als auch auf Konformität mit Konventionen der Meta Quest Plattform geachtet.
NF1c	Usability: Sicherheit	Der Nutzer wird nicht zu gefährlichen Handlungen verleitet und durch Sicherheitsfeatures wie Begrenzungen geschützt.
NF1d	Usability: Anpassbarkeit	Die Anwendung kann an individuelle Präferenzen angepasst werden, insbesondere zur Vermeidung von Motion Sickness.
NF1e	Usability: Feedback	Nutzer erhalten konsistentes, visuelles und akustisches Feedback auf ihre Aktionen.
NF1f	Usability: XR-spezifische Heuristiken	Die Gestaltung orientiert sich an heuristischen Prinzipien für XR-Anwendungen.
NF2	Performance	Die Anwendung läuft flüssig auf dem Zielsystem mit stabiler Bildrate.
NF3	Kompatibilität	Die Anwendung funktioniert nativ und vollständig autark auf der Meta Quest 3 ohne zusätzliche Hardware.

Nr.	Anforderung	Beschreibung
NF4	Gestalterische Konsistenz	Alle visuellen, auditiven und interaktiven Elemente folgen einer gemeinsamen, thematisch passenden Designlinie.

5.2. Technologie-Stack

Für die technische Umsetzung für die Meta Quest 3 wird die Unity Game Engine (Editor Version 6000.0.42f1) mit dem OpenXR-Plugin verwendet. OpenXR ist ein von der Khronos-Group entwickelter, gebührenfreier Standard, der die Entwicklung von XR-Anwendungen durch ein einheitliches Interface für u.a. Tracking, Rendering und Controller-Input vereinfacht [46].

Zusätzlich wird das Meta XR All-in-One SDK (Version 74) verwendet. Dieses vereint Funktionen des Meta-Ökosystems wie Hand-Tracking, Passthrough, Spatial Audio und Interaktionsmodelle (z.B. das Interaction SDK) in einem zentralen Software Development Kit [47]. Die Integration interaktiver XR-Funktionalitäten wird durch sogenannte Building Blocks erleichtert, über die beispielsweise ein vorkonfiguriertes Kamera-Rig direkt in der Szene platziert werden kann. Die Verwendung von OpenXR in Kombination mit dem Meta SDK ist das empfohlene Vorgehen von Meta für die Entwicklung in Unity. [48]

Alle Skripte, die während der Entwicklung entstehen, werden in der von Unity unterstützten, objektorientierten Programmiersprache C# erstellt [49].

Für Motion-Capture-Animationen wurde Animotive verwendet. Dieses ermöglicht die Aufnahme von Körperanimationen über ein HMD und den anschließenden Export in Unity [50]. Die synchron in Animotive aufgenommenen Sprachaufnahmen wurden mithilfe der ElevenLabs Voice Changer KI verändert [51].

Die für das Spiel genutzten 3D-Modelle, Sounds und Musikstücke wurden nicht selbst erstellt, sondern durch verschiedene Pakete aus dem Unity Asset Store oder aus anderen Quellen bezogen.

5.3. Architektur der Anwendung

Szenenstruktur: Das Unity-Projekt ist in sogenannten Szenen strukturiert, die jeweils eine in sich geschlossene Umgebung mit GameObjects und Komponenten darstellen. Diese repräsentieren bestimmte Abschnitte oder Zustände innerhalb des Spiels. Die Szenenstruktur von XR-Jump gliedert sich in folgende Bereiche:

- **Menüszene:** Startumgebung mit UI-Elementen zur Spielnavigation und Konfiguration. In dieser Szene werden zudem Spieldaten geladen.
- **Intro-Szene:** Beginn im Passthrough-Modus (AR), inklusive Intro-Sequenz, interaktiver Umgebung, Levelauswahl und einem MR-Portal zum Wechsel zwischen AR und VR.
- **Level-Auswahl:** Duplikat der Intro-Szene ohne Intro-Sequenz. Der Spieler startet direkt in VR. Zusätzlich enthält diese Szene die Outro-Sequenz sowie das zugehörige MR-Portal für das Spielende.
- **Level01 und Level02:** Eigenständige Szenen, welche die jeweilige Spielwelt, interaktive Objekte und Gameplay-Elemente enthalten.

Modularisierung: Die Anwendung ist komponentenbasiert aufgebaut. Wiederverwendbare Objekte wurden als Prefabs gespeichert, wodurch sie in verschiedenen Szenen eingesetzt werden können. C#-Skripte auf Basis der Unity-eigenen MonoBehaviour-Klasse werden als Komponenten an GameObjects angehängt. Zahlreiche Unity-Komponenten stehen außerdem für Standardfunktionen wie Eingabeverarbeitung oder Animationen zur Verfügung. Die wichtigsten funktionalen Module sind im Folgenden aufgelistet:

- **XR-Rig:** Basiert auf dem XR Interaction SDK [52] von Meta und stellt Tracking-Funktionen, sowie Hand- und Controller-Interaktionen zur Verfügung. Konfiguriert für Quest 3, simultane Controller- und Handinteraktion sowie Passthrough-Unterstützug.
- **Input-System:** Das Unity Input System in Kombination mit der PlayerInput-Komponente ermöglicht die Erfassung und Verarbeitung der Eingaben des Touch Plus Controllers, z.B. für die Charakterbewegung.
- **Animations-System:** Die Animationen basieren auf Unitys Keyframe-System. Animator-Komponenten steuern Zustandswechsel animierter Objekte. Die Timeline-Funktion ermöglicht die Inszenierung ganzer Sequenzen. Über Signale oder Animation Events werden gezielt Skriptfunktionen aufgerufen, z.B. zum synchronen Abspielen von Audiodateien.
- **UI-System:** Aufbauend auf World-Space-Canvases und durch das Meta Interaction SDK ergänzte Komponenten wird eine für XR optimierte Benutzeroberfläche realisiert. Eigene Skripte steuern die UI-Navigation und Darstellung dynamischer Inhalte.

- **Handinteraktionssystem (Eigenentwicklung):** Eigene Skripte regeln die Logik und Visualisierung von Grab-Events, das Verhalten interaktiver Objekte sowie akustisches und visuelles Feedback bei Interaktionen.
- **Charaktersteuerung (Eigenentwicklung):** Skripte für Bewegung, Kameraanpassung (XR-Rig), Ground Detection sowie das Mapping von Eingaben auf Bewegungsabläufe ermöglichen die Steuerung der Spielfigur.
- **Checkpoint- und Respawn-System (Eigenentwicklung):** Mechanismen, die das aktivieren von Checkpoints und den Respawn des Charakters steuern.
- **Ressourcen-System (Eigenentwicklung):** Verwaltung sammelbarer Objekte wie Kristalle, Leben oder Collectibles. Inklusive Feedbacksystem und Speicherung.
- **AR/VR-Übergangslogik (Eigenentwicklung):** Skripte für die Portalmechanik, mit der Spieler oder Objekte zwischen realer und virtueller Welt wechseln können.
- **Szenen-Controller (Eigenentwicklung):** Steuert den Ablauf von Spielabschnitten wie Intro, Outro oder einzelne Level.
- **Game-Data-Verwaltung (Eigenentwicklung):** Verwaltung persistenter Spieldaten wie Spielfortschritt, Highscores und benutzerdefinierte Einstellungen.
- **Umgebungs- und Hilfssysteme (Eigenentwicklung):** Weitere spezialisierte Skripte steuern z.B. das Verhalten dynamischer Objekte (z.B. Plattformen), Umgebungselemente (z.B. Wolkenanimationen), passen die Spielumgebung an die Körpergröße des Spielers an oder regeln performancerelevante Funktionen wie Foveated Rendering.

Insgesamt wurden während der Entwicklung 57 C#-Skripte erstellt. Im Rahmen der schriftlichen Arbeit wird jedoch nicht auf sämtliche Skripte im Detail eingegangen, sondern exemplarisch auf zentrale, besonders relevante Funktionen. Der vollständige Quellcode ist in Anhang A gelistet.

5.4. Implementierung der Funktionalitäten

Dieser Abschnitt befasst sich mit der Entwicklung der in der Anforderungsanalyse festgehaltenen Funktionen.

5.4.1. Charaktersteuerung (F1)

Die Steuerung des Charakters erfolgt über den einhändigen Controller der Meta Quest 3 mit Verwendung des Unity Input Systems. Ziel war es, ein intuitives und responsives Bewegungsverhalten zu gestalten, das typische Jump'n'Run-Mechaniken wie Laufen, Springen und Dashen unterstützt.

Aufbau und Komponenten: Der Charakter ist ein GameObject, welches Unitys Physik-System unterliegt und aus folgenden Komponenten und Skripten aufgebaut ist:

- **PlayerInput** (Unity-Komponente): Stellt Eingabeevents über ein InputActionAsset zu Verfügung.
- **Rigidbody und CapsuleCollider** (Unity-Komponenten): Ermöglichen realistische Bewegung und Kollisionen basierend auf physikalischen Kräften.
- **Animator** (Unity-Komponente): Steuert die Animationen über Blend Trees und Bool-Parameter.
- **Char_InputManager** (Skript): Interpretiert die Eingabeevents des PlayerInput.
- **Char_Movement** (Skript): Setzt Eingaben in physikalische Bewegung um und steuert Animationen.
- **GroundDetector** (Skript): Ermittelt, ob sich der Charakter am Boden befindet.

Eingabe und Bewegungsverarbeitung: Die Eingabe erfolgt über ein InputActionAsset, das Bewegungsrichtung, Sprung-, Dash- und Interaktionsaktionen verarbeitet. Der `Char_InputManager` übersetzt diese in Zustände, die im FixedUpdate vom `Char_Movement` abgefragt werden. Der Bewegungsvektor wird dabei kamerarelativ berechnet, um ein konsistentes Steuerungsgefühl aus der Third-Person-Perspektive zu ermöglichen.

Die Bewegung erfolgt über eine Manipulation der Rigidbody-Geschwindigkeit mithilfe von `AddForce` und `VelocityChange`, wobei die maximale Geschwindigkeit durch `ClampMagnitude` limitiert wird. Dies ermöglicht realistische Bewegungen, bei denen auch externe Kräfte (z.B. bewegliche Plattformen) berücksichtigt werden. Eine Auswahl an Schrittgeräuschen wird zufällig über eine Funktion abgespielt, die über Animations-Events in der Bewegungsanimation des Charakters aufgerufen wird.

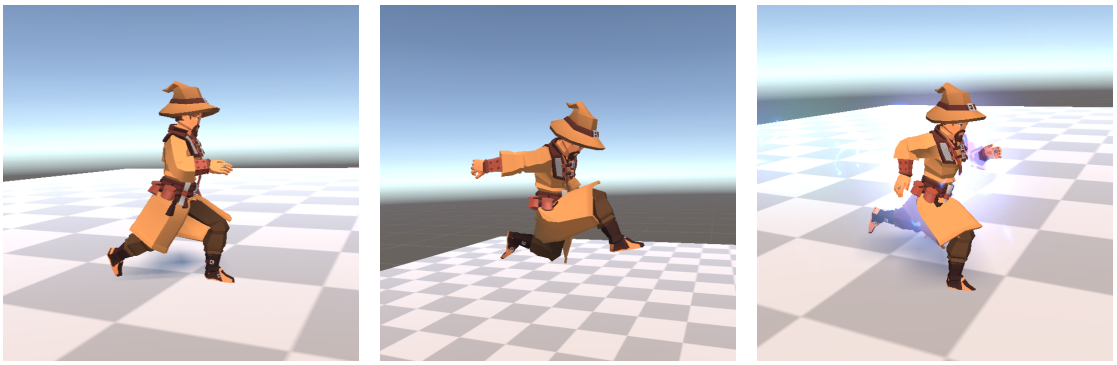
Die Sprungmechanik wird realisiert durch einen kurzen Impuls nach oben, kombiniert mit einem nach vorne gerichteten Schub, der sich an die aktuelle Eingabestärke für die

Bewegung orientiert. Wird der Sprungbutton länger gehalten, kann durch zusätzlichen Schub ein „Long Jump“ ausgeführt werden.

Das Dash-System erlaubt eine kurze Beschleunigung in Bewegungsrichtung, wird durch Partikeleffekte und Audiofeedback unterstützt und unterliegt einem Cooldown.

Ground Detection: Das `GroundDetector`-Skript nutzt einen `SphereCast`, um eine Bodenüberprüfung inklusive maximal erlaubtem Bodenneigungswinkel umzusetzen. Dadurch werden fehlerhafte Sprungerkennungen vermieden. Außerdem greifen hierdurch der `airAccelerationFactor` und `airRotationFactor` des `Char_Movement`, welche die Kontrolle über den Charakter in der Luft reduzieren.

Der Code für die Charaktersteuerung befindet sich im Verzeichnis `Charaktersteuerung` auf dem Datenträger (siehe Anhang A.1).



(a) Laufbewegung. (b) Sprung. (c) Dash mit Partikel-Effekt.

Abbildung 5.1.: Charakterbewegung (Screenshots aus der Unity Engine).

5.4.2. Bewegung des Spielers (F2)

Der Spieler (XR-Rig) folgt dem Charakter im Raum - entweder sanft gleitend oder per Dash-Sprung. Zusätzlich gibt es einen Komfortmodus (Diorama-Ansicht), in dem das XR-Rig fest positioniert ist und über Trigger gewechselt werden kann.

Aufbau und Komponenten: Der Spieler wird in Unity repräsentiert durch das XR-Rig, welches über einen Building-Block des Meta-SDKs hinzugefügt und entsprechend konfiguriert wurde. Dieses ermöglicht direkt die Bewegung des Spielers im Raum durch reale physische Bewegungen. Damit der Spieler in der Nähe des Charakters bleibt, wenn sich dieser bewegt, werden folgende Komponenten benötigt:

- **XR-Rig** (Meta SDK): Beinhaltet die XR-Kameras und den Tracking-Space, in dem sich der Spieler frei durch reale Bewegungen bewegen kann.

- **OVRVignette** (Meta SDK) [53]: Wird genutzt um das Sichtfeld des Spielers (z.B. bei Bewegungen) zu verkleinern.
- **XRPlayer_Follow** (Skript): Enthält die Logik zum Verfolgen des Charakters in verschiedenen Modi.
- **CheckpointCameraSwitcher** (Skript): Steuert den Wechsel der Diorama-Kamera, wenn der Charakter bestimmte Zonen betritt.

Standardkonfiguration: Das zentrale Element für die Bewegung des Spielers ist das Skript `XRPlayer_Follow`, welches direkt an das XR-Rig angehängt wird. In der Standardkonfiguration folgt der Spieler dem Charakter durch sanftes Gleiten. Die Positionierung des Rigs erfolgt in einem festen, aber einstellbaren Winkel zur Weltachse und in einer definierten Distanz zum Charakter. Die Bewegung wird mittels `Vector3.SmoothDamp()` geglättet, wodurch eine sanfte Nachverfolgung entsteht. Sie wird aufgeteilt in horizontale (X/Z) und vertikale (Y) Bewegung. Dies erlaubt eine separate Glättung der Höhe, mit einer geringeren Reaktionszeit, um schnelle vertikale Verschiebungen des Spielers zu verhindern, z.B. wenn der Charakter springt.

Zur Steigerung des Komforts, insbesondere in Hinblick auf Motion-Sickness, wird optional eine „Comfort-Vignette“ verwendet. Diese wird durch die Komponente `OVRVignette` aus dem Meta SDK bereitgestellt und über `XRPlayer_Follow` dynamisch gesteuert. Die Intensität der Vignette wird dabei in Abhängigkeit zur Geschwindigkeit des XR-Rigs angepasst. Bei schneller Bewegung wird das Sichtfeld (Field of View, FOV) stärker reduziert, was das periphere Sichtfeld einschränkt und das Gefühl von Übelkeit bei empfindlichen Nutzern minimieren kann [21].

Dash-Movement: Zusätzlich zur Standardbewegung unterstützt das System auch das optional aktivierbare sogenannte „Dash-Movement“. Anstelle einer sanften Bewegung verharrt das XR-Rig an einer Stelle, bis die Distanz zum Charakter einen festgelegten Schwellenwert überschreitet und eine kurze Wartezeit (Cooldown) vergangen ist. In diesem Fall wird das Rig schlagartig in Richtung Zielposition bewegt, was die VR-Krankheit verringern soll [21].

Komfort-Modus: Im „Komfort-“ oder „Diorama-Modus“ folgt das XR-Rig dem Spieler nicht mehr dynamisch, sondern befindet sich an statischen, vordefinierten Positionen innerhalb der Spielwelt. Diese Positionen werden durch Trigger-Zonen im Level definiert, welche mit dem Skript `CheckpointCameraSwitcher` ausgestattet sind. Beim Betreten dieser Zonen durch den Charakter wird überprüft, ob sich das Spiel im Komfortmodus

befindet, und setzt dann die Position des Rigs sowie dessen Rotation entsprechend. Hierbei wird zusätzlich der Versatz zwischen dem Mittelpunkt des Tracking-Space und dem Kopf des Spielers berücksichtigt, damit dieser an der korrekten Position platziert wird, auch wenn er sich vorher durch reale Bewegungen im physischen Raum vom Mittelpunkt des Tracking-Space entfernt hat.

Der Code für die Bewegung des Spielers befindet sich im Verzeichnis `Spielerbewegung` auf dem Datenträger (siehe Anhang A.2).

5.4.3. Handinteraktion: Greifen (F3)

Die Interaktion mit der Umwelt erfolgt über die nicht für die Controller-Steuerung genutzte, freie Hand. Der Spieler kann Objekte greifen, um neue Wege zum Überwinden von Hindernissen für den Charakter zu schaffen.

Aufbau und Komponenten: Die Greifinteraktionen im Spiel wurden mithilfe der Interaction SDK von Meta und eigenen Erweiterungen realisiert. Folgende Komponenten sind zentral für die jeweilige Greif-Methode (nah oder aus der Distanz):

- **XR-Rig mit Interaction-Building-Block** (Meta SDK): Das XR-Rig wird mit dem Interaction-Building-Block erweitert, welches die Komponenten für Handinteraktionen, z.B. die synthetischen Hände des Spielers, bereitstellt.
- **TouchHandGrab-Objekt** (Meta SDK) [54]: Wird als Kindobjekt zu greifbaren Objekten hinzugefügt und enthält Komponenten wie `TouchHandGrabInteractable`, um das direkte Greifen zu ermöglichen.
- **GrabbableEventHandler** (Skript): Löst UnityEvents (`OnGrabbed/OnReleased`) bei Greif-Interaktionen aus und steuert so weitere Interaktionen (z.B. visuelles Feedback). Spielt Feedback-Sounds beim Greifen und Loslassen von Objekten ab.
- **GrabbableSoundController** (Skript): Gibt ein an die Bewegungsgeschwindigkeit angepasstes Geräusch während des Haltens eines Objekts wieder.
- **HandCollisionTrigger** (Skript): Erkennt, ob sich die Hand in einem bestimmten Trigger-Bereich (z.B. Hover-Zone um ein greifbares Objekt) befindet. Löst UnityEvents für visuelles Feedback oder weitere Spielinteraktionen aus.
- **OutlineController** (Skript): Aktiviert visuelle Umrandungen für Hover- oder Grab-Zustände (auf Basis des „Quick Outline“-Packages).
- **Outline** (QuickOutline-Package) [55]: Ermöglicht das Zeichnen farbiger Outlines um Meshes.

- **OneGrabRotateTransformer** (Meta SDK) [56]: Beschränkt die Rotation greifbarer Objekte, etwa bei aufklappbaren Plattformen.

Für Distanz-Interaktionen gelten teils angepasste Komponenten:

- **DistanceHandGrab-Objekt** (Meta SDK) [57]: Entspricht funktional dem Touch-Objekt, ist jedoch für Ferninteraktion optimiert.
- **DistanceGrabEventHandler** (Skript): Analog zum `GrabbableEventHandler`, angepasst für Distanz-Interaktionen.
- **OneGrabTranslateTransformer** (Meta SDK) [58]: Ermöglicht lineare Objektbewegung innerhalb eines Bereiches auf einer Achse, z.B. für das Verschieben von Plattformen in eine Richtung.

Implementierung: Greifbare Objekte werden zur Wiederverwendung in Prefabs organisiert. Jedes Objekt erhält dabei eine Kombination aus den oben genannten Komponenten.

Direktes Greifen: Die Implementierung basiert auf dem `TouchHandGrabInteractable` der Meta SDK. Der `GrabbableEventHandler` hört auf dessen Statusänderungen (z.B. Übergang in den `Select`-Zustand) und triggert UnityEvents. Damit wird Feedback ausgelöst - akustisch (über zugewiesene AudioClips) und visuell (über den `Outline-Controller`). Parallel aktiviert der EventHandler den `GrabbableSoundController`, der über ein an die Bewegungsgeschwindigkeit angepasstes Loop-Sample dynamisches Soundfeedback gibt.

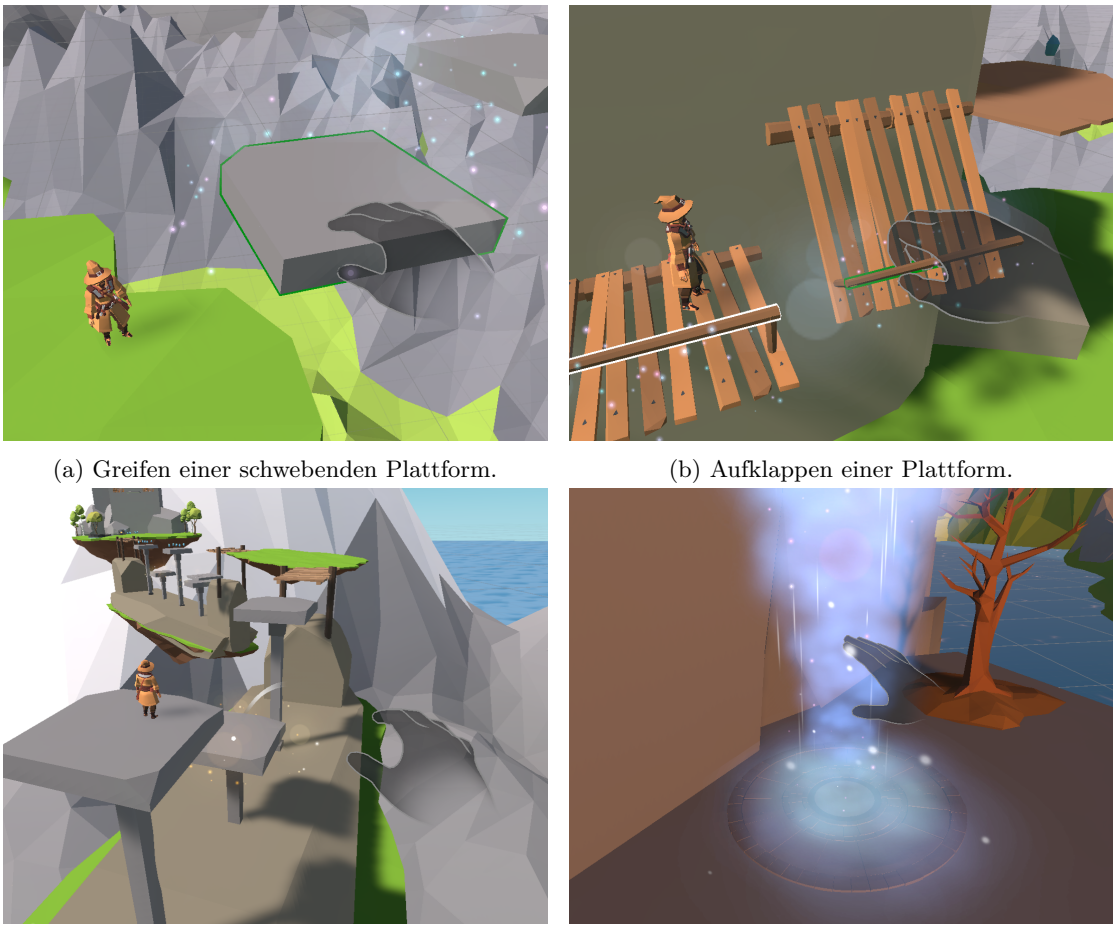
Für eine konsistente Nutzungserfahrung werden verschiedene Outline-Farben eingesetzt: Befindet sich die Hand in der Nähe des greifbaren Objekts, werden weiße Umrandungen angezeigt - bei gehaltenen Objekten eine grüne. Dies erfolgt über direkte Aufrufe des `OutlineController`, gesteuert durch Events aus `GrabbableEventHandler` und `HandCollisionTrigger`.

Distanz-Greifen: Objekte, die aus der Entfernung gegriffen werden können, verwenden das `DistanceHandGrabInteractable`. Die Eventbehandlung erfolgt analog über den `DistanceGrabEventHandler`. Die Bewegung der Objekte wird über den `OneGrab-TranslateTransformer` eingeschränkt. Dieser begrenzt die Verschiebung auf bestimmte Achsenbereiche, etwa beim Verschieben von Treppen oder dem Heben von Plattformen.

Hover-basierte Interaktionen: Der `HandCollisionTrigger` wird auch verwendet, um einfache Hover-Aktionen zu erkennen, bei denen keine Grab-Interaktion notwendig ist. Er erkennt, ob sich eine Handkapsel im Triggerbereich befindet und kann beliebige UnityEvents auslösen, z.B. visuelle oder spielmechanische Reaktionen. Dies erlaubt zusätzliche Interaktionsformen, z.B. das Aktivieren von Kraftfeldern durch Bewegung der Hand in deren Einflussbereich.

Visuelles Feedback: Neben den Outlines, die den Zustand der Objekte visualisieren, werden farbige Partikeleffekte verwendet. Diese signalisieren dem Nutzer, ob es sich um ein direkt greifbares, aus der Ferne greifbares oder Hover-Objekt handelt.

Der Code für die Greifinteraktionen befindet sich im Verzeichnis `HandinteraktionGreifen` auf dem Datenträger (siehe Anhang A.3).



(a) Greifen einer schwebenden Plattform.

(b) Aufklappen einer Plattform.

(c) Distanz-Greifen und Verschiebung in der Höhe.

(d) Kraftfeld-Aktivierung durch Hovern.

Abbildung 5.2.: Handinteraktion: Greifen (Screenshots aus der Unity Engine).

5.4.4. Handinteraktion: Posen (F4)

Neben dem Greifen wurden für das Spiel auch Handposen als Eingabemechanismus verwendet. So können Aktionen ausgelöst werden, indem der Spieler bestimmte Gesten mit der freien Hand formt.

Aufbau und Komponenten: Zur schnellen Implementierung der Posen-Erkennung wurden, neben dem für Handinteraktion erweiterten XR-Rig, von der Meta SDK bereitgestellte Pose-Prefabs verwendet [59]. Diese bieten eine vorkonfigurierte Möglichkeit, bestimmte Handgesten zu erkennen und direkt Unity-Events auszulösen.

Durch die Interaction-SDK wäre es auch möglich, eigene Posen zu erstellen. Die Wahl fiel jedoch bewusst auf die vorgefertigten Komponenten, da sie sich als zuverlässig erwiesen und sich schnell in das bestehende Projekt integrieren ließen.

Darauf aufbauend wurden Skripte entwickelt, welche Spiellogik, Zustandsverwaltung und Reaktion auf die Posen-Erkennung übernehmen.

Zum einen wurden bewegliche Plattformen entworfen, die durch Posen in Bewegung versetzt, beziehungsweise gestoppt werden können. Diese bestehen aus folgenden Komponenten:

- **MovingPlatform** (Skript): Steuert eine Platform entlang definiertere Wegpunkte. Die Bewegung kann pausiert und wieder gestartet werden. Über Eigenschaften wie Geschwindigkeit, Wartezeiten oder Dämpfungsbereiche lassen sich verschiedene Bewegungsmuster umsetzen. Sie werden auch für bewegliche Plattformen im Spiel eingesetzt, die sich nicht durch Posen steuern lassen.
- **PoseControlledPlatforms** (Skript): Dient als zentrale Steuerkomponente für alle untergeordneten `MovingPlatform`-Instanzen und reagiert auf Posen-Events.

Zum anderen wurden klappbare Plattformen implementiert, welche durch Posen zwischen eingeklappten und ausgeklappten Zustand wechseln können. Diese sind durch folgende Komponenten aufgebaut:

- **FoldablePlatform** (Skript): Steuert den Zustand einzelner klappbare Plattformen.
- **FoldablePlatformsPoseManager** (Skript): Verwaltet mehrere `FoldablePlatform`-Instanzen und stellt Methoden bereit, um diese gesammelt über die Posen umzuschalten.

Bewegliche Plattformen: Die beweglichen Plattformen reagieren auf zwei Posen: Eine flache, nach vorne ausgestreckte Hand startet die Bewegung der Plattform, während eine flache, nach oben gerichtete Hand sie wieder stoppt. Die eigentliche Bewegung wird durch ein eigenständiges Skript `MovingPlatform` gesteuert, das die Plattform entlang vordefinierter Wegpunkte bewegt. Dabei kann für jeden Wegpunkt eine Wartezeit und ein Dämpfungsbereich angegeben werden, in dem die Plattform sanft abbremst.

Um die Plattformen gemeinsam über Posen kontrollieren zu können, wurde eine übergeordnete Komponente `PoseControlledPlatforms` genutzt. Dieses GameObject fungiert als Parent aller untergeordneten MovingPlatform-Instanzen. Es enthält einen Trigger-Collider, der prüft, ob sich der Spielcharakter in der Nähe befindet, denn nur dann werden Posen berücksichtigt. Die Methoden `ActivateWithPose()` und `DeactivateWithPose()` werden durch die `SelectorUnityEventWrapper` der Pose-Prefabs aufgerufen. Dadurch lassen sich beliebig viele Plattformen gleichzeitig durch eine Geste aktivieren oder deaktivieren.

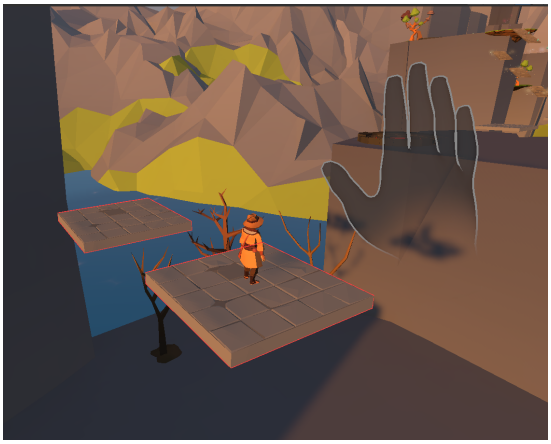
Visuelles Feedback wird durch das „Quick Outline“-Paket [55] gegeben: Die Plattformen erhalten Umrisse, deren Farbe sich je nach Zustand (aktiv oder inaktiv) ändert. Wird eine Pose erkannt, wird außerdem ein jeweiliger Feedback-Sound abgespielt.

Klappbare Plattformen: Die klappbaren Plattformen werden mittels einer Daumen-hoch- bzw. Daumen-runter-Geste ein- oder ausgeklappt. Dies ermöglicht es, Sprungpassagen zu gestalten, bei denen der Zustand der Plattformen aktiv durch den Spieler, beispielsweise bei jedem Sprung, verändert werden muss. Jede Plattform besitzt ein eigenes `FoldablePlatform`-Skript, sowie einen Animator, der die Klapp-Animation entsprechend steuert. Es kann eingestellt werden, ob eine Plattform zu Beginn des Spiels eingeklappt oder ausgeklappt ist, was den Aufbau der Sprungfolgen ermöglicht.

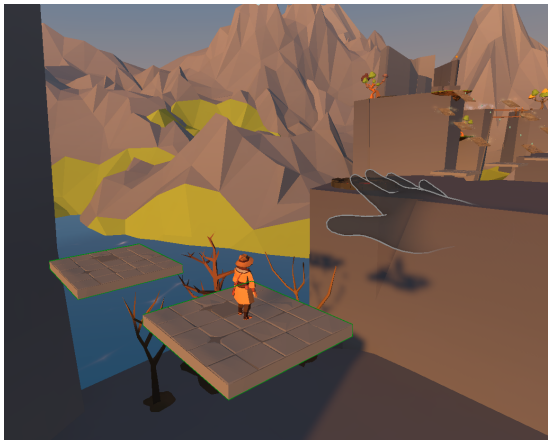
Gesteuert werden die Plattformen über das `FoldablePlatformsPoseManager`-Objekt, welches alle `FoldablePlatform`-Instanzen verwaltet. Auch hier wird über einen Trigger-Collider geprüft, ob sich der Spielcharakter in der Nähe befindet. Nur in diesem Fall werden die über `FoldPose01()` und `FoldPose02()` ausgelösten Events berücksichtigt. Diese werden ebenfalls direkt über den `SelectorUnityEventWrapper` mit den entsprechenden Posen verknüpft.

Die visuelle Rückmeldung erfolgt auch hier über Outlines, deren Farbe sich je nach aktuellem Zustand ändert. Zudem werden passende Soundeffekte bei der Detektion einer Pose abgespielt.

Der Code für die Gesten-Interaktion befindet sich im Verzeichnis `HandinteraktionPosen` auf dem Datenträger (siehe Anhang A.4).

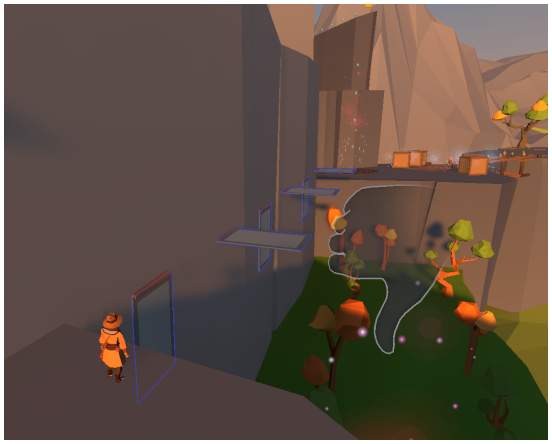


(a) Bewegliche Plattformen mit Stopp-Pose



(b) Bewegliche Plattformen mit Start-Pose.

(c) Klappbare Plattformen mit Pose01.



(d) Klappbare Plattformen mit Pose02.

Abbildung 5.3.: Handinteraktion: Posen (Screenshots aus der Unity Engine).

5.4.5. AR/VR-Übergang (F7)

Der Übergang zwischen AR und VR wird durch ein Portal realisiert. Durch dieses kann der Spieler die virtuelle Welt sehen, wenn er sich im Passthrough-Modus befindet, bzw. die reale Welt sehen, wenn er sich in der virtuellen Umgebung befindet. Des Weiteren ist es möglich, das Portal in beide Richtungen zu durchschreiten und auch Objekte, wie der Charakter, können durch das Portal zwischen den Welten wechseln.

Der Passthrough-Building-Block [60] des Meta-SDKs ermöglicht eine einfache Integration der realen Umgebung durch Video-See-Through-AR.

Erster Ansatz: Ein erster Implementierungsversuch basierte auf einer Render-Texture, die auf ein Portal-Objekt gelegt wurde, bei der die virtuelle Welt durch eine separate Kamera auf die Textur projiziert wird. Diese Lösung ergab jedoch kein überzeugendes dreidimensionales Erlebnis beim Blick durch das Portal.

Finaler Ansatz - Stencil-Maskierung: Stattdessen wurde eine Lösung auf Basis von Stencil Masks gewählt. Diese erlaubt es, die virtuelle Welt nur in bestimmten Bereichen, z.B. innerhalb des Portals, sichtbar zu machen. Technisch wurde die Szene hierfür in zwei Hauptbereiche unterteilt:

- **Die reale Welt** wird durch das Passthrough-System dargestellt.
- **Die virtuelle Welt** befindet sich auf einem separaten Render Layer namens `VirtualWorld`.

In der Universal Render Pipeline (URP) wurde dieser Layer sowohl in der **Opaque Layer Mask** als auch in der **Transparent Layer Mask** deaktiviert. Dadurch wird die virtuelle Welt standardmäßig nicht gerendert. Um sie gezielt sichtbar zu machen, wurden zwei **Render Objects** als **Renderer Features** zur `UniversalRenderData` hinzugefügt:

- **Stencil Mask 1 Opaque** (für undurchsichtige Objekte)
- **Stencil Mask 1 Transparent** (für transparente Objekte)

In beiden Render-Objekten wurde das Stencil-Feature aktiviert. Dieses nutzt den sogenannten **Stencil Buffer**, einen 8-Bit-Buffer, der für jeden Pixel einen ganzzahligen Wert (0-255) speichert.

Die Render-Features wurden so konfiguriert, dass Objekte im Layer `VirtualWorld` nur an den Pixelpositionen gerendert werden, an denen der Stencil-Buffer den Wert 1 enthält. Hierfür wird der Stencil-Wert auf `1` und die Compare-Function auf `Equal` gesetzt.

Das Setzen des Stencil-Werts für die Bereiche, in denen die `VirtualWorld` angezeigt werden soll, erfolgt dann über ein spezielles Material, das auf ein Quad im Bereich des Portals angewendet wird. Dieses Material nutzt einen Shader, der beim Rendern des Portals den definierten Stencil-Wert `1` in den Stencil Buffer schreibt. In Kombination mit den zuvor erwähnten Render-Features wird so sichergestellt, dass die virtuelle Welt nur innerhalb des Portals sichtbar ist.

Rückblick in die reale Welt: Auch aus der virtuellen Welt heraus muss der Blick zurück in die reale Welt möglich sein. Hierfür wurde das durch das Meta-SDK bereitgestellte Passthrough-Window verwendet. Dieses wird auf der Rückseite des Portals angebracht, sodass es aus der Perspektive der virtuellen Welt aussieht, als blicke man zurück in die physische Umgebung.

Das Portal besteht somit aus zwei Quads innerhalb eines Rahmens:

- Ein Quad mit dem **Stencil-Material**, sichtbar aus der „realen Welt“.

- Ein Quad mit dem **Passthrough-Window**, sichtbar aus der „virtuellen Welt“.

Portal-Übergänge: Damit Objekte - insbesondere der Spieler - zwischen den beiden Welten wechseln können, mussten Mechanismen implementiert werden, die beim Durchschreiten des Portals die Render Layer ändern.

Das `PortalLayerSwitcher`-Skript wird Objekten angehängt, die sich durch das Portal bewegen können. Es prüft mittels `Vector3.Dot`, in welche Richtung das Objekt das Portal durchquert und ändert den Layer des Objekts entsprechend von `Default` auf `VirtualWorld` oder umgekehrt. Zusätzlich können beim Durchgang UnityEvents ausgelöst werden.

Das `PortalTransition_Player`-Skript steuert den Wechsel der Welt für den Spieler. Es wird einem Collider-Objekt angehängt, das sich mit dem Center-Eye (also dem HMD) mitbewegt. Beim Durchqueren eines Portals (gekennzeichnet durch das `PortalMarker`-Skript) wird das gesamte Root-Object der virtuellen Welt in den jeweils passenden Layer versetzt.

Der Code und Shader für den AR/VR-Übergang befindet sich im Verzeichnis `MR_Portal` auf dem Datenträger (siehe Anhang A.5).

(a) Kleines Portal aus Intro.

(b) Übergang von Objekten durch das Portal.

(c) Blick in die virtuelle Welt.

(d) Blick in die reale Welt.

Abbildung 5.4.: AR/VR-Übergang (Screenshots von der Meta Quest 3).

5.4.6. Ressourcen (F9, F10)

Neben den sammelbaren Objekten im Spiel zählen auch die Leben des Charakters und das Punktesystem zu den Ressourcen.

Aufbau und Komponenten: Im Spiel gibt es drei Arten sammelbarer Ressourcen: Kristalle, Leben und Collectibles. Diese Ressourcen sind eng mit dem Respawn-System, dem Punktesystem sowie der Level-Fortschrittskontrolle verknüpft. Die Speicherung erfolgt separat und wird im Kapitel *Speicherfunktion* behandelt. Das Ressourcen-System ist aus folgenden Modulen aufgebaut:

- **Kristalle**: Funktionieren über ein „Coin“-System (ursprünglich waren Münzen statt Kristalle vorgesehen). Besteht aus dem `Coin`-Skript sowie dem `CoinManager`-Singleton.

- **Leben**: Repräsentieren die verbleibenden Versuche und sind an die Respawn-Vorgänge gekoppelt. Zentral hierfür ist das `LifeManager`-Singleton.
- **Respawn-System**: Stellt sicher, dass der Charakter bei Absturz wieder am letzten Checkpoint erscheint - gegebenenfalls wird ein Leben abgezogen. Besteht aus dem `Char_Respawn`- und dem `Checkpoint`-Skript.
- **Level_Controller** (Skript): Zentrale Komponente zur Verwaltung des Fortschritts im Level. Schnittstelle zu UI-Elementen über das Interface `ILevelController`.

Kristalle mit Coin-System: Das `Coin`-Skript ist an die Kristall-Objekte gebunden und steuert das Einsammeln dieser. Betritt der Charakter einen Trigger-Radius um den Kristall, wird der Kristall durch die `MoveToCharacter()`-Methode „magnetisch“ zum Charakter gezogen, inklusive Animation und Soundeffekt. Nach erfolgreichem Einsammeln ruft `Coin` den Singleton `CoinManager` auf, um die globale Kristallzahl zu erhöhen. Zusätzlich wird der Score über den `Level_Controller` gesteigert.

Der `CoinManager` überwacht die Gesamtzahl der Kristalle. Beim Erreichen jedes Vielfachen von 50 ruft er `LifeManager.AddLife()` auf. Die gesammelten Kristalle können über `SaveCoins()` dauerhaft gespeichert werden. Dies sollte beim Szenenwechsel - konkret beim erfolgreichen Beenden eines Levels - aufgerufen werden.

Lebenssystem und Respawn: Das Lebenssystem wird vom Singleton-Skript `LifeManager` verwaltet. Es speichert die aktuelle Lebensanzahl, zeigt Animationen beim Lebensgewinn und verwaltet das Game Over beim Verlust aller Leben.

Das Respawn-System (`Char_Respawn`) prüft in `Update()`, ob sich der Charakter unterhalb eines bestimmten Schwellenwerts befindet. Im Fall eines Sturzes in den Abgrund wird der Spieler an die zuletzt gespeicherte Position (Checkpoint) zurückgesetzt. Optional kann bei einem Respawn ein Leben abgezogen werden. Der Checkpoint selbst wird durch das Skript `Checkpoint` gesetzt, wenn der Charakter mit dem jeweiligen Trigger kollidiert.

Collectibles: Jedes Collectible ist mit dem Skript `Collectible` ausgestattet. Beim Einsammeln wird ein Audioeffekt und eine Animation abgespielt. Außerdem wird die Methode `Level_Controller.CollectibleGathererd()` über ein UnityEvent aufgerufen, damit das jeweilige Collectible (1-3) als eingesammelt gilt. Zusätzlich werden 1000 Punkte gutgeschrieben.

Levelfortschritt und Punktesystem: Die zentrale Klasse `Level_Controller` kontrolliert den Level-Verlauf. Sie startet und beendet das Level, zählt die Spielzeit (`currentTime`), erfasst alle gesammelten Ressourcen, berechnet den Punktestand (unter Abzug einer Zeitstrafe) und ruft bei Abschluss `SaveLevelProgress()` auf. Letzteres speichert den Fortschritt persistent und aktualisiert z.B. den Highscore.

Die Schnittstelle `ILevelController` erlaubt die einfache Anbindung der Level-Daten an UI-Komponenten.

Der Code für Sammelobjekte und Lebenssystem befindet sich im Verzeichnis `Ressourcen` auf dem Datenträger (siehe Anhang A.6).

(a) Sammelbare Kristalle weisen den Weg.

(b) Collectible unter klappbarer Plattform.

(c) Der Charakter erhält ein neues Leben.

Abbildung 5.5.: Ressourcen (Screenshots aus der Unity Engine).

5.4.7. Speicherfunktion (F8)

Das Spiel verwendet ein automatisches, dreiteiliges System, das Spielfortschritt, Leveldaten und Nutzereinstellungen persistent speichert. Ein manuelles Speichern durch den Spieler ist nicht erforderlich.

1. **GameData_Controller - Fortschritts- und Ressourcenverwaltung:**

   Der `GameData_Controller` übernimmt das Speichern des Spielfortschritts und der Ressourcen mittels Unitys `PlayerPrefs`. Gespeichert wird dabei, wie viele Fokus-Steine gesammelt und eingesetzt wurden (zur Freischaltung neuer Level), die aktuelle Anzahl an Leben sowie, gesammelte Kristalle.

   Die Klasse stellt Methoden zu Verfügung, mit denen andere Komponenten (z.B. `LifeManager` oder `CoinManager`) Daten speichern und auslesen können. Beim Start eines neuen Spiels können sämtliche gespeicherte Fortschritte mit `DeleteAllPlayerPrefs()` zurückgesetzt werden.

2. **LevelDataManager - Verwaltung von Leveldaten:**

   Der `LevelDataManager` ist eine Singleton-Instanz, die beim Spielstart geladen wird und sämtliche Level-spezifischen Daten verwaltet: Highscore, Status gesammelter Collectibles (bis zu drei pro Level), Schnellste absolvierte Zeit.

   Die Daten werden zur Laufzeit in einem Dictionary gespeichert und beim Beenden der Applikation bzw. beim Pausieren automatisch in einer JSON-Datei gespeichert. Die Daten können so auch über die UI (z.B. in der Levelauswahl) dargestellt werden. Es wird außerdem sichergestellt, dass einmal gesammelte Collectibles nicht wieder verloren gehen, und dass neue Bestzeiten oder höhere Punktzahlen korrekt aktualisiert werden.

3. **SettingsManager - Speicherung von Nutzereinstellungen:**

   Der `SettingsManager` dient der persistenten Speicherung nutzerspezifischer Einstellungen, wie Kameramodus, Komfortoptionen oder die Aktivierung des Testmodus, mit unbegrenzten Leben.

   Auch der `SettingsManager` ist als Singleton implementiert und speichert seine Daten in einer JSON-Datei auf dem Gerät. Änderungen an den Einstellungen werden direkt gespeichert, sodass die Präferenzen des Nutzers über alle Spielsitzungen hinweg erhalten bleiben.

Der Code für das Speichersystem befindet sich im Verzeichnis `Speicherfunktion` auf dem Datenträger (siehe Anhang A.7).

5.4.8. 2D-Benutzeroberflächen (F5, F6)

Das für das Projekt verwendete 2D-UI-System basiert auf dem UI-Set, das durch das Meta XR SDK bereitgestellt wird. Dies ermöglicht das einfache Zusammensetzen interaktiver UI-Elemente, die speziell für den Einsatz in XR-Umgebungen ausgelegt sind. [61]

Verschiedene UI-Komponenten bilden unterschiedliche Interaktionen und Informationen ab. Neben klassischen Menüs (z.B. das Hauptmenü) enthält das System auch kontextabhängig aktivierbare Interfaces, die aktuelle Informationen und Navigationsmöglichkeiten bereitstellen.

Komfort-Skripte: Zwei Skripte wurden entwickelt, um den allgemeinen Komfort bei der Darstellung und Interaktion mit UI-Elementen im Raum zu erhöhen:

- **SmoothUIFollow**: Kann an ein Canvas angehängt werden, damit dieses dem Blickfeld des Spielers folgt. Die UI wird neu positioniert, wenn ein bestimmter Abstand oder ein Schwellwert für den Blickwinkel überschritten wird.
- **GameObjectBillboarding**: Durch dieses Skript wird das GameObject (meist UI-Elemente) permanent in Richtung des Spielers ausgerichtet. Die Rotation erfolgt ebenfalls weich, um abrupte Bewegungen zu vermeiden. Dieses Verhalten unterstützt die Lesbarkeit z.B. bei im Raum platzierten Tooltips.

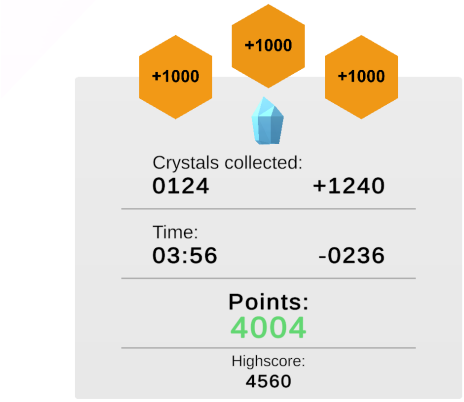
UI-Spezifische Skripte: Die folgenden Skripte steuern jeweils spezifische UIs, die in bestimmten Spielsituationen eingeblendet werden:

- **CheckpointUI**: Wird eingeblendet, wenn der Spielcharakter einen Checkpoint betritt. Die UI zeigt dabei die verbleibenden Leben direkt über dem Checkpoint an.
- **ControllerUI**: Stellt während eines Levels Informationen wie Lebensanzahl, eingesammelte Kristalle, Spielzeit und Punktestand dar. Die Daten stammen aus dem `ILevelController`. Die UI kann über das Gedrückthalten des sekundären Buttons am Controller aktiviert werden.
- **LevelFinishedUI**: Diese UI erscheint nach Abschluss eines Levels und zeigt die Ergebnisse an. Hierfür wird ebenfalls der `ILevelController` sowie der `LevelDataManger` verwendet.
- **LevelSelectionUI**: Kommt im Level-Auswahlbildschirm zum Einsatz und informiert über bereits erspielte Highscores, scshnellste Levelzeiten sowie eingesammelte Collectibles pro Level. Die Daten werden über den `LevelDataManager`, mit der jeweiligen `levelId` abgefragt.

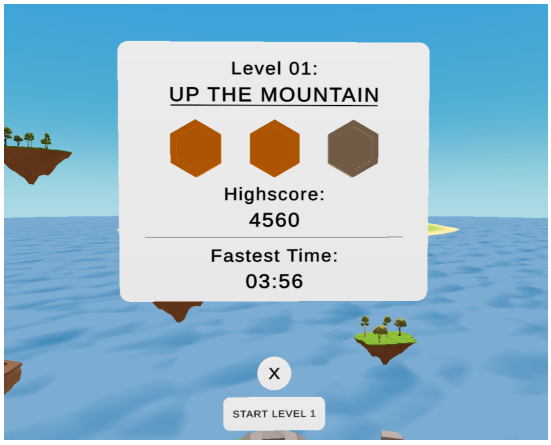
- **MainMenuContinueButton**: Dieses Skript überprüft beim Laden des Hauptmenüs, ob das Intro bereits abgeschlossen wurde. Ist dies der Fall, wird der „Continue"-Button aktiviert.
- **MenuUI**: Wird zusammen mit der `ControllerUI` eingeblendet und bietet weiterführende Buttons, mit denen je nach Spielsituation beispielsweise ein Level neu gestartet oder ins Hauptmenü zurückgekehrt werden kann.
- **RecenterUI**: In bestimmten Spielsituationen ist es erforderlich, das XR-Rig neu zu zentrieren (z.B. nach dem Szenenwechsel). Da das Meta SDK kein entsprechendes Event bereitstellt, überprüft dieses Skript die Differenz zwischen dem `centerEye`- und dem `cameraRig`-Transform hinsichtlich Position und Rotation. Sobald diese innerhalb der vordefinierten Schwellenwerte liegen, wird die UI automatisch deaktiviert und ein Event (z.B. zum Starten des Levels über den `Level_Controller`) ausgelöst.
- **SettingsUI**: Verarbeitet die Interaktionen in den Einstellungen und übergibt geänderte Werte an den `SettingsManager`. Die Einstellungen umfassen die verschiedenen Kamera- und Komfortmodi. Für bessere Verständlichkeit werden zugehörige Videos in die UI integriert und abhängig von den Einstellungen automatisch gewechselt und abgespielt.

Der Code für 2D-Benutzeroberflächen befindet sich im Verzeichnis `UI` auf dem Datenträger (siehe Anhang A.8).

(a) Das Hauptmenü des Spiels.



(b) UI mit Ergebnissen bei Level-Abschluss.



(c) UI zum Starten eines Levels.

Abbildung 5.6.: Benutzeroberflächen-Beispiele (Screenshots aus der Unity Engine).

5.4.9. Animationssystem (F12)

Das Animationssystem des Projekts basiert auf dem **Unity Animator Controller** und verwendet eine Kombination aus klassischen State Machines, parametergesteuerten Zustandsübergängen sowie ereignisbasierten Triggern.

Zahlreiche interaktive Elemente werden durch eigene Animationen visuell unterstützt. Beispielsweise skalieren sich UI-Elemente dynamisch beim Öffnen und Schließen, wodurch ein flüssiger Aufbau bzw. Rückzug der Benutzeroberfläche entsteht. Auch Spielobjekte wie Fokussteine oder sammelbare Collectibles verfügen über eigene Idle-Schwebeanimationen sowie eine zusätzliche Animation, die beim Einsammeln getriggert wird. Die Zustandswechsel der Animationen werden meist durch Skripte ausgelöst, etwa mittels `Animator.SetBool()`, um gezielt bestimmte Übergänge zu aktivieren.

Zusätzlich zu den parameter-basierten Zustandswechseln werden **Animation Events** genutzt, um gezielt Logik während einer Animation auszulösen. So können beispielsweise

über das `AnimationSoundTrigger`-Skript Sounds an bestimmten Stellen einer Animation getriggert werden. Dies wird unter anderem für die Schrittgeräusche beim Laufen des Charakters verwendet.

Für die Intro-Sequenz des Spiels, in der eine Reihe von Animationen und Soundeffekten in fester Reihenfolge abläuft, wird Unitys **Timeline-System** eingesetzt. Mit dieser Funktion lassen sich Cutscenes oder Gameplay-Sequenzen als Abfolgen auf einem Zeitstrahl anordnen. Dabei können verschiedene Spuren für Animationen, Audio, Signale und Skriptelemente verwendet werden, um komplexe Szenenabläufe zu realisieren. Da die standardmäßigen Audio-Tracks der Timeline jedoch fehlerhaft funktionierten (z.B. fehlerhaftes Laden oder verzerrte Wiedergabe von Audiodateien), wurde auf eine alternative Lösung gesetzt: Ein Signal-Track sendet über sogenannte Signal-Emitter gezielte Ereignisse an ein eigenes Skript namens `TimelineAudioTrigger`, welches die synchronisierte Wiedergabe von AudioClips übernimmt.

Die Sprung-, Dash und Bewegungsanimationen des Charakters wurden über Adobes Mixamo heruntergeladen. Mixamo bietet eine umfangreiche Bibliothek an 3D-Charakteranimationen, die sich direkt auf Unity-kompatible Rig-Modelle anwenden lassen [62].

Der Code für das Animationssystem befindet sich im Verzeichnis `Animationssystem` auf dem USB-Stick (siehe Anhang A.9).

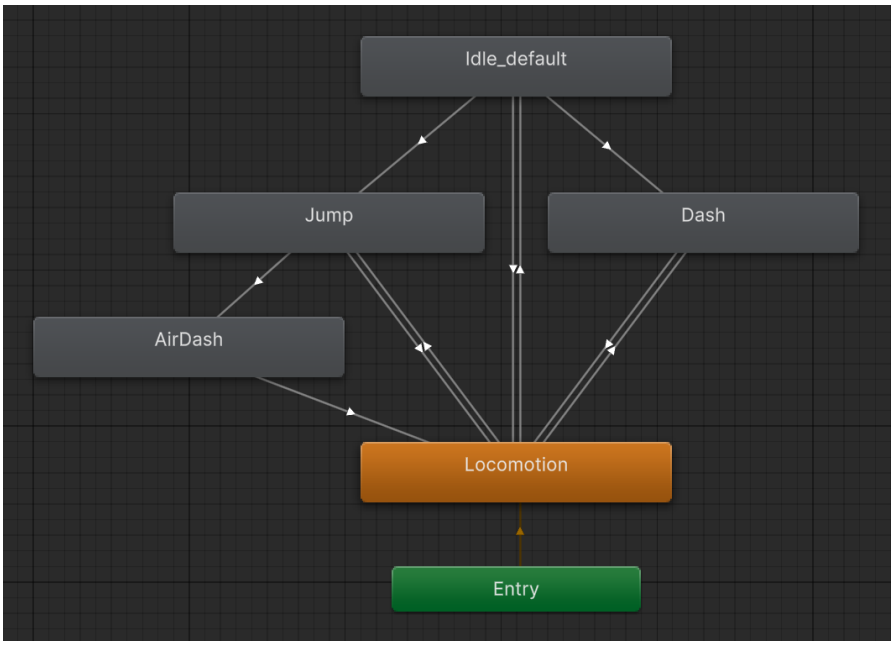


Abbildung 5.7.: Aufbau des Charakter-Animators für Bewegungs-Animationen (Screenshot aus der Unity-Engine).

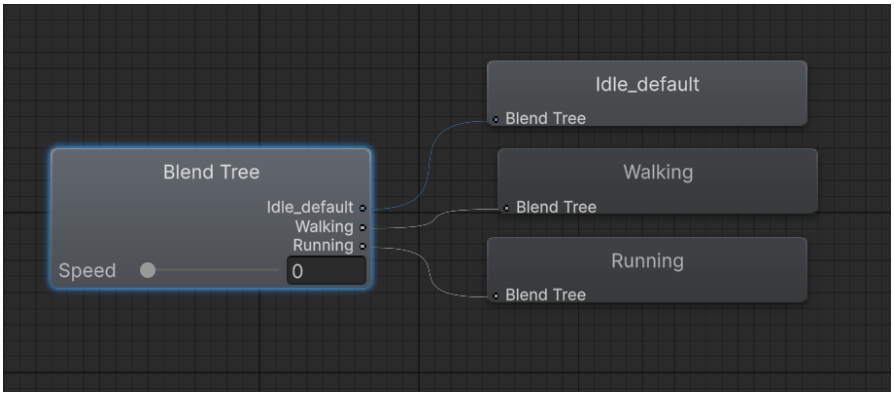


Abbildung 5.8.: Laufanimation (Locomotion-Node) wird gesteuert über Blend-Tree (Screenshot aus der Unity-Engine).

5.4.10. Dialogsystem (F11)

Zur Verbesserung der Immersion und Nutzerführung wurde ein eigenes Dialogsystem implementiert, über das der Spielcharakter direkt mit dem Spieler kommuniziert. Die Dialoge sind vollständig vertont und werden durch Motion-Capture-gestützte Körperanimationen begleitet. Sie tragen nicht nur zur erzählerischen Tiefe bei, sondern unterstützen den Spieler durch Hinweise zu Spielmechaniken und Interaktionen.

Aufnahme und Verarbeitung der Motion-Capture-Daten: Für die Aufnahme der Motion-Capture-Animationen wurde das Programm **Animotive** [50] in Verbindung mit der Meta Quest 3 eingesetzt. Die Body-Tracking-Funktionalität des HMDs ermöglicht ohne externe Basistationen jedoch nur eine limitierte Aufzeichnung des Oberkörpers. Das im Spiel verwendete Charaktermodell wurde im FBX-Format in Animotive importiert, um eine direkte Visualisierung während der Aufnahme zu ermöglichen.

Dialogskripte können im Vorfeld als Textdateien in die Software geladen und über einen integrierten Prompter abgelesen werden. Dies ermöglicht eine zeitgleiche Aufnahme von Körperbewegung und Voiceover, wodurch Animation und Sprache synchronisiert sind.

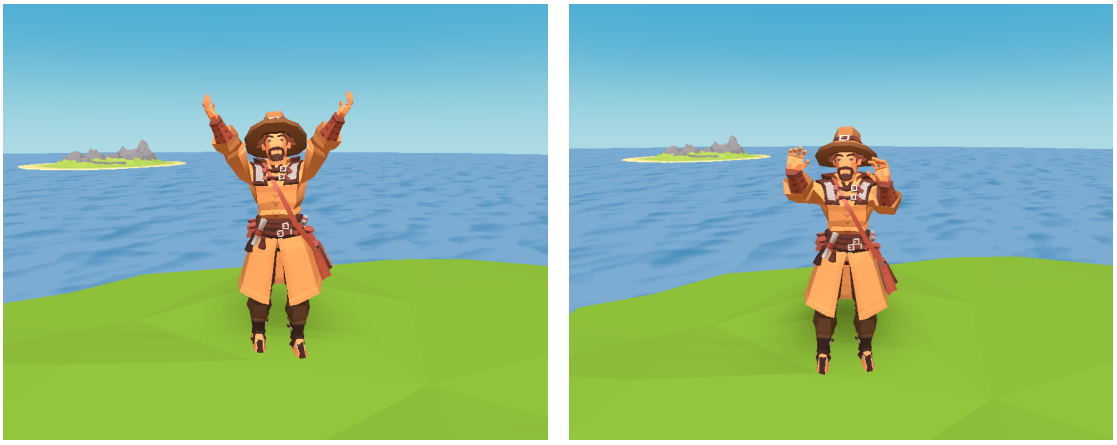
Nach der Aufnahme können die Animationen über die Exportfunktion von Animotive in ein für Unity optimiertes Format umgewandelt werden. Mithilfe des **Animotive Importer Plugins** [63] lassen sich die aufgenommenen Clips schnell in das Projekt integrieren und anschließend dem Animator-Controller des Charakters zuweisen.

Integration in den Animator: Um eine Kombination aus Motion-Capture-Animationen und gleichzeitiger Bewegungsanimation des Charakters zu ermöglichen, werden die Dialoganimationen auf einem separaten **Animator-Layer** platziert. Eine konfigurierte **Avatar Mask** sorgt dann dafür, dass ausschließlich der Oberkörper, also Kopf, Arme und

Rumpf, von den Motion-Capture-Clips beeinflusst wird. Dadurch bleiben gleichzeitig die Lauf- und Bewegungsanimationen der unteren Körperhälfte erhalten.

Voiceover-Erstellung und Nachbearbeitung: Die Sprachaufnahmen werden im Anschluss an die Motion-Capture-Aufnahmen mit dem KI-gestützten **Voice-Changer** von **ElevenLabs** [51] überarbeitet. Dadurch konnte eine glaubwürdigere Charakterstimme generiert werden. Die Audioclips werden automatisiert in die Zielstimme umgewandelt, wobei jedoch teils Störgeräusche und Artefakte oder Übersteuerungen entstanden. Diese wurden manuell mit **Audacity** bereinigt, um ein besseres Klangbild zu erzeugen.

Die finalen Audiodateien wurden ebenfalls über **Animation Events** und den `AnimationSoundTrigger` in die entsprechenden Animationsclips eingebettet.



(a) Ausschnitt 01 einer Motion-Capture Animation.

(b) Ausschnitt 02 einer Motion-Capture Animation.

Abbildung 5.9.: Ausschnitte von Motion-Capture Animationen (Screenshots aus der Unity-Engine).

5.4.11. Levelarchitektur und Weltstruktur (F13, F14)

Die Anwendung ist, wie bereits beschrieben, in Szenen unterteilt, die jeweils ein einzelnes Level oder einen übergeordneten Bereich (z.B. Hauptmenü, Intro, Levelauswahl) repräsentieren. Diese Abschnitte basieren auf einer modularen Struktur und vereinen sämtliche zuvor erläuterten Funktionalitäten zu einem Spielerlebnis.

Aufbau der Spielwelt: Das zentrale Element jeder Szene ist das XR-Rig, welches die Spielerposition sowie Eingaben per Hand- und Controllertracking zusammen mit dem Input-System von Unity verarbeitet. Um das Rig herum ist der Spielbereich aufgebaut, der aus mehreren Ebenen besteht:

Hintergrundkomponenten schaffen eine visuelle, atmosphärische Kulisse und sind aus Elementen wie der Skydome, animierten Wolken, Wasserflächen und Gebirgszügen aufgebaut.

Der Spielbereich im Vordergrund ist in mehrere Sektionen untergliedert, die unterschiedliche Aufgaben und Herausforderungen bieten. In jeder Sektion sind Elemente wie Sammelobjekte, Checkpoints sowie Hindernisse und verschiedene Spielmechaniken implementiert.

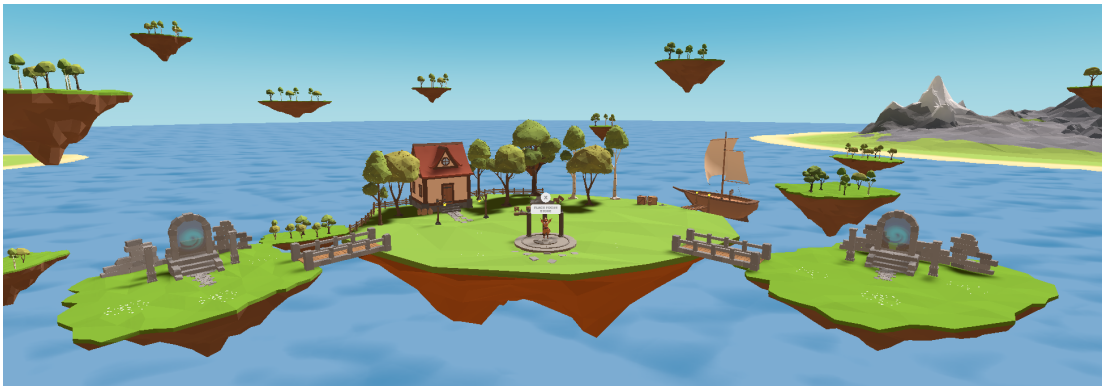
Das Leveldesign enthält verschiedene Triggerpunkte, die unterschiedliche Funktionen aktivieren. Diese beinhalten unter anderem die Trigger-Zonen zum Wechsel der Kamera im Komfort-Modus, Dialog-Trigger oder Tooltip-Trigger. Sie sind über die Level verteilt und sorgen für eine intuitive Progression durch die Spielwelt.

Am Ende jedes Levels befindet sich eine spezielle Sektion, in der der Fokusstein platziert ist. Sobald der Charakter diesen einsammelt, wird die Level-Finished-UI eingeblendet, welche die Ergebnisse (Higscore, gesammelte Ressourcen, etc.) anzeigt. Gleichzeitig wird die Bewegung des Spielcharakters so eingeschränkt, dass der Bereich nicht mehr verlassen werden kann. Dies verhindert, dass nach Abschluss des Levels weitere Objekte eingesammelt, Mechaniken erneut ausgelöst oder Leben verloren werden.

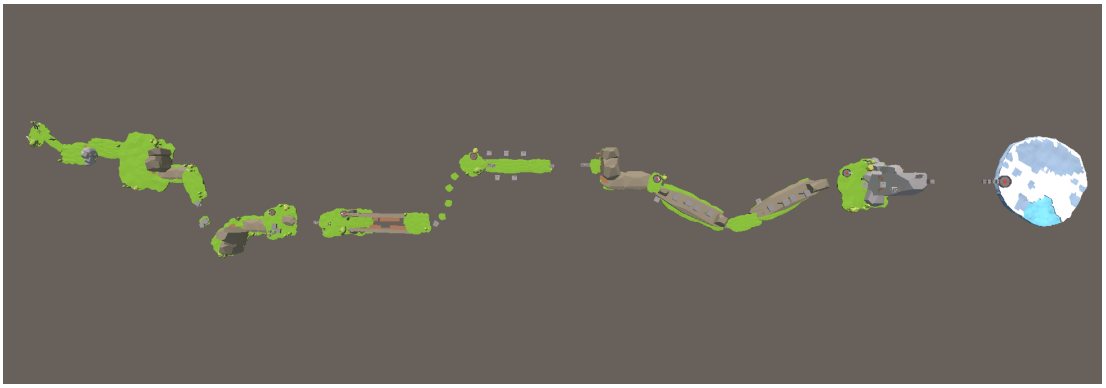
Zur Rückkehr in die Levelauswahl muss der Spieler einen letzten Interaktionspunkt - dargestellt durch ein Portal - aktivieren. So gelangt er zurück zur Levelauswahl.

Für jede Szene wurden außerdem individuelle Ambient-Sounds und Hintergrundmusik eingefügt, um die Stimmung der Abschnitte zu unterstreichen.

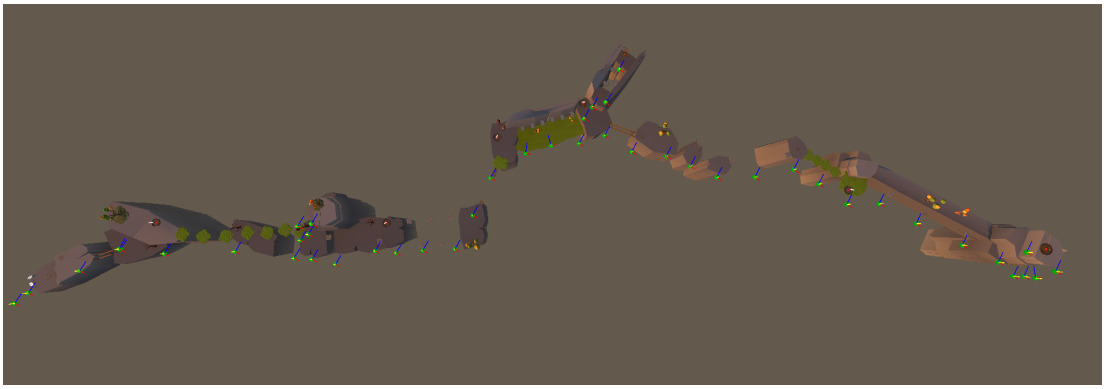
Systemelemente und technische Objekte: Neben den sichtbaren Gameplay-Elementen sind in jeder Szene verschiedene systemrelevante Objekte integriert. Diese übernehmen technische Funktionen wie die Logik zur Levelprogression, Datenmanagement und Systemkonfigurationen (z.B. Renderqualität und Performance-Optimierungen).



(a) Levelauswahl-Szene.



(b) Level 01 aus der Vogelperspektive, ohne Hintergrundelemente.



(c) Level 02 aus der Vogelperspektive, ohne Hintergrundelemente, mit sichtbaren Kamera-Diorama-Punkten.

Abbildung 5.10.: Spielwelt-Beispiele (Screenshots aus der Unity-Engine).

5.5. UX-Design unter Berücksichtigung von Design-Heuristiken

In diesem Kapitel wird konkret zusammengefasst, in welcher Form die in *Kapitel 3* dargestellten Design-Heuristiken und Guidelines in der Entwicklung berücksichtigt wurden.

Dabei handelt es sich um erste Umsetzungsversuche, die bisher noch nicht im Rahmen der explorativen Nutzertests evaluiert wurden. Eine Bewertung der tatsächlichen Wirksamkeit dieser Maßnahmen erfolgt in den Kapiteln *Evaluierung des Prototyps* und *Diskussion*.

5.5.1. Übereinstimmung zwischen System und realer Welt (NF1a)

Es wurde versucht, zentrale Interaktionen möglichst intuitiv und an reale Handlungen angelehnt umzusetzen. Beispielsweise orientiert sich das Greifen virtueller Objekte am natürlichen Greifverhalten: Es genügt, die virtuelle Hand um ein Objekt zu schließen, ein direkter Fingerkontakt ist nicht notwendig.

Auch die physikbasierte Charaktersteuerung soll ein natürliches Verhalten von Bewegungsabläufen nachahmen. Die Bewegung des Spielers wird direkt durch reale Körperbewegungen ausgelöst und entspricht dadurch der tatsächlichen Bewegung in der physischen Welt. Dies erleichtert auch das Verständnis des Übergangs zwischen AR und VR, da dieser durch das tatsächliche Durchschreiten des Portals erfolgt.

Animationen und das durch Motion-Capture unterstützte Dialogsystem lassen die Welt zusätzlich nachvollziehbarer wirken. Die erzählerischen Elemente unterstützen außerdem das Verständnis für den kleinen Maßstab der Spielwelt und des Charakters.

Die Verwendung von 2D-Benutzeroberflächen (wie z.B. das Hauptmenü) ist der Nutzung von UI auf Touchscreen-Geräten nachempfunden, was Vertrautheit schaffen soll.

5.5.2. Wahrung von Konsistenz und Einhaltung von Standards (NF1b/NF4)

Um die Konsistenz im Nutzererlebnis zu unterstützen, wurde auf Standardkonventionen des Meta-Ökosystems zurückgegriffen [17]. Durch das Meta XR SDK entsprechen grundlegende Interaktionen weitgehend denen, die Nutzer von anderen Anwendungen auf der Quest kennen. So wurde beispielsweise darauf geachtet, die Button-Belegung am Controller so zu nutzen, wie es von Meta empfohlen wird [9].

Auch innerhalb der Anwendung wurde auf interne Konsistenz geachtet: Die Steuerung bleibt in allen Spielabschnitten gleich, die Interaktionen sind durchgängig einheitlich gestaltet. Zusätzlich folgt das Spiel einem einheitlichen Design in einer konsistenten Spielwelt.

5.5.3. Physische Sicherheit und Fehlervermeidung (NF1c)

Es wurde versucht, potenzielle physische Risiken beim Spielen zu minimieren: Das Spiel nutzt das native Guardian-System der Meta Quest, um Kollisionen mit realen Objekten zu vermeiden.

Zudem erfordert das Spiel keine gefährlichen oder anstrengenden Bewegungen. Schnelles Rennen, Springen oder Bücken sind nicht notwendig. Beim Start wird der Spieler aktiv aufgefordert, einen sicheren Spielbereich mit mindestens zwei Metern Freiraum nach vorne bereitzustellen und gegebenenfalls die Begrenzung anzupassen.

5.5.4. Flexibilität und Anpassbarkeit (NF1d)

Der Spieler hat die Möglichkeit, die Anwendung an eigene Präferenzen anzupassen, um ein komfortables Erlebnis zu ermöglichen.

Die Steuerung kann jederzeit zwischen linker und rechter Hand gewechselt werden. Um Symptome der VR-Krankheit zu vermeiden, können die unterschiedlichen Kameramodi gewählt werden. Eine optionale Komfort-Vignette oder das Dash-Movement können aktiviert werden. In Bereichen, in denen der Spieler nicht an den Charakter gebunden ist, wie dem Intro oder der Levelauswahl, wird die Spielwelt automatisch an die Körpergröße des Nutzers angepasst. Gleiches gilt auch für UI-Elemente im World-Space, wie dem Hauptmenü.

Die Spielumgebung ist so gestaltet, dass wichtige Interaktionen im unmittelbaren Nahbereich stattfinden und die Anzahl gleichzeitiger Interaktionsmöglichkeiten überschaubar bleibt, um Überforderung zu vermeiden.

5.5.5. Feedback und Hinweise (NF1e)

Die Anwendung gibt dem Nutzer an möglichst vielen Stellen Rückmeldung über sein Handeln: Interaktionen werden mit visuellen Effekten sowie mit Soundeffekten begleitet, um deren Erfolg oder Misserfolg eindeutig zu kommunizieren.

Beispielsweise zeigen greifbare Objekte eine Outline an, wenn sich die Hand des Spielers in der Nähe befindet. Bei erfolgreichem Greifen verändert die Outline ihre Farbe. Des Weiteren wird durch unterschiedlich farbige Partikeleffekte signalisiert, ob es sich um eine direkte, indirekte oder Hover-Interaktion handelt. Akustisches Feedback erfolgt zusätzlich beim Greifen, Loslassen oder bei der Erkennung von Handgesten. Auch das Sammeln von Gegenständen wird durch visuelle und auditive Signale unterstützt.

Zusätzlich erscheinen Tooltips beim erstmaligen Auftreten neuer Interaktionsmöglichkeiten. Die Charakter-Dialoge ergänzen dies durch Hinweise, die als Teil des Storytellings vermittelt werden.

5.5.6. Förderung der Immersion (NF1f)

Die Spielwelt wurde mit dem Ziel entworfen, eine atmosphärische Spielerfahrung zu ermöglichen. Visuelle Effekte, Musik, räumlicher Klang (Spatial Audio) sowie die Erzählerischen Aspekte sollen die Immersion verstärken und die Wahrnehmung der virtuellen Umgebung glaubwürdiger gestalten.

5.5.7. Vertrauen und exploratives Verhalten fördern (NF1f)

Ein wichtiges Ziel war es, dem Spieler durch nachvollziehbares Verhalten der Anwendung ein Gefühl von Kontrolle zu vermitteln. Dies soll etwa durch eine konsistente Navigation, das Fehlen unerwarteter Zustände sowie durch eine verlässliche Systemreaktion unterstützt werden.

Gleichzeitig wurde versucht, das Spiel so zu gestalten, dass experimentierfreudiges Verhalten belohnt wird, ohne dass der Spieler irreversible Fehler befürchten muss:

Selbst wenn der Nutzer alle Leben verliert, gibt es kein endgültiges Game-Over. Verlorene oder geworfene Objekte kehren außerdem automatisch an ihren Ausgangspunkt zurück. Durch die Checkpoints müssen im Zweifel nur kleinere Level-Passagen wiederholt werden. Sollte der Spieler trotzdem irgendwo festhängen, kann ein Level jederzeit neu gestartet werden.

Teilweise gibt es auch Passagen im Spiel, die gespielt werden können, ohne den vorhergesehenen Weg zu gehen. Diese wurden bewusst zugänglich gelassen, um den Spieler zu motivieren, mit alternativen Lösungswegen zu experimentieren.

5.6. Probleme und Herausforderungen

5.6.1. Performance und Bildqualität (NF2)

Insbesondere im XR-Bereich ist eine stabile und hohe Bildrate essenziell für ein angenehmes Nutzungserlebnis. Zu niedrige Framerates können zu einer Beeinträchtigung der Immersion und zu Symptomen der VR-Krankheit beitragen [22]. Laut den Richtlinien von Meta müssen XR-Anwendungen mindestens 72 Bilder pro Sekunde (FPS) erreichen, um eine akzeptable Nutzererfahrung zu gewährleisten [64].

Während der Entwicklungsphase erfolgten erste Tests hauptsächlich über den Unity-Editor, wobei das HMD über ein Kabel an den Entwicklungsrechner angeschlossen war. Hierbei übernimmt der PC sämtliche Berechnungen, während das HMD lediglich zur Bildausgabe dient. Um die tatsächliche Performance der Anwendung unter realen Bedingungen zu beurteilen, war es daher notwendig, regelmäßig native Builds für die Meta Quest 3 zu erstellen. Die Performanceanalyse erfolgte dabei mithilfe des offiziellen **OVR Metrics Tools** von Meta [65].

Ein erster Build, welcher lediglich die Intro- und Levelauswahl-Szene beinhaltete, offenbarte starke Performanceprobleme: Die durchschnittliche Bildrate betrug lediglich 25,02 FPS. Zudem war das Bild extrem niedrig aufgelöst, da das Dynamic Resolution Feature des Meta XR SDKs versuchte, die niedrige Framerate durch Reduzierung der Auflösung auszugleichen. Dies wirkte sich negativ auf die visuelle Qualität aus. Auch die Schattenqualität war sehr schlecht, was vermutlich durch die sehr klein skalierte Spielwelt bedingt war - da dieses Problem vor allem bei den klein skalierten Objekten auftrat.

5.6.1.1. Lösungsansätze zur Performance-Optimierung:

Die über das OVR Metrics Tool gewonnenen Daten zeigten eine Überlastung der GPU als Hauptursache für die niedrige Bildrate. Ziel war es daher, durch Optimierungen eine stabile Performance bei gleichzeitig akzeptabler visueller Qualität zu erreichen.

Projektkonfiguration und Rendering: Zunächst wurden die von Meta empfohlenen Projektkonfigurationen übernommen [66]: HDR und Post-Processing wurden deaktiviert, MSAA (Kantenglättung) wurde auf 4x reduziert und der empfohlene Rendering Path (Forward) wurde in den URP-Einstellungen gewählt. Außerdem wurde dynamisches `Foveated Rendering` über ein Skript (`FoveatedRenderingManager`) aktiviert. Dadurch werden die Ränder der erzeugten Frames in einer niedrigeren Auflösung gerendert, als die Mitte des Bildes. Dies ist für den Nutzer kaum wahrnehmbar, spart jedoch signifikant GPU-Leistung [67].

Des weiteren wurde das `Dynamic Resolition` Feature deaktiviert, da es visuelle Probleme (u.a. im MR-Portal) verursachte. Die Render-Scale wurde auf einen festen Wert von 1.0 gesetzt, was sich als guter Kompromiss zwsichen Performance und Bildqualität erwies.

Ein zusätzliches Bottleneck stellte das ursprünglich verwendete Package für die visuellen Outlines dar. Dieses verursachte einen deutlichen Leistungsverlust, weshalb ein Wechsel auf das Quick-Outline-Package erfolgte [55], was zu einer spürbaren Verbesserung der Framerate führte.

Darüber hinaus wurden mehrere Shader, die ursprünglich nicht für mobile Plattformen optimiert waren, durch vereinfachte Mobile-Shader ersetzt oder entsprechend angepasst. Auch dies hatte einen positiven Effekt auf die GPU-Auslastung.

Asset-Optimierung und Texturen: Im Vorfeld wurden bereits bewusst Low-Poly-Assets gewählt. Zusätzlich wurden nun die LOD-Groups (Level of Detail) angepasst, sodass weiter entfernte Objekte schneller vereinfacht oder vollständig ausgeblendet werden. Dabei wurde darauf geachtet, dass keine sichtbaren Pop-ins im Nahbereich entstehen. 3D-Modelle, die keine LOD-Groups unterstützen, wurden so angepasst, dass sie bei Erreichen eines bestimmten Abstands trotzdem ausgeblendet werden und einige Mesh-Collider wurden durch simplere Collider-Kombinationen ersetzt.

Da die Spielwelt sehr klein skaliert ist, konnten außerdem die Texturgrößen reduziert werden. Hierzu wurde ein eigenes Unity Editor Tool entwickelt: `MiniatureTexture-Optimizer`. Dieses Skript analysiert alle Object-Renderer einer Szene und reduziert automatisch die Texturauflösung bei besonders kleinen Objekten, wodurch dies nicht manuell für jedes Objekt gemacht werden musste.

Beleuchtung, Lichtberechnung und Schatten: Die ursprünglich verwendete Realtime-Beleuchtung stellte sich als weitere Ursache für die schlechte Performance heraus. Daher wurde es durch eine Kombination aus `Baked Lighting`, `Light Probe Groups` und `Reflection Probes` ersetzt:

Statische Objekte wurden für die Lichtberechnung vorgerendert (Lightmaps) [68] und zusätzlich für `Static Batching` markiert. Letzteres ermöglicht beim Rendern das Zusammenfassen mehrerer Objekte mit gleichem Material zu einem einzigen Mesh, wodurch die Anzahl der Draw Calls reduziert werden kann [69].

Dynamische Objekte nutzen `Light Probes`, die vorberechnete Lichtinformationen liefern [70]. Hierfür müssen die Probes in der Szene platziert, Lichtquellen auf „`Light Mode: Mixed`“ gestellt und die Beleuchtung über die Lighting-Einstellungen „gebaked“ werden.

Zur Verbesserung der Schattenqualität bei gleichzeitig guter Performance wurde das URP-Asset angepasst. Es wurden präzisere Einstellungen u.a. für Depth-Bias, Normal-Bias, Shadow-Distance oder Cascade-Count vorgenommen.

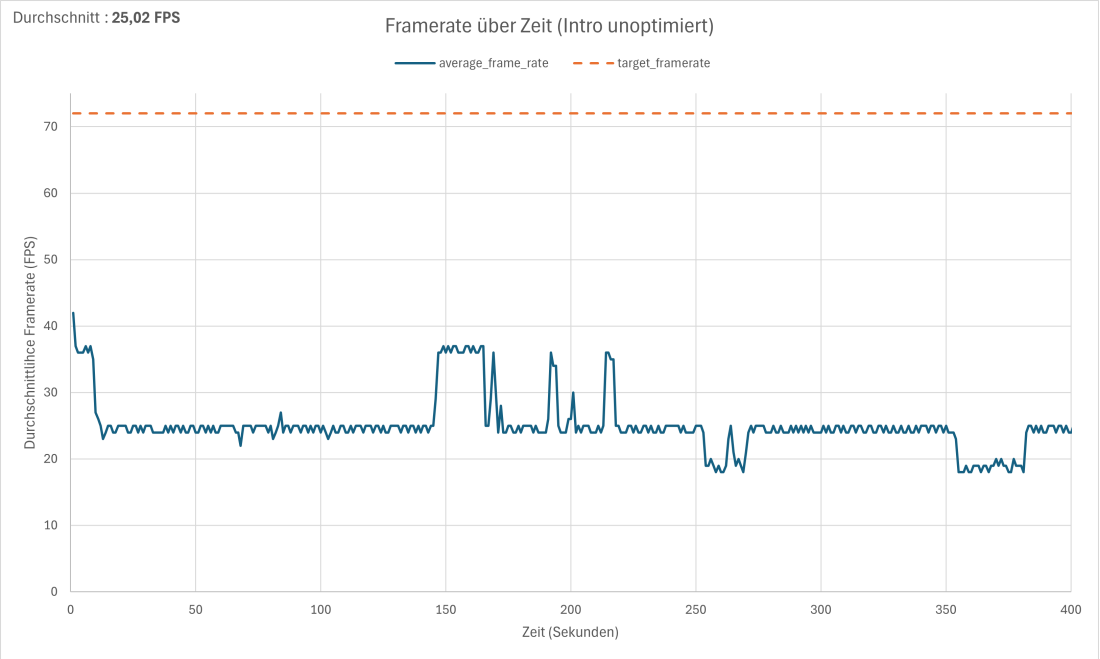
5.6.1.2. Ergebnis der Optimierungen:

Die dargestellten Maßnahmen führten zu einem deutlichen Performancegewinn: Die durchschnittliche Performance stieg im gleichen Spielabschnitt auf **71,27 FPS**, außerdem war die subjektiv wahrgenommene Bildqualität ebenfalls wesentlich besser als zuvor. Die Bildrate blieb größtenteils stabil, lediglich beim Aktivieren mancher Objekte kam es vereinzelt zu kurzzeitigen Frame-Einbrüchen. Das MR-Portal erwies sich als leistungshungrig, insbesondere wenn es einen größeren Teil des Sichtfelds einnahm. Dies liegt wahrscheinlich an erhöhtem Overdraw bei der Kombination von Stencil-Buffer und URP-Rendering.

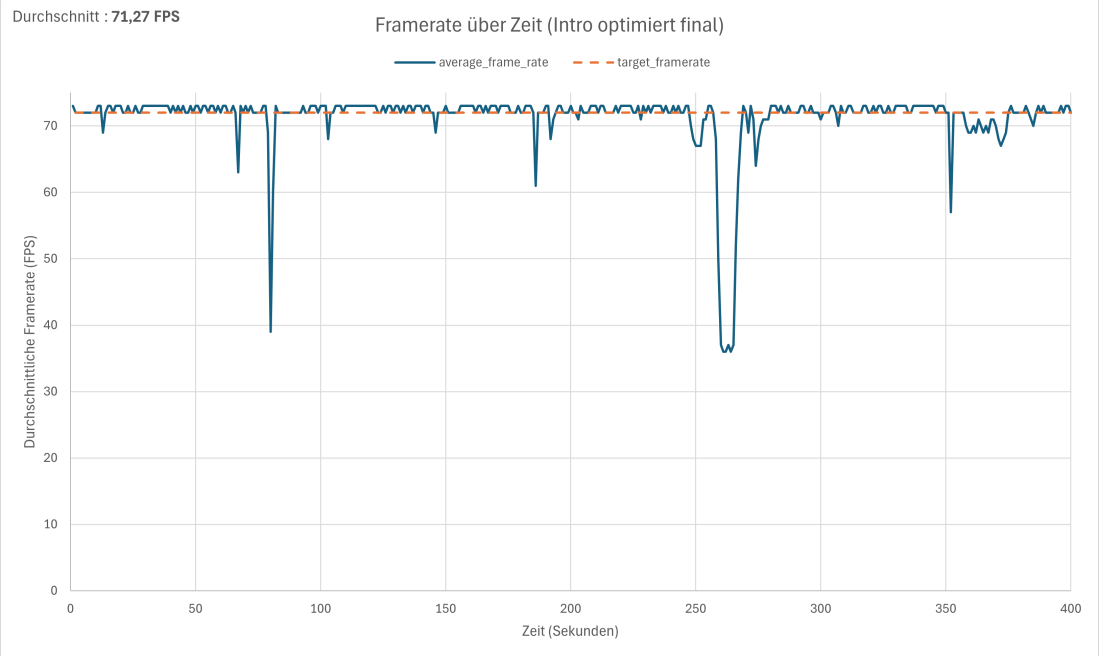
Da die Bildrate nun durchschnittlich sehr nahe an den geforderten 72 FPS liegt und das Nutzungserlebnis insgesamt zufriedenstellend war, wurde auf eine weitere Optimierung zunächst verzichtet. Kurzzeitige Framedrops können bei der Quest 3 außerdem durch sogenannte `Stale Frames` abgefangen werden. Diese ermöglichen es dem Headset, die Kopfbewegungen auch bei niedrigerer Framerate weiterhin flüssig darzustellen, indem Bewegungsinformationen vorheriger Frames interpoliert werden [71].

Auch in den übrigen Leveln des Spiels konnte nach diesen Maßnahmen eine weitgehend stabile Bildrate von **72 FPS** erreicht werden.

Die mit dem OVR Metrics Tool ausgelesenen Daten befinden sich als Excel-Datei im Verzeichnis `Performance-Daten` auf dem Datenträger (siehe Anhang B).



(a) Framerate über Zeit im Intro vor den Performance-Optimierungen.



(b) Framerate über Zeit im Intro nach den Performance-Optimierungen

Abbildung 5.11.: Framerate über Zeit im Intro vor und nach den Performance-Optimierungen (Daten über OVR-Metrics-Tool ausgelesen [65]).

5.6.2. Hand-Tracking-Herausforderungen

Ein weiteres Problem trat im Zusammenhang mit dem Hand-Tracking auf, konkret bei der Kollisionserkennung der virtuellen Hände. Dieses Verhalten zeigte sich ausschließlich

in Situationen, in denen sich das XR-Rig zuvor durch fremde Einflüsse bewegte, z.B. durch das Folgen des Charakters. In solchen Fällen war das Hand-Tracking für kurze Zeit unzuverlässig, vermutlich weil die Handpositionen nicht sofort korrekt aktualisiert wurden.

Besonders auffällig war dabei, dass die visuelle Darstellung der Hände weiterhin exakt wirkte, während gleichzeitig physikalische Kollisionen und Hover-Interaktionen fehlerhaft waren. Dies zeigte sich beispielsweise durch plötzliche, unvorhersehbare Bewegungen physikalischer Objekte, obwohl diese scheinbar gar nicht berührt wurden. Auch fehlende Hover-Outlines bei interaktiven Objekten traten auf.

Diese Beobachtungen deuten darauf hin, dass die physikalischen Hand-Collider (Hand-Kapseln) nicht korrekt mit der Handposition synchronisiert wurden - ein Effekt, der möglicherweise auf eine fehlerhafte Aktualisierung des XR-Subsystems beim Bewegen des Rigs hinweist.

Lösungsansätze und Workaround: Zur Problemlösung wurden zunächst verschiedene Ansätze getestet, etwa die Anpassung der Update-Methoden für das Handtracking sowie die Änderung der Input-Einstellungen innerhalb der XR Interaction Toolkit Komponenten. Diese führten jedoch zu keiner Verbesserung des Verhaltens.

Letztendlich wurde ein Workaround implementiert: Die physikalischen Eigenschaften der Handkapseln wurden vollständig deaktiviert. Zur Erkennung von Hover-Interaktionen wurde stattdessen ein eigener Collider direkt am Hand-Anchor des XR-Rigs befestigt. Dieser neue Collider wurde zuverlässig aktualisiert und konnte so zuverlässig für Hover-Effekte (z.B. Outlines oder Kraftfeld-Aktivierung) genutzt werden.

Der Nachteil dieser Lösung besteht jedoch darin, dass eine Interaktion mit physikalischen Objekten durch bloßes Berühren nun nicht mehr möglich ist, da die Hände keine physischen Eigenschaften mehr besitzen. Gegenstände lassen sich somit nur noch greifen, jedoch nicht mehr allein durch Handbewegung verschieben oder stoßen.

5.6.3. Herausforderungen bei der Charaktersteuerung

Auch die Implementierung einer zuverlässigen Charaktersteuerung stellte eine zentrale Herausforderung dar, insbesondere im Hinblick auf die einhändige Steuerung mit nur einem Controller. Diese limitiert die Anzahl gleichzeitig möglicher Aktionen erheblich: Weitere Interaktionen zu implementieren, die ausführbar sein sollen, während der Charakter läuft, wäre sehr schwierig, da schlichtweg nicht genug gleichzeitig erreichbare Buttons vorhanden sind.

Auch auf technischer Ebene ergaben sich im Laufe der Entwicklung einige Probleme: Der Charakter kann in bestimmten Situationen in kleineren Lücken oder Kantenbereichen stecken bleiben. Des Weiteren ist bei hohen Geschwindigkeiten, etwa beim Dash oder bei Stürzen aus größerer Höhe, die Kollisionserkennung unzuverlässig: Der Charakter kann sich durch andere Objekte hindurch bewegen, oder sogar durch den Boden fallen (Clipping).

Diese Fehler lassen sich wahrscheinlich auf Limietierungen in der Aktualisierung der Kollisionserkennung sowie auf die Konfiguration der Rigidbody- und Collider-Komponenten zurückführen. Die aktuelle Implementierung ist für einen Prototyp funktional ausreichend, jedoch besteht in dieser Hinsicht klar Optimierungspotenzial für eine stabilere und robustere Steuerungslogik.

Optimierungspotenzial gibt es auch bei weiteren Funktionalitäten. Da es bei der Charaktersteuerung besonders auffällig ist und gegebenenfalls zu Störungen im Spielgeschehen führen kann, wurde dies an dieser Stelle besonders herausgehoben.

6. Evaluierung des Prototyps

6.1. Testziele

Ziel der Evaluation ist es, die Usability der Anwendung auf Basis der zuvor definierten Design-Heuristiken zu untersuchen und die Spielbarkeit (z.B. Spielspaß, Motivation, Balancing) zu bewerten. Dabei wird insbesondere geprüft, inwiefern die zuvor formulierten funktionalen und nicht-funktionalen Anforderungen erfüllt werden.

6.2. Testdesign und Methodik

Zur Evaluation wurde ein explorativer Ansatz gewählt, der quantitative und qualitative Daten erfasst. Im Zentrum stehen Usability-Tests mit Playtesting-Komponenten, die durch eine qualitative Nachbefragung ergänzt werden, sowie ein standardisierter Fragebogen.

6.2.1. Testansatz

Folgende Methoden werden für die Tests eingesetzt:

- **Usability-Test:** Beobachtete Spielsession mit festgelegten Aufgaben zur Interaktion mit der XR-Umgebung und den Spielmechaniken. Die Teilnehmenden wurden dabei bei der Interaktion beobachtet, und relevante Beobachtungen wie Probleme, Missverständnisse oder Verzögerungen wurden dokumentiert. Durch ein anschließendes Interview wird zusätzliches Feedback eingeholt.
- **Playtesting:** Die subjektive Wahrnehmung von Spielspaß, Motivation und Balancing wird zugleich über freies Spielen und die anschließende Nachbesprechung untersucht.
- **Fragebogen:** Ein standardisierter Fragebogen (inkl. der System Usability Scale, SUS [72]) wurde eingesetzt, um quantitatives Feedback zur Benutzerfreundlichkeit und zum Spielerlebnis zu erfassen.

6.2.2. Testpersonen

Die Tests wurden mit insgesamt **15 Teilnehmenden** (Alter: 12-56 Jahre) durchgeführt. Der Usability-/Playtest wurde dabei bei **fünf Teilnehmenden** dokumentiert.

Bei der Auswahl der Teilnehmenden am beobachteten und dokumentierten Usability-Test wurde darauf geachtet, dass sie der Zielgruppe der Anwendung entsprechen. Dazu zählen Casual- oder Core-Gamer mit einem grundsätzlichen Interesse für XR-Anwendungen. Vorerfahrung mit XR war nicht erforderlich, wurde allerdings dokumentiert, um Rückschlüsse auf Unterschiede im Nutzungserlebnis ziehen zu können.

Teilnehmende, bei denen Daten nur über den Fragebogen erhoben wurden, entsprachen nicht zwingend der Zielgruppe. Es galt lediglich die Voraussetzung, dass mindestens das erste Level abgeschlossen werden musste, damit die Spielmechaniken ausreichend erprobt wurden.

6.2.3. Bezug zur Anforderungsliste

Die Durchführung der Tests orientierte sich an den zuvor definierten funktionalen (*vgl. Tabelle 5.1*) und nicht-funktionalen Anforderungen (*vgl. Tabelle 5.2*). Diese beinhalten auch die vorgestellten Heuristiken (*siehe Kapitel 3.1 und 3.2*) als Grundlage für die spätere Analyse der Ergebnisse.

Um die Nachvollziehbarkeit zu erhöhen, wurde eine Zuordnung zwischen Anforderungen, Testmethode und zu erwartendem Beobachtungsverhalten vorgenommen (siehe Anhang C.1).

6.3. Durchführung

Die fünf beobachteten Usability- und Playtests wurden zwar an verschiedenen Orten durchgeführt, allerdings unter möglichst gleichen Bedingungen. Alle Tests fanden bei den Testpersonen zuhause statt, was dem typischen Nutzungskontext des Spiels entspricht. Vor Beginn wurde ein geeigneter Spielbereich mit ausreichend Platz und Boundary-Setup eingerichtet.

Im Anschluss wurden die Daten zur Testperson erfasst und die grundlegende Bedienung der Meta Quest 3 erläutert, insbesondere die Zentrierung des XR-Rigs. Danach erhielten die Teilnehmenden das Headset sowie einen Controller. Das Spiel war dabei bereits gestartet und befand sich im Hauptmenü.

Die Teilnehmenden wurden angewiesen, ein neues Spiel zu starten und so weit zu spielen, wie sie möchten. Sie wurden gebeten, laut zu denken (Thinking-Aloud-Methode [73]), um bessere Einsichten in die Gedanken und Eindrücke zu erhalten. Alle relevanten Beobachtungen (z.B. Kommentare, Probleme, Erfolge) wurden in einem Testprotokoll festgehalten. Anschließend folgte die Nachbefragung mit offenen Fragen und die Testpersonen wurden gebeten, den standardisierten Fragebogen zu beantworten.

Nach dem Test wurde das jeweilige Protokoll überarbeitet: Die Beobachtungen wurden den jeweiligen Anforderungen zugeordnet und ein Kommentar bzw. eine Hypothese mit Verbesserungsvorschlägen wurde hinzugefügt, um die spätere Auswertung zu erleichtern.

Auch bei Testpersonen, bei denen kein Beobachtungsprotokoll erstellt wurde, sondern lediglich der Fragebogen verwendet wurde, wurde die gleiche Ausgangssituation geschaffen. Zusätzlich wurde der Prototyp im Rahmen des XR-Days des Nürnberg Digital Festivals präsentiert [74]. Besucher, die mindestens ein Level durchspielten, wurden gebeten, den Fragebogen auszufüllen.

6.4. Ergebnisse

6.4.1. Ergebnisse des Usability- und Playtests

Die Auswertung der protokollierten Usability- und Playtests zeigt ein insgesamt positives Bild hinsichtlich der Spielbarkeit, Immersion und des allgemeinen Spielerlebnisses. So gaben alle Teilnehmer an, dass ihnen das Spiel Spaß gemacht habe und sie es weiterempfehlen, beziehungsweise weiterspielen würden, wenn es weitere Inhalte gäbe. Die zentralen Interaktionen wurden von allen verstanden und in den meisten Fällen erfolgreich ausgeführt. Allerdings traten auch wiederkehrende Usability-Probleme auf, insbesondere im Zusammenhang mit UI-Interaktionen, Multi-Tasking und der Charaktersteuerung. Auch Probleme hinsichtlich der VR-Krankheit traten zum Teil auf - die Komfort-Optionen wurden entsprechend genutzt und hauptsächlich positiv bewertet. Im Folgenden werden die wichtigsten Erkenntnisse zusammengefasst:

Steuerung und Bewegung (F1, F2, F6): Die Charaktersteuerung wurde insgesamt als motivierend und intuitiv wahrgenommen. Teilweise wurde eine Eingewöhnungszeit, mit klar erkennbarer Lernkurve, beobachtet. Die Steuerung mit dem einhändigen Controller und die vom Blickwinkel des Spielers abhängige Bewegungsrichtung des Charakters wurde häufig als ungewohnt beschrieben und erforderte eine zusätzliche Eingewöhnung. Der Dash mit dem Charakter wurde durch die Positionierung des Buttons am Controller von mehreren Testpersonen häufiger aus Versehen ausgelöst.

Bei der Spielerbewegung und den entsprechenden Anpassungsoptionen wurde folgendes beobachtet: Die Möglichkeit der freien Bewegung im Raum sowie das selbstständige Zentrieren wurden von allen verstanden und genutzt. Die meisten Teilnehmer verwendeten den „Immersiven Modus“, in dem man sanft gleitend dem Charakter folgt. Allerdings zeichnete sich häufiger ab, dass die Testpersonen die zur Verfügung stehenden Komfort-Optionen nicht vollständig verstanden.

Lediglich eine Testperson musste im ersten Level unmittelbar vom „Immersiven-“ zum „Komfort-Modus“ wechseln, da sich starke Symptome der VR-Krankheit abzeichneten, darunter Schwindel und Übelkeit. Der Komfort-Modus wurde daraufhin als sehr hilfreich bewertet - die Testperson konnte ohne Probleme weiter spielen. Beim „Komfort-Modus“ zeichnete sich jedoch ab, dass die Kamerawechsel zwar zuverlässig funktionierten, zum Teil jedoch störend bei der Charakterbewegung sein können.

Handinteraktionen (F3, F4, NF1a): Das Greifen von Objekten wurde grundsätzlich intuitiv verstanden und von vielen Testpersonen positiv bewertet. Die Präzision der Interaktion variierte jedoch je nach Testperson. Besonders kleinere Hände (z.B. TP-01) führten zu Problemen beim Greifen und Loslassen. Wenn Objekte besonders präzise platziert werden mussten, wurde dies teils als umständlich empfunden.

Das Distanz-Greifen hingegen wurde durchweg als „befriedigend“ und zuverlässig wahrgenommen. Hier kam es lediglich zu Beginn teils zu Verständnisproblemen, da viele Teilnehmer dachten, sie müssten die Objekte ebenfalls direkt greifen.

Die Handposen wurden meistens korrekt ausgeführt und recht schnell verstanden - die Detektion der Posen wirkte größtenteils sehr zuverlässig. Den Kommentaren der Nutzer ist zu entnehmen, dass die Funktion als innovativ aufgefasst und zunächst positiv bewertet wird. Während die meisten Spieler ein paar Anläufe benötigten, um die Ausführung der Posen und die gleichzeitige Charaktersteuerung zu beherrschen, führte das Multi-Tasking bei manchen Teilnehmern zu stärkeren Problemen.

Benutzeroberfläche (F5, NF1a, NF1e): Alle Testpersonen hatten zu Beginn Schwierigkeiten mit der UI-Interaktion im Hauptmenü. Die erwartete Interaktion mit der Hand war nicht selbsterklärend und Buttons reagierten teilweise nicht zuverlässig. Diese Probleme wurden bei allen Testpersonen unabhängig voneinander beobachtet, was auf eine fehlende Natürlichkeit sowie mangelhaftes Feedback hinweist. Ansonsten waren die UI-Elemente größtenteils verständlich aufgebaut und halfen bei der Nutzerführung. Lediglich die Darstellung der Komfortoptionen erwies sich als teilweise unverständlich.

AR/VR-Übergang und Immersion (F7, NF1f): Der AR/VR-Übergang durch das Portal wurde von allen als immersiv, intuitiv und emotional positiv bewertet. Aussagen wie „Das kribbelt in den Fingern“ oder „Das war voll geil“ belegen deutlich die Wirksamkeit des Übergangs.

Die Beobachtungen zeigen außerdem, dass das Dialogsystem und der räumliche Klang zur Immersion beitragen. Viele Testpersonen haben mit dem Charakter gesprochen oder antworteten ihm, wenn er etwas sagte.

Levelarchitektur und Spielwelt (F13, F14, NF1b): Die Struktur der Level und die Navigation durch die Spielwelt wurde als verständlich empfunden. Die Spieler konnten sich meist gut orientieren, äußerten in Einzelfällen allerdings Unsicherheit über den nächsten Handlungsschritt. Unvorhergesehenes Verhalten, wie die Mitnahme von Objekten bei gleichzeitigem Ignorieren von Checkpoints, führte vereinzelt zu Situationen, in denen der Spieler gefangen war und keine Progression mehr machen konnte („Softlock“). Die Tooltips wurden teilweise übersehen oder ignoriert, was den Lernprozess negativ beeinflusste. Die Wiederverwendbarkeit von gelernten Mechaniken zwischen den Leveln wurde beobachtet und spricht für ein konsistentes Design.

Feedback, Motivation und Flow (NF1e, F9, 10): Das audio-visuelle Feedback wurde häufig gelobt. Besonders der Sound bei erkannten Handposen und die Outline-Farben von greifbaren Objekten wurden als hilfreich beschrieben.

Die Sammelobjekte wirkten meist stark motivierend, es wurde häufig eine gezielte Suche nach ihnen beobachtet. Das Lebenssystem wurde meist nach kurzer Spielzeit verstanden und beeinflusste das Verhalten der Spieler: Einige verzichteten bewusst auf schwer erreichbare Collectibles, um keine Leben zu verlieren. Mehrere Testpersonen beschrieben ein starkes Flow-Erlebnis während des Spiels. Hin und wieder konnte aber auch eine gewisse Frustration beobachtet werden, wenn Herausforderungen zu groß waren.

Design und Ästhetik (NF4): Das visuelle und auditive Design wurde von allen Testpersonen durchweg positiv bewertet. Besonders hervorgehoben wurde die Gestaltung der Spielwelt, die stimmige Farbgebung sowie die atmosphärische Hintergrundmusik. Die Kommentare der Nutzer verdeutlichen, dass sowohl die visuelle Präsentation als auch narrative Elemente zur ästhetischen Wahrnehmung beigetragen haben.

Auch das visuelle Feedback-System (z.B. Partikeleffekte, Outline-Farben) wurde als hilfreich und immersiv beschrieben. Kleinere Kritikpunkte betrafen vereinzelt die grafische Auflösung („manche Bereiche sahen pixelig aus“, TP-04).

Exploratives Verhalten (NF1f): Die Spielmechaniken scheinen exploratives Verhalten zu fördern. Mehrere Testpersonen probierten bewusst alternative Lösungen aus, etwa indem sie Steine mitnahmen, um Sprünge zu erleichtern, oder absichtlich das Portal blockierten, um die Reaktion zu testen.

In vielen Fällen zeigten Testpersonen Neugier gegenüber der Spielumgebung und suchten gezielt nach versteckten Objekten. Die Levelstrukturen, die teils versteckte Abkürzungen bieten sowie Sammelobjekte, scheinen dieses Verhalten zusätzlich zu unterstützen.

Sicherheit (NF1c): Hinsichtlich der Sicherheit wurden zum einen einige Fehlbenutzungen beobachtet: Darunter beispielsweise der bereits erwähnte, aus Versehen ausgeführte Dash oder das Distanz-Greifen, was zunächst wie das normale Greifen ausgeführt wurde. Bei zwei Testpersonen wurden Objekte versehentlich während des Greifens durch Wände oder den Boden geclippt, woraufhin sie nicht wiedergefunden wurden.

Physische Sicherheit war ebenfalls ein Thema: Eine Testperson (TP-05) ignorierte mehrfach die Begrenzungen des Boundary-Setups und wäre beinahe mit realen Objekten kollidiert. Zusätzlich traten bei einzelnen Teilnehmern Anzeichen von körperlicher Überlastung auf - z.B. durch Kopfschmerzen oder Druckgefühl nach längerer Spieldauer.

Das Template der Testprotokolle sowie die ausgefüllten Testprotokolle aller Testpersonen befinden sich auf dem beigelegten Datenträger im Verzeichnis `Tests` (siehe Anhang C.2).

6.4.2. Ergebnisse des Fragebogens

Die System Usability Scale (SUS) wurde von 15 Teilnehmenden ausgefüllt. Die wichtigsten Ergebnisse sind im Folgenden dargestellt:

- **Durchschnittlicher SUS-Score: 83,17**
- **Standardabweichung: 7,04**
- **Spannweite (Min/Max): von 70,0 bis 92,5**

Um die Evaluation zu ergänzen, wurden projektspezifische Fragen gestellt, die sich auf die nicht-funktionalen Anforderungen (NF1-NF4) beziehen. Auch diesen Teil des Fragebogens beantworteten 15 Teilnehmende. Die Antworten wurden auf einer fünfstufigen Likert-Skala erfasst (1 = Stimme überhaupt nicht zu, 5 = Stimme voll zu). Die angegebenen Mittelwerte spiegeln also die durchschnittliche Zustimmung zu den jeweiligen Aussagen wider:

Tabelle 6.1.: Auswertung der projektspezifischen Fragen

Frage	**Anforderung**	**MW**	**SD**
Die Interaktionen im Spiel wirkten natürlich und nachvollziehbar.	NF1a	4,47	0,50
Die Steuerung und Benutzeroberfläche waren konsistent aufgebaut.	NF1b	4,73	0,57
Ich hatte während des Spielens keine körperlichen Probleme (z. B. Schwindel, Übelkeit).	NF1c	4,20	1,28
Ich fühlte mich beim Spielen jederzeit orientiert.	NF1f	4,40	0,88
Ich konnte das Spiel gut an meine eigenen Vorlieben anpassen.	NF1d	3,47	0,72
Ich habe klares Feedback auf meine Handlungen im Spiel erhalten.	NF1e	4,47	0,72
Die Spielerfahrung war glaubwürdig und ich fühlte mich in die Welt hineinversetzt.	NF1f	4,40	0,71
Ich wurde vom Spiel dazu ermutigt, die Umgebung selbstständig zu erkunden.	NF1f	4,20	0,98
Die Darstellung und Bewegungen wirkten flüssig.	NF2	4,53	0,62
Das audiovisuelle Design war stimmig und einheitlich.	NF4	4,80	0,40

MW = Mittelwert, SD = Standardabweichung.

Ein Ausdruck des Fragebogens sowie eine Excel-Tabelle mit den Antworten, der SUS-Auswertung und der Auswertung der projektspezifischen Fragen befinden sich auf dem beigelegten Datenträger im Verzeichnis `Tests` (siehe Anhang C.2).

7. Diskussion

7.1. Stärken und Limitationen der Evaluation

Durch das Erheben qualitativer und quantitativer Daten wird eine umfassende Analyse der Usability und Spielbarkeit ermöglicht. Hierbei konnten vor allem wertvolle erste Erkenntnisse gesammelt und grobe Trends erkannt werden.

Trotzdem bestehen auch Einschränkungen im Versuchsaufbau, die in zukünftigen Forschungsarbeiten optimiert werden können:

- Die Anzahl der dokumentierten Usability-Tests ist mit fünf Teilnehmenden begrenzt, was eine tiefere Analyse einschränkt.
- Die Anzahl der Teilnehmenden am Fragebogen ist mit 15 nicht ausreichend um statistisch relevante Mittelwerte und Standardabweichungen zu ermitteln.
- Die Testpersonen unterscheiden sich in Bezug auf Alter, Spiel- und XR-Vorerfahrung. Diese individuellen Unterschieden konnten das Nutzungserlebnis beeinflussen.
- Aufgrund der nicht iterativen Entwicklung konnten identifizierte Probleme nicht mehr innerhalb des Projektzeitraumes verbessert werden.

7.2. Rückschlüsse für Usability und Spielbarkeit

Die Ergebnisse der Evaluation deuten an, dass zentrale Usability-Ziele erreicht wurden. Ein SUS-Score von durchschnittlich 83,17 spricht für eine sehr hohe Benutzerfreundlichkeit [75]. Auch die projektspezifischen Fragen zu den nicht-funktionalen Anforderungen (siehe Tabelle 6.1) erhielten hohe Zustimmungswerte.

Besonders positiv wurden natürliche Interaktionen (NF1a), konsistente Steuerung (NF1b), klares Feedback (NF1e), visuelles Design (NF4) und die Immersion (NF1f) bewertet. Die Teilnehmer hatten grundsätzlich Spaß am Spielen, zeigten häufig exploratives Verhalten und wurden durch Sammelobjekte motiviert.

Schwachstellen zeigen sich in erster Linie bei der Anpassbarkeit (NF1d) sowie in Teilen der UI-Interaktionen (F5). Hier besteht deutliches Optimierungspotenzial, vor allem

durch klarere Nutzerführung durch die Komfortoptionen und robustere Interaktionen mit UI-Elementen.

Außerhalb der protokollierten Tests konnte festgestellt werden, dass Nutzer mit weniger Spiel- oder Controller-Erfahrung deutlich stärkere Probleme mit der Charaktersteuerung haben als Spieler, die der Zielgruppe entsprachen. Verbesserte Einstiegshilfen und Tutorials könnten helfen das Spiel niederschwelliger zu gestalten.

7.3. Anforderungen mit Optimierungspotenzial

Grundsätzlich zeigt sich bei allen funktionalen und nicht-funktionalen Anforderungen Spielraum für Optimierungen. Besonders hervorzuheben sind:

- **F5 - XR optimierte Benutzeroberfläche:** Die Interaktion mit der Hand in Menüs war nicht selbsterklärend. Die UI reagierte häufig nicht auf Eingaben der Nutzer. Die Interaktion mit UI-Elementen sollte zwingend überarbeitet werden.
- **NF1d - Anpassbarkeit:** Komfortoptionen wurden teilweise nicht verstanden. Die Darstellung und Kommunikation der Optionen sollte überarbeitet werden.
- **NF1c - Sicherheit:** Die physische Sicherheit sollte durch zusätzliche Hinweise verbessert werden. Fehlbenutzungen bei verschiedenen Handinteraktionen können durch verbesserte Tooltips oder anderweitige Erklärung der Mechaniken verhindert werden. Um das versehentliche Drücken des Greif-Buttons (Dash) zu vermeiden kann die Empfindlichkeit der Eingabe überprüft werden.
- **F3/F4 - Handinteraktionen:** Die Präzision der Greif-Interaktionen könnte durch Snap-Funktionen verbessert werden. Probleme beim Greifen und Loslassen von Objekten müssen behoben werden. Hier könnte man auch eine optionale Greif-Art bereitstellen, die das Greifen von Objekten durch eine „Pinch-Geste“ ermöglichen.
- **F1 - Charaktersteuerung:** Die Steuerung des Charakters mit dem Controller kann präziser und zuverlässiger gestaltet werden. Die Verwendung der Dash-Taste sowie der Dash im allgemeinen sollte kontrollierbarer sein.
- **F13 - Levelarchitektur:** Das Leveldesign kann unter anderem durch einfachere Einstiege, kürzere Level, das Beheben möglicher Softlocks und klarer kommunizierte Handlungsschritte verbessert werden.

7.4. Erfüllung der Zielsetzungen der Arbeit

Die übergeordnete Zielsetzung der Arbeit - einen Beitrag zur nutzerzentrierten Gestaltung zukünftiger XR-Erfahrungen zu leisten - wurde aus mehreren Perspektiven erreicht:

Die systematische Analyse von Usability-Heuristiken im XR-Kontext und deren Anwendung in Game Design, Entwicklung und Evaluation liefert praxisnahe Erkenntnisse. Der entworfene Prototyp belegt außerdem die Umsetzbarkeit eines immersiven XR-Plattformers mit dualer Steuerung und Übergängen zwischen AR und VR. Die abschließende Evaluation liefert Daten zur Nutzungserfahrung und identifiziert die Stärken und Schwächen des Designs.

7.5. Erfüllung der Zielsetzungen des Prototyps

Auch die Zielsetzungen des Prototyps, also die funktionalen und nicht-funktionalen Anforderungen, wurden weitgehend erfüllt:

Die funktionalen Anforderungen konnten alle erfolgreich implementiert werden und wurden größtenteils positiv bewertet. Viele dieser Anforderungen können dabei zwar hinsichtlich der technischen Umsetzung verbessert und robuster gestaltet werden, sind für einen Prototyp jedoch ausreichend verwendbar. Die nicht-funktionalen Anforderungen zur Usability zeigen hohe Mittelwerte in der Fragebogenauswertung und wurden auch in den subjektiven Einschätzungen überwiegend positiv bewertet. Einzelne Anforderungen weisen auf konkreten Optimierungsbedarf hin. Der Prototyp läuft komplett autark auf der Meta Quest 3, sollte jedoch hinsichtlich der Performance in einigen Abschnitten weiter optimiert werden.

8. Zusammenfassung und Ausblick

8.1. Zusammenfassung

Im Rahmen dieser Bachelorarbeit wurde ein Third-Person-XR-Plattformer mit dualer Eingabeinteraktion für die Meta Quest 3 entwickelt. Das Ziel, ein spielbares XR-Erlebnis zu schaffen, bei dem der Spieler mit einem einhändigen Controller einen Charakter durch die Spielwelt steuert, während er mit der anderen Hand per Gestenerkennung in die Spielwelt eingreift, wurde erfüllt. Der Prototyp kombiniert klassische Plattformer-Mechaniken mit verschiedenen Interaktionen in AR- und VR-Abschnitten und bettet diese in eine erzählerische Rahmenhandlung ein. Dabei wurden praxisnahe Erkenntnisse für die Verwendung von Usability-Heuristiken in Bezug auf XR geliefert, indem diese bei der Konzeption und Entwicklung berücksichtigt und für eine abschließende Evaluation genutzt wurden.

Zunächst wurden etablierte Usability-Heuristiken von Nielsen [11] [13] in Bezug auf XR-Anwendungen analysiert und durch spezifische XR-Designprinzipien ergänzt. Der Fokus lag hierbei vor allem auf Komfortaspekten, beispielsweise um Problemen wie VR-Krankheit entgegenzuwirken [4]. Die Auswahl der Meta Quest 3 als Zielplattform wurde hinsichtlich ihrer technischen Eigenschaften und Interaktionsmöglichkeiten begründet.

Im Game Design wurde die grundlegende Spielidee durch einen frühen Software-Prototyp zunächst auf technische Machbarkeit sowie intuitives Verständnis und Unterhaltsamkeit überprüft. Vergleichbare Spiele wurden als Inspirationsquellen analysiert und deren Einfluss auf das eigene Konzept dargestellt. Anschließend erfolgte die detaillierte Ausarbeitung der formalen und dramaturgischen Spielelemente.

Die Entwicklung des Prototyps orientierte sich an einer Anforderungsanalyse, bei der funktionale Anforderungen aus dem Game Design abgeleitet und nicht-funktionale Anforderungen hauptsächlich anhand der zuvor untersuchten Design-Heuristiken definiert wurden. Es folgt eine Vorstellung des Technologie-Stacks und die Erläuterung der Systemarchitektur. Daraufhin wurde die technische Umsetzung der einzelnen Spielelemente wie Charaktersteuerung, Handinteraktionen, der AR/VR-Übergang oder das durch Motion-Capture-Animationen und Voiceover gestützte Dialogsystem dargestellt. Die Umsetzung der Usability-Heuristiken im UX-Design fand dabei besondere Beachtung.

Abschließend wurden hier Herausforderungen und Probleme während der Entwicklung aufgezeigt.

Die Evaluierung des Prototyps zielte darauf ab, die Usability des Spiels anhand der definierten Heuristiken sowie die allgemeine Spielbarkeit zu untersuchen. Hierfür wurden beobachtete Usability-Tests zur qualitativen Erhebung von Daten durchgeführt. Ergänzt wurden diese durch eine quantitative Analyse mittels eines standardisierten Fragebogens. Die funktionalen und nicht-funktionalen Anforderungen wurden dabei im Vorfeld den Testmethoden und zu erwartenden Beobachtungen zugeordnet. Die Testergebnisse lieferten wertvolle Erkenntnisse hinsichtlich der Stärken, Schwächen und Optimierungspotenziale des Spiels.

Abschließend wurden die Ergebnisse der Evaluation diskutiert. Dabei wurden sowohl Stärken als auch Limitationen der gewählten Testmethodik betrachtet. Die Analyse lieferte konkrete Rückschlüsse auf die Usability und Spielbarkeit des Prototyps und ermöglichte die Identifikation der Anforderungen mit dem größten Verbesserungspotenzial. Die gewonnenen Erkenntnisse bilden eine Basis für zukünftige Weiterentwicklungen und zeigen, wie etablierte und XR-spezifische Heuristiken im Design- und Entwicklungsprozess eines XR-Spiels praktisch angewendet werden können.

8.2. Ausblick

Die zentralen Ziele hinsichtlich der Anforderungen und Usability-Aspekte wurden erreicht und die grundlegende Spielidee funktioniert. Der sich aus der Evaluation ergebende positive Gesamteindruck bildet eine solide Basis für zukünftige Weiterentwicklungen.

So sollen die identifizierten Stärken des Prototyps gezielt ausgebaut werden. Dazu zählen vor allem die glaubwürdige XR-Welt, das motivierende Sammelsystem, der Übergang zwischen AR und VR und das Dialogsystem. Eine Überarbeitung und Erweiterung der Spielwelt mit zusätzlichen Levels, alternativen Lösungswegen und stärkerer Integration der AR-Features kann das Spiel weiter verbessern und die Langzeitmotivation steigern. Auch die Rahmenhandlung kann zu einer tiefgehenden Geschichte ausgebaut werden, um den Spieler emotional stärker anzusprechen.

Gleichzeitig sollten die in der Evaluation aufgedeckten Usability-Schwächen behoben und Anforderungen mit Optimierungsbedarf verbessert werden. Hierfür ist in Zukunft eine iterative Arbeitsweise mit regelmäßigen Tests von Vorteil, um die Optimierungsmaßnahmen und das Balancing der Spielmechaniken zu validieren.

Für eine inhaltliche Weiterentwicklung sollen weitere Spielmechaniken integriert werden. Denkbar wären hier verschiedene Gegner mit unterschiedlichem Verhalten, die zusätzli-

che Herausforderungen für den Charakter oder die Handinteraktionen bieten. Auch neue Handinteraktionen und Fähigkeiten, die sich weiterentwickeln lassen, oder Rätselelemente könnten das Spiel sinnvoll erweitern.

Langfristig könnte das Spiel im Meta Store veröffentlicht werden. Dafür müssten allerdings weitere Anforderungen erfüllt werden. Dazu zählt beispielsweise eine stabile Performance und die Implementierung von Store-Richtlinien wie Sicherheits- und Datenschutzhinweisen [76]. Auch automatisierte Tests und eine strukturierte Update-Pipeline wären hierbei von Vorteil.

A. Übersicht über abgegebene Quellcodes

Der vollständige Quellcode ist auf dem beigelegten Datenträger abgelegt. Die Struktur folgt einer Modulaufteilung in separaten Verzeichnissen. Im Folgenden werden die Dateien kurz beschrieben.

A.1. Charaktersteuerung

Die folgenden Dateien befinden sich im Ordner `/Quellcode/Charaktersteuerung`:

Dateiname	**Beschreibung**
`Char_InputManager.cs`	Interpretation des PlayerInputs.
`Char_Movement.cs`	Bewegung des Charakters und Animationssteuerung.
`GroundDetector.cs`	Erkennung der Bodenberührung.

Tabelle A.1.: Skripte zur Charaktersteuerung

A.2. Bewegung des Spielers

Die folgenden Dateien befinden sich im Ordner `/Quellcode/Spielerbewegung`:

Dateiname	**Beschreibung**
`XRPlayer_Follow.cs`	Folgen des Charakters in verschiedenen Modi.
`CheckpointCameraSwitcher.cs`	Trigger für Wechsel der XR-Rig Position in Diorama--Modus.

Tabelle A.2.: Skripte zur Spielerbewegung

A.3. Handinteraktion: Greifen

Die folgenden Dateien befinden sich im Ordner `/Quellcode/HandinteraktionGreifen`:

Dateiname	Beschreibung
`GrabbableEventHandler.cs`	UnityEvents bei Greif-Interaktionen.
`DistanceGrabEventHandler.cs`	UnityEvents bei Fernmanipulation.
`GrabbableSoundController.cs`	Dynamisches Audiofeedback.
`HandCollisionTrigger.cs`	UnityEvents beim Berührung mit der Hand.
`OutlineController.cs`	Aktivierung visueller Umrandungen bei Greif- oder Hover-Zuständen.
`ForceField.cs`	Steuert das Kraffeld, welches bei Hover-Interaktion den Charakter nach oben transportiert.

Tabelle A.3.: Skripte zur Greif-Interaktion

A.4. Handinteraktion: Posen

Die folgenden Dateien befinden sich im Ordner `/Quellcode/HandinteraktionPosen`:

Dateiname	Beschreibung
`FoldablePlatform.cs`	Zustandssteuerung der klappbaren Plattform.
`FoldablePlatformPoseManager.cs`	Verwaltet Posen-Steuerung mehrerer FoldablePlatforms.
`MovingPlatform.cs`	Steuerung der beweglichen Plattformen.
`PoseControlledPlatform.cs`	Verwaltet Posen-Steuerung mehrerer MovingPlatforms.

Tabelle A.4.: Skripte zur Posen-Interaktion

A.5. AR/VR-Übergang

Die folgenden Dateien befinden sich im Ordner `/Quellcode/MR_Portal`:

Dateiname	Beschreibung
`PortalLayerSwitcher.cs`	Layer Anpassung für Objekte beim Durchgang durch MR-Portal.
`PortalMarker.cs`	Simple Klasse zum Markieren des Portals.
`PortalTransitionPlayer.cs`	Welten Wechsel beim Übergang des Spielers durch MR-Portal.
`StencilGeomOptimized.shader`	Stencil-Shader für das MR-Portal Material.

Tabelle A.5.: Skripte und Shader für MR-Portal

A.6. Ressourcen

Die folgenden Dateien befinden sich im Ordner `/Quellcode/Ressourcen`:

Dateiname	Beschreibung
`Coin.cs`	Logik zum Sammeln der Kristalle.
`CoinManager.cs`	Verwaltet globale Kristall-Anzahl.
`Collectible.cs`	Logik zum Sammeln von Collectibles.
`ILevelController.cs`	UI Schnittstelle für Level-Daten.
`Level_Controller.cs`	Kontrolliert Level-Verlauf.
`LifeManager.cs`	Verwaltet Lebens-System.

Tabelle A.6.: Skripte für Ressourcen

A.7. Speicherfunktion

Die folgenden Dateien befinden sich im Ordner `/Quellcode/Speicherfunktion`:

Dateiname	Beschreibung
`GameData_Controller.cs`	Speichert Spieldaten in PlayerPrefs.
`LevelDataManager.cs`	Speichert Leveldaten in JSON-File.
`SettingsManager.cs`	Speichert Nutzerpräferenzen in JSON-File.

Tabelle A.7.: Skripte für die Speicherfunktion

A.8. 2D-Benutzeroberflächen

Die folgenden Dateien befinden sich im Ordner `/Quellcode/UI`:

Dateiname	Beschreibung
`CheckpointUI.cs`	Zeigt verbleibende Leben über Checkpoints an.
`ControllerUI.cs`	Zeigt Levelinformationen an.
`GameObjectBillboarding.cs`	Richtet Objekte zum Spieler hin aus.
`LevelFinishedUI.cs`	Zeigt Ergebnisse nach Abschluss eines Levels.
`LevelSelectionUI.cs`	Zeigt Level-Informationen in Level-Auswahl.
`MainMenuContinueButton.cs`	Steuert Verfügbarkeit von „Continue“-Button.
`MenuUI.cs`	Navigation der MenuUI.
`RecenterUI.cs`	Überprüft XR-Rig Zentrierung.
`SettingsUI.cs`	Verarbeitet Interaktionen im Einstellungen-Menü.
`SmootUIFollow.cs`	Lässt UI dem Blickfeld des Spielers folgen.

Tabelle A.8.: Skripte für 2D-Benutzeroberflächen

A.9. Animationssystem

Die folgenden Dateien befinden sich im Ordner `/Quellcode/Animationssystem`:

Dateiname	Beschreibung
`AnimationSoundTrigger.cs`	Zum Triggern von Sound über Animation-Events.
`TimelineAudioTrigger.cs`	Zum Triggern von Sound über Signal-Emitter.

Tabelle A.9.: Skripte für das Animationssystem

A.10. Trigger-Zonen

Die folgenden Dateien befinden sich im Ordner `/Quellcode/Trigger_Zonen`:

Dateiname	Beschreibung
`CharacterEventTrigger.cs`	Löst Unity-Events aus, wenn der Charakter bestimmte Bereiche betritt.
`InteractionPoint.cs`	Logik für Interaktions-Punkte (z.B. zum Starten eines Levels).

Tabelle A.10.: Skripte für Trigger-Zonen

A.11. Checkpoint-System

Die folgenden Dateien befinden sich im Ordner `/Quellcode/Checkpoint_System`:

Dateiname	Beschreibung
`Char_Respawn.cs`	Logik für Charakter-Respawn.
`Checkpoint.cs`	Visuals und Logik für Checkpoints.

Tabelle A.11.: Skripte für das Checkpoint-System

A.12. Umwelt

Die folgenden Dateien befinden sich im Ordner `/Quellcode/Umwelt`:

Dateiname	Beschreibung
`CloudSpawner.cs`	Instanziierung und Bewegung der Wolken.
`FloatingIslandFloat.cs`	Bewegung der fliegenden Inseln.

Tabelle A.12.: Skripte für die Umwelt.

A.13. Sonstige Skripte

Die folgenden Dateien befinden sich im Ordner `/Quellcode/Sonstiges`:

Dateiname	Beschreibung
`AdjustForPlayerHeight.cs`	Verändert die Höhe ausgewählter GameObjects je nach Größe des Spielers.
`DialogSystem.cs`	Spielt AudioClips für Dialoge nacheinander ab.
`FocusStone_Controller.cs`	Logik für Sammeln von Fokus-Steinen und Animationen.
`FoveatedRenderingManager.cs`	Aktiviert dynamisches Foveated-Rendering mit „Medium“ als Target-Level.
`Intro_Controller.cs`	Steuert den Ablauf des Intros.
`LocatorController.cs`	Logik für das Platzieren der Fokus-Steine in der Levelauswahl.
`ObjectPlacementTrigger.cs`	Logik für Objekte, die in einem bestimmten Bereich platziert werden können.
`ObjectPositionReset.cs`	Setzt die Position und Rotation von Objekten zurück, wenn diese mit einem Reset-Trigger kollidieren.
`OutroController.cs`	Steuert den Ablauf des Outros.
`ScaleObjectByVolume.cs`	Verwendet für den Orb im Intro, um diesen anhand der Voiceover-Lautstärke zu skalieren.
`SceneLoader.cs`	Lädt eine neue Szene anhand ihres Namens.
`MiniatureTextureOptimizer.cs`	Editor Skript, das Texturen von kleinen Objekten automatisch verkleinert.

Tabelle A.13.: Sonstige Skripte.

B. Daten zur Performance-Analyse

Die mit dem OVR Metrics Tool ausgelesenen Daten zur Performance sind auf dem beigelegten Datenträger im Ordner `Performance-Daten` abgelegt. Enthalten sind drei Excel-Listen in verschiedenen Optimierungszuständen:

Dateiname	**Beschreibung**
`FPS_Over_Time_Intro_-optimized.xlsx`	Performance-Daten aus dem finalen Build, mit allen Optimierungsversuchen.
`FPS_Over_Time_Intro_-semiOptimized.xlsx`	Performance-Daten aus einem optimierten Build, ohne Textur- und Shader-Anpassungen.
`FPS_Over_Time_Intro_-unptimized.xlsx`	Performance-Daten aus dem initialen, unoptimierten Build.

Tabelle B.1.: Performance-Daten

C. Zusätzliche Informationen zur abschließenden Evaluation

C.1. Zuordnung Anforderung - Testmethode - Bewertungskriterium

Tabelle C.1.: Zuordnung von funktionalen Anforderungen zu Testmethoden

Funktionale Anforderungen	**Testmethode**	**Beobachtbares Verhalten / Bewertungskriterium**
F1: Charaktersteuerung	Usability-Test, Playtesting, Nachbefragung	Steuerpräzision, Eingewöhnungszeit, subjektive Kontrolle
F2: Bewegung des Spielers	Usability-Test, Playtesting, Nachbefragung	Bewegung im Raum, Orientierung, Motion Sickness
F3: Handinteraktion: Greifen	Usability-Test, Playtesting, Nachbefragung	Erfolgsrate und Präzision, Natürlichkeit
F4: Handinteraktion: Posen	Usability-Test, Playtesting, Nachbefragung	Korrekte Ausführung, Verständlichkeit, Multi-Tasking
F5: XR-optimierte Benutzeroberfläche	Usability-Test, Playtesting, Nachbefragung	Verständlichkeit, Lesbarkeit, Interaktion mit UI
F6: Zentrierung des XR-Rigs	Usability-Test, Playtesting, Nachbefragung	Verständlichkeit, Orientierung nach Zentrierung
F7: AR/VR-Übergang	Usability-Test, Playtesting, Nachbefragung	Verständnis, Wirkung, Immersion
F8: Speicherfunktion	Nicht explizit getestet	Verständnis für automatisches Speichern

Funktionale Anforderungen	**Testmethode**	**Beobachtbares Verhalten / Bewertungskriterium**
F9: Sammelobjekte	Playtesting, Nachbefragung	Motivation, Sammelverhalten, Sichtbarkeit
F10: Lebenssystem	Playtesting, Nachbefragung	Verständnis, Balancing, Reaktionen bei Lebensverlust/Game-Over
F11: Dialogsystem	Playtesting, Nachbefragung	Verständlichkeit, Wirkung, Aufmerksamkeit
F12: Animationssystem	Playtesting, Nachbefragung	Wirkung
F13: Levelarchitektur	Usability-Test, Playtesting, Nachbefragung	Navigation, Verständnis, Flow
F14: Weltstruktur	Usability-Test, Playtesting, Nachbefragung	Navigation, Verständlichkeit

Tabelle C.2.: Zuordnung von nicht-funktionalen Anforderungen zu Testmethoden

Nicht-funktionale Anforderungen	**Testmethode**	**Beobachtbares Verhalten / Bewertungskriterium**
NF1a: Übereinstimmung mit der realen Welt	Fragebogen, Nachbefragung	Natürlichkeit der Interaktionen, mentale Modelle
NF1b: Konsistenz	Usability-Test, Fragebogen, Nachbefragung	Einheitlichkeit in Steuerung, Interaktionen, UI, Feedback
NF1c: Sicherheit	Usability-Test, Nachbefragung	Fehlbedienung, Begrenzungen, unischere Bewegungen
NF1d: Anpassbarkeit	Usability-Test, Nachbefragung, Fragebogen	Subjektive Präferenz, Nutzung, Wunsch nach Optionen
NF1e: Feedback	Fragebogen, Usability-Test, Playtesting, Nachbefragung	Verständnis, Nutzen

Nicht-funktionale Anforderungen	**Testmethode**	**Beobachtbares Verhalten / Bewertungskriterium**
NF1f: XR-spezifische Heuristiken	Fragebogen, Usability-Test, Playtesting, Nachbefragung	Immersion, Vertrauen, exploratives Verhalten, Vermeidung von VR-Krankheit
NF2: Performance	Fragebogen, Usability-Test, Nachbefragung	subjektives Empfinden von „Flüssigkeit“
NF3: Kompatibilität	Funktionstest vorab	Vollständige Funktion ohne Zusatzhardware
NF4: Gestalterische Konsistenz	Fragebogen, Nachbefragung	Einheitliches „Look & Feel“

C.2. Testprotokoll und Fragebogen

Folgende Dateien zum Usability-/Playtest und dem standardisierten Fragebogen befinden sich im Ordner `/Tests` auf dem beigelegten Datenträger:

Dateiname	**Beschreibung**
`Fragebogen_XR-Spiel_-Auswerung.xlsx`	Antworten des Fragebogens mit SUS-Auswertung und Auswertung der projektspezifischen Fragen auf einem jeweils eigenem Blatt.
`Fragebogen_Blank_-Ausdruck.pdf`	Der verwendete Fragebogen als blanker Ausdruck von Google Forms.
`Testprotokoll_01- bis 05.pdf`	Ausdruck der ausgefüllten Testprotokolle von TP 01 bis TP 05 (auch als docx beigelegt).
`Testprotokoll_Usability_-Playtest_Template.docx`	Leeres Template des Usability-/Playtests.

Tabelle C.3.: Dateien zur abschließenden Evaluation

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] T. Rieke, „Extended Reality (XR)“, Website der FH Münster, 2023, Zugriff am 25. Februar 2025. [Online]. Verfügbar unter: https://www.fh-muenster.de/ipd/a-z/extended-reality-xr.php

[2] P. A. Rauschnabel, R. Felix, C. Hinsch, H. Shahab, and F. Alt, „What is XR? Towards a Framework for Augmented and Virtual Reality“, *Computers in Human Behavior*, vol. 133, p. 107289, 2022. [Online]. Verfügbar unter: https://www.sciencedirect.com/science/article/pii/S074756322200111X

[3] D. North Cook, R. Hunicke, C. Wise, and K. Harris, „How Do You Make VR for Everyone?“, YouTube, Game Developers Conference 2019, 2019, Auf YouTube veröffentlicht am 17.11.2021. [Online]. Verfügbar unter: https://www.youtube.com/watch?v=X-h_sDI06oc

[4] S. Vi, T. S. da Silva, and F. Maurer, „User Experience Guidelines for Designing HMD Extended Reality Applications“, in *Human-Computer Interaction – INTERACT 2019*, D. Lamas, F. Loizides, L. Nacke, H. Petrie, M. Winckler, and P. Zaphiris, Eds. Cham: Springer International Publishing, 2019, pp. 319–341. [Online]. Verfügbar unter: https://link.springer.com/chapter/10.1007/978-3-030-29390-1_18

[5] S. Wössner, „VR, AR und Co: Begriffe kurz erklärt“, Website des Landesmedienzentrums Baden-Württemberg, 2024, Zugriff am 26. Februar 2025. [Online]. Verfügbar unter: https://www.lmz-bw.de/medienbildung/themen-von-f-bis-z/virtual-und-augmented-reality/begriffsklaerungen-was-bedeuten-begriffe-wie-vr-ar-und-mehr

[6] Wikipedia-Mitwirkende, „Virtuelle Realität“, Wikipedia, Die freie Enzyklopädie, 2025, Version: 2025-02-11 15:55, Zugriff am 26. Februar 2025. [Online]. Verfügbar unter: https://de.wikipedia.org/wiki/Virtuelle_Realit%C3%A4t

[7] ——, „Stereoskopisches Sehen“, Wikipedia, Die freie Enzyklopädie, 2022, Version: 2022-02-28 16:22, Zugriff am 27. Februar 2025. [Online]. Verfügbar unter: https://de.wikipedia.org/wiki/Stereoskopisches_Sehen

[8] ——, „Immersion (virtuelle Realität)“, Wikipedia, Die freie Enzyklopädie, 2024, Version: 2024-11-17 21:17, Zugriff am 27. Februar 2025. [Online]. Verfügbar unter: https://de.wikipedia.org/wiki/Immersion_(virtuelle_Realit%C3%A4t)

[9] Meta, „Controller“, Meta Horizon - Design, o.D., Zugriff am 27. Februar 2025. [Online]. Verfügbar unter: https://developers.meta.com/horizon/design/controllers#technology

[10] Wikipedia-Mitwirkende, „Erweiterte Realität“, Wikipedia, Die freie Enzyklopädie, 2025, Version: 2025-02-15 12:12, Zugriff am 27. Februar 2025. [Online]. Verfügbar unter: https://de.wikipedia.org/wiki/Erweiterte_Realit%C3%A4t

[11] J. Nielsen, „Enhancing the explanatory power of usability heuristics“, in *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, 1994, pp. 152–158.

[12] T. C. Endsley, K. A. Sprehn, R. M. Brill, K. J. Ryan, E. C. Vincent, and J. M. Martin, „Augmented Reality Design Heuristics: Designing for Dynamic Interactions“, *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 61, no. 1, pp. 2100–2104, 2017. [Online]. Verfügbar unter: https://doi.org/10.1177/1541931213602007

[13] J. Nielsen, „10 Usability Heuristics for User Interface Design“, Nielsen Norman Group - Website, 1994, Updated Jan. 30, 2024, Zugriff am 03. März 2025. [Online]. Verfügbar unter: https://www.nngroup.com/articles/ten-usability-heuristics/

[14] Wikipedia-Mitwirkende, „Heuristische Evaluierung“, Wikipedia, Die freie Enzyklopädie, 2024, Version: 2024-12-21 23:57, Zugriff am 03. März 2025. [Online]. Verfügbar unter: https://de.wikipedia.org/wiki/Heuristische_Evaluierung

[15] A. Kendrick, „10 Usability Heuristics Applied to Virtual Reality“, Nielsen Norman Group - Website, 2021, Zugriff am 03. März 2025. [Online]. Verfügbar unter: https://www.nngroup.com/articles/usability-heuristics-virtual-reality/

[16] M. Chan, „Mental Models“, Nielsen Norman Group - Website, 2024, Zugriff am 14. März 2025. [Online]. Verfügbar unter: https://www.nngroup.com/articles/mental-models/

[17] Meta, „Wichtige Abwägungen“, Meta Horizon - Design, 2025, Version: 13.05.2025, Zugriff am 15. Mai 2025. [Online]. Verfügbar unter: https://developers.meta.com/horizon/design/mr-design-guideline/

[18] ——, „Hands“, Meta Horizon - Design, o.D., Zugriff am 07. März 2025. [Online]. Verfügbar unter: https://developers.meta.com/horizon/design/hands

[19] Wikipedia-Mitwirkende, „Vr-krankheit“, Wikipedia, Die freie Enzyklopädie, 2024, Version: 2024-12-14 13:22, Zugriff am 07. März 2025. [Online]. Verfügbar unter: https://de.wikipedia.org/wiki/VR-Krankheit

[20] Meta, „Komfort und Nutzungsfreundlichkeit in Bezug auf die Fortbewegung“, Meta Horizon - Design, o.D., Zugriff am 07. März 2025. [Online]. Verfügbar unter: https://developers.meta.com/horizon/design/locomotion-comfort-usability

[21] ——, „Nutzer*innenpräferenzen für Fortbewegung“, Meta Horizon - Design, o.D., Zugriff am 07. März 2025. [Online]. Verfügbar unter: https://developers.meta.com/horizon/design/locomotion-user-preferences

[22] Wikipedia-Mitwirkende, „Virtual reality sickness“, Wikipedia, Die freie Enzyklopädie, 2025, Version: 2025-01-15 01:33, Zugriff am 14. März 2025. [Online]. Verfügbar unter: https://en.wikipedia.org/wiki/Virtual_reality_sickness

[23] A. Koch, I. Cascorbi, M. Westhofen, M. Dafotakis, S. Klapa, and J. P. Kuhtz-Buschbeck, „See- und Reisekrankheit“, *Deutsches Ärzteblatt*, vol. 115, no. 41, pp. 687–96, 9 2018. [Online]. Verfügbar unter: https://doi.org/10.3238/arztebl.2018.0687

[24] C. Clinic, „Motion Sickness“, Cleveland Clinic, Health Library - Website, 2024, Version: 2024-08-04, Zugriff am 14. März 2025. [Online]. Verfügbar unter: https://my.clevelandclinic.org/health/diseases/12782-motion-sickness

[25] T. E. Chemaly, M. Goyal, T. Duan, V. Phadnis, S. Khattar, B. Vlaskamp, A. Kulshrestha, E. L. Turner, A. Purohit, G. Neiswander, and K. Tsotsos, „Mind the GAP: Geometry Aware Passthrough Mitigates Cybersickness“, 2025. [Online]. Verfügbar unter: https://arxiv.org/abs/2502.11497

[26] Wikipedia-Mitwirkende, „Meta Quest 3“, Wikipedia, Die freie Enzyklopädie, 2025, Version: 2025-01-02 10:45, Zugriff am 26. Februar 2025. [Online]. Verfügbar unter: https://de.wikipedia.org/wiki/Meta_Quest_3

[27] Meta, „Meta Quest 3“, Meta - Website (Shop), o.D., Zugriff am 15. März 2025. [Online]. Verfügbar unter: https://www.meta.com/de/quest/quest-3/?srsltid=AfmBOopkijLZtlkseDb6_sonqHIFKacOG4AFrWQPK7M8hDwrh1XTX1VJ

[28] Qualcomm, „Snapdragon XR2 Gen 2 Platform“, Qualcomm Website, o.D., Zugriff am 15. März 2025. [Online]. Verfügbar unter: https://www.qualcomm.com/products/mobile/snapdragon/xr-vr-ar/snapdragon-xr2-gen-2-platform

[29] VRcompare, „Meta Quest 3“, VRcompare Website, o.D., Zugriff am 15. März 2025. [Online]. Verfügbar unter: https://vr-compare.com/headset/metaquest3

[30] Wikipedia-Mitwirkende, „Meta Horizon OS“, Wikipedia, Die freie Enzyklopädie, 2025, Version: 2025-03-14 23:56, Zugriff am 15. März 2025. [Online]. Verfügbar unter: https://en.wikipedia.org/wiki/Meta_Horizon_OS

[31] D. Blinov, „Hallway Tests — Quick Feedback on Your Prototype“, Medium - Website, 2024, Version: 2024-07-29, Zugriff am 26. März 2025. [Online]. Verfügbar unter: https://medium.com/agile-pies/hallway-tests-quick-feedback-on-your-prototype-8f090b4d6a0d

[32] Wikipedia-Mitwirkende, „Platformer“, Wikipedia, Die freie Enzyklopädie, 2025, Version: 2025-03-30 20:09, Zugriff am 01. April 2025. [Online]. Verfügbar unter: https://en.wikipedia.org/wiki/Platformer

[33] M. Brown, „Super Mario 3D World’s 4 Step Level Design“, YouTube, Game Maker’s Toolkit, 2015, Auf YouTube veröffentlicht am 16.03.2015. [Online]. Verfügbar unter: https://www.youtube.com/watch?v=dBmIkEvEBtA

[34] Wikipedia-Mitwirkende, „Kishōtenketsu“, Wikipedia, Die freie Enzyklopädie, 2020, Version: 2020-10-29 15:41, Zugriff am 30. März 2025. [Online]. Verfügbar unter: https://de.wikipedia.org/wiki/Kish%C5%8Dtenketsu

[35] Nintendo, „Super Mario 3D World + Bowser’s Fury“, https://www.nintendo.com/de-de/Spiele/Nintendo-Switch-Spiele/Super-Mario-3D-World-Bowser-s-Fury-1832228.html, 2021, Nintendo Switch-Version - Zugriff am 20. Juli 2025.

[36] M. Brown, „Super Mario’s Invisible Difficulty Settings“, YouTube, Game Maker’s Toolkit, 2024, Auf YouTube veröffentlicht am 19.01.2024. [Online]. Verfügbar unter: https://www.youtube.com/watch?v=gkvyYTSKTQY

[37] ——, „Das Design von Super Mario Odyssey“, YouTube, Game Maker’s Toolkit, 2017, Auf YouTube veröffentlicht am 08.11.2017. [Online]. Verfügbar unter: https://www.youtube.com/watch?v=z_KVEjhT4wQ

[38] Wikipedia-Mitwirkende, „Moss (Computerspiel)“, Wikipedia, Die freie Enzyklopädie, 2024, Version: 2024-06-01 14:30, Zugriff am 31. März 2025. [Online]. Verfügbar unter: https://de.wikipedia.org/wiki/Moss_(Computerspiel)

[39] C. Scrivens, „Engaging VR Storytelling: A ’Moss’ Postmortem“, YouTube, Game Developers Conference 2019, 2019, Auf YouTube veröffentlicht am 17.02.2022. [Online]. Verfügbar unter: https://www.youtube.com/watch?v=GjGRg5HTfMU&list=PL2e4mYbwSTbb62aBzyKM6U7mfnEcJceyy

[40] Playful, „Lucky's Tale“, Meta Store, 2021, Zugriff am 12. Juli 2025. [Online]. Verfügbar unter: https://www.meta.com/de-de/experiences/luckys-tale/3652037328256745/?srsltid=AfmBOopSzUSxlvCPV_3WEfdagwICGAcCyQpL58s4xpVu8q196pUSTlvF

[41] M. Games, „Ven VR Adventure“, Meta Store, 2021, Zugriff am 12. Juli 2025. [Online]. Verfügbar unter: https://www.meta.com/de-de/experiences/ven-vr-adventure/3327317663977182/?srsltid=AfmBOoqU-JAtofID_ZS_3H_5LaQl4yywTn6PO7JFa3zAbgt_gSYWPfGG

[42] T. Fullerton, *Game design workshop: a playcentric approach to creating innovative games*, 5th ed. CRC Press, 2024, iSBN: 978-1-032-60700-9.

[43] Quantic Foundry, „Our Gamer Motivation Model“, Quantic Foundry Website, N.A., Zugriff am 18. Juni 2025. [Online]. Verfügbar unter: https://quanticfoundry.com/

[44] M. Csikszentmihalyi, *Flow: The Psychology of Optimal Experience*. Harper Row, 01 1990.

[45] S. Robertson and J. Robertson, *Mastering the requirements process: Getting requirements right*. Addison-wesley, 2012.

[46] U. Technologies, „OpenXR Plugin“, Unity Manual, o.D., Zugriff am 23. Juni 2025. [Online]. Verfügbar unter: https://docs.unity3d.com/Packages/com.unity.xr.openxr@1.14/manual/index.html

[47] Meta, „Meta XR UPM Packages“, Meta Horizon - Develop, 2025, Version: 01.06.2025, Zugriff am 23. Juni 2025. [Online]. Verfügbar unter: https://developers.meta.com/horizon/documentation/unity/unity-package-manager/

[48] ——, „Set up Unity for XR development“, Meta Horizon - Develop, 2025, Version: 09.06.2025, Zugriff am 23. Juni 2025. [Online]. Verfügbar unter: https://developers.meta.com/horizon/documentation/unity/unity-project-setup

[49] U. Technologies, „Introduction to scripting“, Unity Documentation, 2025, Zugriff am 23. Juni 2025. [Online]. Verfügbar unter: https://docs.unity3d.com/6000.1/Documentation/Manual/intro-to-scripting.html

[50] Retinize, „Virtual Production for Everyone“, Animotive Website, o.D., Zugriff am 23. Juni 2025. [Online]. Verfügbar unter: https://www.animotive.com/

[51] ElevenLabs, „Die realistischste KI-Sprachplattform“, ElevenLabs Website, o.D., Zugriff am 23. Juni 2025. [Online]. Verfügbar unter: https://elevenlabs.io/de

[52] Meta, „Interaction SDK Overview“, Meta Horizon, 2025, Version: Feb 18, 2025 - Zugriff am 18. Juli 2025. [Online]. Verfügbar unter: https://developers.meta.com/horizon/documentation/unity/unity-isdk-interaction-sdk-overview

[53] ——, „OVRVignette Class“, Meta Horizon, o.D., Zugriff am 18. Juli 2025. [Online]. Verfügbar unter: https://developers.meta.com/horizon/reference/unity/v72/class_o_v_r_vignette

[54] ——, „Touch Hand Grab Interactions“, Meta Horizon, o.D., Zugriff am 18. Juli 2025. [Online]. Verfügbar unter: https://developers.meta.com/horizon/documentation/unity/unity-isdk-touch-hand-grab-interaction/

[55] C. Nolet, „Quick Outline“, Unity Asset Store, o.D., Version 1.1, Zugriff am 30. Juni 2025. [Online]. Verfügbar unter: https://assetstore.unity.com/packages/tools/particles-effects/quick-outline-115488?srsltid=AfmBOorsnA4Aq6hCftFymFHYuesrYaolK8lFRK19EReiOnSI0yJnaKZJ

[56] Meta, „OneGrabRotateTransformer Class“, Meta Horizon, o.D., Zugriff am 18. Juli 2025. [Online]. Verfügbar unter: https://developers.meta.com/horizon/reference/interaction/v72/class_oculus_interaction_one_grab_rotate_transformer

[57] ——, „Distance Hand Grab Interactions“, Meta Horizon, o.D., Zugriff am 18. Juli 2025. [Online]. Verfügbar unter: https://developers.meta.com/horizon/documentation/unity/unity-isdk-distance-hand-grab-interaction/

[58] ——, „OneGrabTranslateTransformer Class“, Meta Horizon, o.D., Zugriff am 18. Juli 2025. [Online]. Verfügbar unter: https://developers.meta.com/horizon/reference/interaction/v72/class_oculus_interaction_one_grab_translate_transformer

[59] ——, „PoseExamples Scene“, Meta Horizon, 2024, Version: Aug 7, 2024 - Zugriff am 18. Juli 2025. [Online]. Verfügbar unter: https://developers.meta.com/horizon/documentation/unity/unity-isdk-pose-examples-scene/

[60] ——, „Basic Passthrough Tutorial with Building Blocks“, Meta Horizon, 2025, Version: Feb 18, 2025 - Zugriff am 18. Juli 2025. [Online]. Verfügbar unter: https://developers.meta.com/horizon/documentation/unity/unity-passthrough-tutorial-with-blocks/

[61] ——, „UI Set“, Meta Horizon, 2025, Version: Mar 18, 2025 - Zugriff am 18. Juli 2025. [Online]. Verfügbar unter: https://developers.meta.com/horizon/documentation/unity/unity-isdk-uiset

[62] A. S. Incorporated, „Mixamo - Get animated“, Mixamo - Website, o.D., Zugriff am 30. Juni 2025. [Online]. Verfügbar unter: https://www.mixamo.com/#/

[63] Retinize, „UnityAnimotiveImporterPlugin“, GitHub, o.D., Zugriff am 30. Juni 2025. [Online]. Verfügbar unter: https://github.com/retinize/UnityAnimotiveImporterPlugin

[64] Meta, „Testing and performance analysis“, Meta Horizon - Develop, 2024, Version: 30.10.2024, Zugriff am 30. Juni 2025. [Online]. Verfügbar unter: https://developers.meta.com/horizon/documentation/unity/unity-perf/

[65] ——, „Monitor Performance with OVR Metrics Tool“, Meta Horizon - Develop, 2024, Version: 28.06.2024, Zugriff am 30. Juni 2025. [Online]. Verfügbar unter: https://developers.meta.com/horizon/documentation/native/android/ts-ovrmetricstool

[66] ——, „Project Configuration Overview“, Meta Horizon - Develop, 2025, Version: 18.02.2025, Zugriff am 30. Juni 2025. [Online]. Verfügbar unter: https://developers.meta.com/horizon/documentation/unity/unity-project-configuration#set-quality-options

[67] ——, „Using Fixed Foveated Rendering“, Meta Horizon - Develop, 2024, Version: 18.10.2024, Zugriff am 30. Juni 2025. [Online]. Verfügbar unter: https://developers.meta.com/horizon/documentation/unity/unity-fixed-foveated-rendering/

[68] U. Technologies, „Baked lighting“, Unity Documentation, o.D., Zugriff am 30. Juni 2025. [Online]. Verfügbar unter: https://docs.unity3d.com/2019.1/Documentation/Manual/LightMode-Baked.html

[69] ——, „Introduction to static batching“, Unity Documentation, o.D., Zugriff am 30. Juni 2025. [Online]. Verfügbar unter: https://docs.unity3d.com/6000.2/Documentation/Manual/static-batching.html

[70] ——, „Light Probes“, Unity Documentation, o.D., Zugriff am 30. Juni 2025. [Online]. Verfügbar unter: https://docs.unity3d.com/6000.1/Documentation/Manual/LightProbes.html

[71] T. Dasch, „OVR Metrics Tool + VrApi: What Do These Metrics Mean?“, Meta Horizon, 2019, Zugriff am 30. Juni 2025. [Online]. Verfügbar unter: https://developers.meta.com/horizon/blog/ovr-metrics-tool-vrapi-what-do-these-metrics-mean

[72] J. Brooke, „SUS: A 'Quick and Dirty' Usability Scale“, in *Usability Evaluation in Industry*, P. W. Jordan, B. Thomas, B. A. Weerdmeester, and I. L. McClelland, Eds. Taylor & Francis, 1996, pp. 189–194.

[73] J. Nielsen, *Usability Engineering*. Morgan Kaufmann, 1993.

[74] X. H. Bavaria, „XR Day 2025 Nürnberg Digital Festival“, XR Hub Bavaria Nürnberg - Website, 2025, Zugriff am 24. Juli 2025. [Online]. Verfügbar unter: https://www.xrhub-nue.de/blog/xr-day-2025/

[75] A. Bangor, P. Kortum, and J. Miller, „Determining what individual SUS scores mean: Adding an adjective rating scale“, *Journal of usability studies*, vol. 4, no. 3, pp. 114–123, 2009.

[76] Meta, „Meta Quest Virtual Reality Check (VRC) guidelines“, Meta Horizon, 2025, Version: Jul 17, 2025 - Zugriff am 20. Juli 2025. [Online]. Verfügbar unter: https://developers.meta.com/horizon/resources/publish-quest-req