

Double Trouble

Kollaboratives Rätsellösen in einer dynamischen Umgebung

Bachelorarbeit
zur Erlangung des akademischen Grades

Bachelor of Science in Engineering

Fachhochschule Vorarlberg

Informatik – Software and Information Engineering

Betreut von
Dipl.-Ing. Dr. techn. Ralph Hoch

Vorgelegt von
Nina Hartmann

Dornbirn, 20. Mai 2024

Kurzreferat

Double Trouble – Kollaboratives Rätsellösen in einer dynamischen Umgebung

Diese Arbeit beschäftigt sich mit der Konzeption und Entwicklung eines interaktiven Computerspiels, das sich auf die kooperative Bewältigung von Aufgaben fokussiert. Ziel ist es, ein herausforderndes Spielerlebnis für zwei Spieler zu bieten, die gemeinsam Rätsel lösen.

Die Entwicklung eines interaktiven Koop-2D-Spiels, das kollaboratives Rätsellösen in einer sich verändernden Welt ermöglicht, stellt eine innovative Herausforderung in der Spieleentwicklung dar. In dem in dieser Arbeit entstandenen Spiel Double Trouble werden innovative Spielmechaniken konzipiert und genutzt, um die Zusammenarbeit und Kommunikation zwischen den Spielern fördern.

Die Arbeit behandelt sowohl die theoretischen Grundlagen von kooperativen Spielen als auch deren technische Umsetzung. Durch eine systematische Literaturrecherche werden die theoretischen Grundlagen erarbeitet und in einem entwickelten Spiel in die Praxis umgesetzt. Ausgehend von einer Konzeption werden das Spieldesign entworfen und eine Architektur für das Spiel entwickelt. Besonderer Fokus liegt dabei auf der Integration von alternativen Interaktionsmöglichkeiten, welche das Spiel von einem typischen 2D-Spiel abheben. Die Umsetzung erfolgt in der Unreal Engine und ermöglicht es zwei Personen, in einem 2D-Plattform kooperativ Rätsel zu lösen. Für die Umsetzung wird verstärkt die Blueprint-Engine von Unreal genutzt, um die Entwicklung zu vereinfachen.

Das entwickelte Spielkonzept und dessen Umsetzung werden anhand von Benutzerbefragungen evaluiert. Für die Benutzerbefragungen spielen Spieler immer paarweise das final implementierte Spiel und dokumentieren deren Erfahrungen anhand eines Fragebogens. Die Evaluierung zeigt, inwiefern die angestrebten Ziele erreicht wurden und welche Aspekte weiter verbessert werden können.

Schlagwörter:

2D, Koop, Jump-and-Run-Charakter, Mouse-Only-Charakter, Rätsel lösen

Abstract

Double Trouble - Collaborative puzzle solving in a dynamic environment

This thesis deals with the conception and development of an interactive computer game focussing on the cooperative accomplishment of tasks. The aim is to offer a challenging gaming experience for two players who solve puzzles together.

The development of an interactive co-op 2D game that enables collaborative puzzle solving in a changing world represents an innovative challenge in game development. In the game Double Trouble developed in this thesis, innovative game mechanics are designed and used to promote co-operation and communication between players.

The thesis deals with both the theoretical foundations of co-operative games and their technical implementation. Through a systematic literature research, the theoretical foundations are developed and put into practice in a developed game. Based on a concept, the game design is developed and an architecture for the game is developed. Special focus is placed on the integration of alternative interaction possibilities, which set the game apart from a typical 2D game. The game is realized in the Unreal Engine and enables two people to solve puzzles cooperatively in a 2D platformer. Unreal's Blueprint Engine is increasingly used for the realization to simplify development.

The developed game concept and its implementation are evaluated by means of user surveys. For the user surveys, players always play the finalized game in pairs and document their experiences using a questionnaire. The evaluation shows the extent to which the desired goals have been achieved and which aspects can be further improved.

Keywords:

2D, co-op, jump-and-run character, mouse-only character, puzzle solving

Inhaltsverzeichnis

Darstellungsverzeichnis	6
1 Einleitung	7
1.1 Motivation	7
1.2 Problemstellung	7
1.3 Anforderungen	8
1.4 Ziele	9
1.5 Aufbau der Arbeit	10
2 Stand der Technik	11
2.1 Zweidimensionale Spielgrafik	11
2.2 Spiel-Engines für 2D-Spiele	12
2.2.1 Unity	12
2.2.2 Godot	12
2.2.3 Unreal Engine 5	13
2.2.3.1 Blueprints	13
2.3 Koop-Spiele	14
2.3.1 Kategorien von Koop-Spielen	14
2.3.2 NTU und TU-Games	15
3 Spielkonzept	17
3.1 Spielidee	17
3.2 Spielmechaniken	18
3.2.1 Jump-and-Run-Charakter	18
3.2.2 Mouse-Only-Spieler	18
4 Umsetzung	19
4.1 Tools und Spielentwicklungsumgebung	19
4.2 Implementierung	20
4.2.1 Grundlegende Einstellungen	20
4.2.2 Jump-and-Run-Charakter	21
4.2.2.1 Movement	21
4.2.2.2 Visuelle Darstellung	25
4.2.3 Mouse-Only-Charakter	26

4.2.4	Leben und Schaden System	27
4.2.5	Checkpoints und Respawn	29
4.2.6	Gegner	29
4.2.7	Level Design	32
4.2.8	Sounds	33
5	Tests und Evaluation	35
5.1	Testphasen und -szenarien	35
5.2	Evaluation	36
5.2.1	Ergebnisse der Testphasen und Szenarien	36
6	Zusammenfassung und Ausblick	38
6.1	Zusammenfassung	38
6.2	Ausblick	39
7	Lizenzen	40
	Glossar	41
	Literaturverzeichnis	42
	Anhang	44
	Eidesstattliche Erklärung	56

Darstellungsverzeichnis

Abbildung 1 - Veranschaulichung 2D und 3D.....	11
Abbildung 2 - Beispiel Blueprint	14
Abbildung 3 - Portal 2.....	15
Abbildung 4 - Poker und Overcooked.....	16
Abbildung 5 - Spielidee	17
Abbildung 6 - AnimGraph Beispiel	20
Abbildung 7 - Unterschied zwischen Blueprints	21
Abbildung 8 - Jump Implementation	23
Abbildung 9 - Schuss Limitation.....	24
Abbildung 10 - Schuss Location.....	24
Abbildung 11 - Charakter-Sprites.....	25
Abbildung 12 - Idle Animation	26
Abbildung 13 - Line Trace	26
Abbildung 14 - Zweiter Spieler	27
Abbildung 15 - Gesundheit Implementierung.....	28
Abbildung 16 - Krabbe	30
Abbildung 17 - Fledermaus Movement.....	30
Abbildung 18 - Flyeye Radius	31
Abbildung 19 - Lizard Schuss	32
Abbildung 20 - Tile Sets	32
Abbildung 21 - Level-Design.....	33

1 Einleitung

In diesem Kapitel werden der Kontext und die Motivation hinter der Arbeit beschrieben und der Ausgangspunkt für die Arbeit dargestellt. Die zentrale Problemstellung, die die Arbeit adressiert, wird vorgestellt, gefolgt von den Zielen, die erreicht werden sollen. Es wird deutlich gemacht, welche Ziele zwingend sind, welche optional und welche explizit nicht als Teil der Arbeit verfolgt werden.

Die Spielentwicklung hat in den letzten Jahren erhebliche Fortschritte gemacht, insbesondere in Bezug auf die Entwicklung von kooperativen Spielen, die die Zusammenarbeit zwischen Spielern aufbaut. Diese Art von Spielen bieten eine einzigartige Möglichkeit, die sozialen Fähigkeiten der Spieler zu fördern und gleichzeitig eine unterhaltsame und herausfordernde Spielerfahrung zu schaffen. [1]

1.1 Motivation

Die Motivation für diese Bachelorarbeit ergibt sich aus mehreren Gründen. Erstens, die Entwicklung einer interaktiven Koop-2D-Spiels, das kollaboratives Rätsellösen in einer sich verändernden Welt ermöglicht, bietet eine spannende Herausforderung, die sowohl technische als auch kreative Fähigkeiten erfordert. Zweitens, die Untersuchung und Implementierung von Spielmechaniken, die die Zusammenarbeit und Kommunikation zwischen den Spielern fördern, ist wesentlicher Bestandteil der Forschung im Bereich der Spieleentwicklung. Drittens, die Möglichkeit, ein Spiel zu entwickeln, dass sowohl zur Unterhaltung aber auch die Spieler dazu ermutigt, kreativ zu denken und zusammenzuarbeiten, um gemeinsam Rätsel zu lösen, ist ein Ziel, das sowohl persönlich als auch akademisch sehr erfüllend ist.

1.2 Problemstellung

Die Entwicklung eines interaktiven Koop-2D-Spiels, das kollaboratives Rätsellösen in einer sich verändernden Welt ermöglicht, stellt eine innovative Herausforderung in der Spieleentwicklung dar. Das Ziel dieser Arbeit ist es, ein Spiel mit einzigartiger, kooperativer und unterhaltsamer Erfahrung für zwei Spieler zu schaffen, indem es innovative Spielmechaniken nutzt, die die Zusammenarbeit und Kommunikation zwischen den Spielern fördern.

Die Herausforderungen umfassen die Entwicklung der Spielmechaniken, die Gestaltung der Umgebung, die technische Implementierung mit einer Game Engine und die Durchführung von Spieletests sowie die Einbeziehung von Feedback von Zielgruppen. Jeder dieser Bereiche erfordert spezifische Überlegungen und Lösungen, um die gewünschte Spielerfahrung zu erreichen.

Die Implementierung eines zweidimensionalen kooperativen Spieles, eröffnet eine spannende, aber dennoch anspruchsvolle Aufgabe. Es geht darum, ein Gleichgewicht zu schaffen, das sowohl die Spieler unterhalten als auch fordert. Dieses Gleichgewicht ist entscheidend, um sicherzustellen, dass das Spiel weder zu komplex noch zu trivial erscheint.

Die Gestaltung der Levels und die Implementierung der Umgebungsänderungen, die vom zweiten Spieler aktiviert werden können, erfordert kreative und gestalterische Überlegungen. Es ist wichtig, eine Umgebung zu schaffen, die sowohl visuell ansprechend als auch funktional für das Lösen von Rätseln ist.

Die Umsetzung des Spielkonzepts in einer Game-Engine bringt spezifische technische Herausforderungen mit sich. Es ist notwendig, die Fähigkeiten der Engine optimal zu nutzen, um die gewünschten Spielmechaniken und die gestalterischen Visionen zu realisieren.

Die Durchführung von Spieltests und die Einbeziehung von Feedback von Zielgruppen ist entscheidend für die Verbesserung des Spiels. Es ist wichtig, die Spielmechaniken und die Umgebungsgestaltung basierend auf dem Feedback kontinuierlich zu überarbeiten, um eine optimale Spielerfahrung zu gewährleisten.

1.3 Anforderungen

Für das interaktive, kooperative 2D-Spiel, das kollaboratives Rätsellösen in einer sich verändernden Welt ermöglicht, sind die Anforderungen vielschichtig und umfassen sowohl technische als auch kreative Aspekte. Die Implementierung der Hauptcharaktere ist von entscheidender Bedeutung, da sie die Grundlage für die Interaktion und Zusammenarbeit der Spieler bilden. Jeder Charakter sollte spezifische Fähigkeiten haben, die im Spiel nützlich sind.

Das Hauptlevel, das die grundlegenden Spielmechaniken und die Möglichkeiten der Umgebungsänderung demonstriert, ist ebenfalls ein Muss. Es muss eine Vielzahl von Umgebungen und Herausforderungen bieten, die die Zusammenarbeit und Kommunikation zwischen den Spielern fördern. Zudem sollte das Level mindestens ein zentrales Rätsel enthalten, das nur durch kooperatives Handeln gelöst werden kann.

Zusätzlich zu diesen Kernanforderungen gibt es die Möglichkeit, das Spiel durch die Entwicklung eines Head-up-Displays (HUD) zu verbessern, das relevante Informationen anzeigt und die Benutzerfreundlichkeit erhöht. Die Erweiterung des Spiels um mehrere Level, die jeweils unterschiedliche Rätsel und Herausforderungen aufweisen, kann die Vielfalt und Herausforderung des Spiels erhöhen. Die Integration von Soundeffekten und Musik, die zur Atmosphäre des Spiels beitragen und die Immersion der Spieler erhöhen, ist ebenfalls ein wichtiger Aspekt.

Es ist jedoch zu beachten, dass die Bereitstellung eines Online-Koops nicht im Rahmen dieses Projekts verfolgt wird. Der Fokus liegt auf der lokalen Spielbarkeit und der Verbesserung der Spielerfahrung durch kooperatives Spielen. Diese Anforderungen bieten einen detaillierten Rahmen für die Entwicklung des Spiels, indem sie die Muss-, Zusatz-, und Nicht-Ziele klar definieren und die Prioritäten für die Ressourcenverteilung festlegen.

1.4 Ziele

Aus den im Kapitel 1.3 beschriebenen Anforderungen ergibt sich eine Menge von Zielen, die in diesem Kapitel aufgelistet werden. Diese Ziele werden in drei Kategorien unterteilt: Muss-, Zusatz-, und Nicht-Ziele. Die Mussziele müssen zwingend erfüllt werden und sind für die erfolgreiche Umsetzung des Konzepts notwendig. Die Zusatzziele sind optional und ihre Erfüllung hängt von der technischen Machbarkeit und dem dafür benötigten Aufwand ab. Die Nicht-Ziele dienen als Grenzen für das Projekt, sie sind nicht Teil dieser Arbeit und werden nicht verfolgt.

Mussziele:

- Die Implementierung beider Hauptcharaktere ist von entscheidender Bedeutung, da sie die Grundlage für die Interaktion und Zusammenarbeit der Spieler bilden.
- Mindestens ein Hauptlevel muss entwickelt werden, dass die grundlegenden Spielmechaniken und die Möglichkeiten der Umgebungsänderung demonstriert. Dieses Level dient als Einführung in die Spielwelt und bietet den Spielern die Möglichkeit, die Spielmechaniken zu erlernen und zu üben.

Zusatzziele:

- Die Entwicklung eines Head-up-Displays (HUD) ist optional, kann aber die Spielerfahrung verbessern, indem sie relevante Informationen wie Punktestände, verbleibende Zeit oder Anweisungen anzeigt.
- Die Erweiterung des Spiels um mehrere Level bietet die Möglichkeit, die Vielfalt und Herausforderung des Spiels zu erhöhen. Jedes Level kann unterschiedliche Rätsel und Herausforderungen aufweisen, die die Spieler zu neuen Strategien und Lösungen anregen.
- Die Integration von Soundeffekten und Musik trägt zur Atmosphäre des Spiels bei und kann die Immersion der Spieler erhöhen. Die Qualität und Auswahl der Sounds können die Stimmung und das Gefühl der Spielwelt maßgeblich beeinflussen.

Nicht-Ziele:

- Die Implementierung eines Online-Koops ist in dieser Arbeit nicht vorgesehen. Der Fokus liegt auf der Entwicklung eines kooperativen Spiels, das auf einem lokalen Gerät gespielt werden kann. Die Online-Funktionalität wird als Begrenzung des Projekts betrachtet und wird nicht verfolgt.

Diese Kategorisierung der Ziele bietet einen klaren Rahmen für die Entwicklung des Spiels und hilft dabei, die Prioritäten zu klären und die Ressourcen effizient zu nutzen.

1.5 Aufbau der Arbeit

Die weitere Arbeit gliedert sich in folgenden Kapiteln:

- Kapitel 2 beleuchtet den aktuellen technischen Fortschritt und präsentiert die angewandten Technologien im Kontext der Entwicklung eines 2D-Spiels. Es erläutert zudem die Unterschiede zwischen 2D- und 3D-Entwicklungsmethoden.
- Kapitel 3 konzentriert sich auf die Konzeption und die Kernidee des Spiels, die als Grundlage für die spätere Entwicklung dienen.
- Kapitel 4 setzt sich mit der Umsetzung der Spielkonzeption auseinander. Hierbei werden sowohl die Charakterentwicklung als auch das Level Design ausführlich behandelt.
- Kapitel 5 widmet sich den Testphasen und deren Bewertung, um die Qualität und Leistungsfähigkeit des Spiels zu gewährleisten.
- Kapitel 6 bietet eine Zusammenfassung der bisherigen Arbeitsschritte sowie einen Ausblick auf mögliche zukünftige Entwicklungen.
- Schließlich hebt Kapitel 7 die verwendeten Grafiken hervor und gibt einen kurzen Überblick über ihre Lizenzen, um Transparenz und Rechtssicherheit zu gewährleisten.

2 Stand der Technik

In diesem Kapitel werden Technologien, die für die Entwicklung eines 2D-Spiels genutzt werden, beschrieben. Dabei werden die Unterschiede zwischen 2D- und 3D-Technologien beleuchtet, eine klare Definition von kooperativen Spielen gegeben sowie Vergleiche mit Game-Engines und die Anwendung der Programmiersprache Blueprints vorgenommen.

2.1 Zweidimensionale Spielgrafik

Die Zweidimensionalität in Spielen zeichnet sich durch die Beschränkung der Spielwelt sowie der Bewegungen der Charaktere und Objekte auf zwei Dimensionen aus. Im Vergleich dazu bieten 3D-Spiele eine dreidimensionale Erfahrung, wobei sowohl die Umgebung als auch die Bewegungen in allen drei Raumdimensionen realisiert werden.

2D verwendet flache Grafiken, die in der Videospielbranche als Sprites bezeichnet werden. Sie werden als flache Bilder auf dem Bildschirm gezeichnet und nicht perspektivisch abhängig von der Kamera dargestellt. Abbildung 1 zeigt ein Beispiel von einem Sprite und auch den Unterschied zwischen einem 2D-Actor und einem 3D-Actor. Der 2D-Actor, in der Darstellung die Krabbe, ist von der Seite nicht sichtbar, da sie keine Tiefe (in z-Richtung) hat und wohingegen der Würfel eine Ausdehnung in z-Richtung hat und somit, als 3D-Objekt deklariert wird.

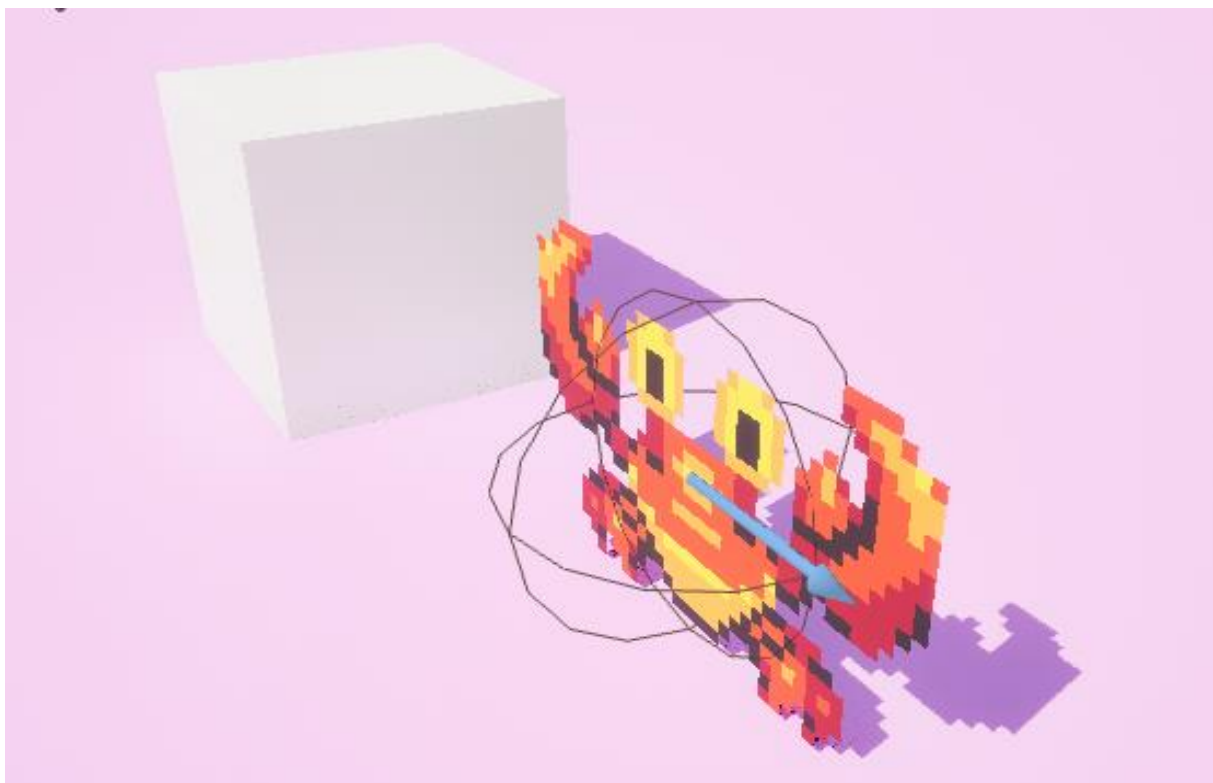


Abbildung 1 - Veranschaulichung 2D und 3D

Quelle: Nina Hartmann, 2024

2.2 Spiel-Engines für 2D-Spiele

Die Auswahl einer geeigneten Spiel-Engine ist für die Umsetzung eine notwendige und wichtige Entscheidung, da jede Engine ihre eigenen Vorzüge bietet. Die Entscheidung hängt stark von der Art des Spiels ab, das man entwickeln möchte. Zu den bekanntesten und am weitesten verbreiteten Spiel-Engines zählen *Unity* [2] und *Unreal Engine 5*. [3] Trotz ihrer Beliebtheit und ihrer umfangreichen Funktionen rücken jedoch auch andere Engines immer mehr in den Vordergrund. *Godot* [4] eignet sich, aufgrund von Spiele mit geringer Hardware-Ausstattung, vollends zur Entwicklung von 2D Spielen. Zudem ist eine Open-Source-Spiel-Engine, die sich durch ihre Flexibilität und Vielseitigkeit auszeichnet. In diesem Abschnitt wird ein Überblick über diese Spiel-Engines gegeben, gefolgt von einer Analyse und Vergleich ihrer jeweiligen Vor- und Nachteile im Hinblick auf die Entwicklung von 2D-Spielen.

2.2.1 Unity

Unity Technologies hat mit Unity eine Spielentwicklungsumgebung geschaffen, die sich durch ihre breite Akzeptanz und Vielseitigkeit auszeichnet. Diese Plattform ermöglicht die Entwicklung von Computerspielen sowie anderer interaktiver 3D-Grafikanwendungen in den Programmiersprachen C# und C++. Unity unterstützt eine Vielzahl von Zielplattformen, darunter PCs, Spielkonsolen, mobile Geräte und Webbrowser. Die Community rund um Unity ist groß und aktiv, und es gibt eine Menge von Plugins, die die Funktionalität und Assets erweitern können. Im Jahr 2022 wurden mehr als 4 Milliarden Downloads pro Monat mit Unity Apps verzeichnet. 70 % der Top 1000 Mobile-Games sind mit Unity erstellt worden. [5] Bekannte 2D-Spiele, die in Unity entwickelt wurden, sind als Beispiel *Hollow Knight*, *Cuphead* oder auch das Mobile Game *Super Mario Run*. [6]

2.2.2 Godot

Godot ist eine Open-Source-Spiele-Engine, die sich durch ihre Flexibilität und Vielseitigkeit auszeichnet. Eine der Besonderheiten von Godot ist die einfache und intuitive Programmiersprache *GDScript*, die speziell für die Spieleentwicklung konzipiert wurde. Diese Sprache ermöglicht es Entwicklern, schnell und effizient mit der Engine zu arbeiten, ohne sich um komplexe Programmierkonzepte kümmern zu müssen.

Darüber hinaus ist Godot ein Community-getriebenes Projekt, das von der Software Freedom Conservancy unterstützt wird. Dies bedeutet, dass die Engine nicht nur kostenlos und ohne Lizenzgebühren genutzt werden kann, sondern auch von einer aktiven und engagierten Community gepflegt und weiterentwickelt wird. Diese Community-Orientierung sorgt dafür, dass Godot stets regelmäßig neue Funktionen und Verbesserungen erhält.

Einige der bekanntesten Spiele, die mit Godot entwickelt wurden, sind *Brotato* und *Of Life and Land*. Diese Projekte zeigen die Vielseitigkeit und Leistungsfähigkeit der Engine und dienen als Inspiration für viele Entwickler, die ihre eigenen Spiele mit Godot erstellen möchten. [7]

2.2.3 Unreal Engine 5

Die Unreal Engine, ursprünglich als eine Plattform für die Entwicklung von 3D-Spielen konzipiert, hat sich im Laufe der Jahre zu einem leistungsstarken und vielseitigen Tool entwickelt, welches auch über die Grenzen der Spielentwicklung hinaus eingesetzt wird. Ein wichtiger Meilenstein in ihrer Geschichte war die Veröffentlichung des First Person Shooter (FPS) *Unreal Tournament* im Jahr 1999. [8] Dieses Spiel wurde für seine innovative Spielmechanik und grafische Darstellung gelobt.

Die Entwicklung von Spielen mit der Unreal Engine erfolgt in der Regel entweder in C++ oder mit den visuellen *Blueprints*, die eine intuitive und benutzerfreundliche Programmiersprache bieten. Diese Flexibilität ermöglicht es Entwicklern, Spiele mit hohen Grafikanforderungen zu erstellen, wie zum Beispiel *Hogwarts Legacy* oder das Remake von *Final Fantasy VII*, die beide für ihre beeindruckenden visuellen Darstellungen und komplexen Weltgestaltung bekannt sind.

Die aktuelle Version 5 der Unreal Engine bringt eine Reihe von Verbesserungen gegenüber dem Vorgänger mit sich, die sowohl die Leistung als auch die Funktionalität der Engine erheblich steigern.

2.2.3.1 Blueprints

Blueprints in Unreal Engine sind ein visuelles Scripting-System, das es ermöglicht Gameplay-Elemente innerhalb des Unreal Editors zu erstellen ohne Code schreiben zu müssen. Sie basieren auf einem *Node*-basierten Interface und ermöglichen es Designern fast alle Konzepte, Tools und Funktionen zu verwenden, die normalerweise nur über die Programmierschnittstelle verfügbar sind. Darüber hinaus bieten Blueprints eine flexible und Möglichkeit, um Objekte und Klassen direkt in der Engine mittels visueller Programmierung zu erstellen.

Abbildung 2 zeigt ein Beispiel für die Anwendung eines Blueprints in einem Projekt dargestellt, welches eine Waffe repräsentiert. Es zeigt, wie ein spezifischer Sprite aktiviert wird, sobald die Waffe auf eine Wand trifft, bevor sie schließlich zerstört wird. Die Funktion *Play Hit Effects* in C++ dient als eine solche spezifische Implementierung, die für das Abspielen des Sprites verantwortlich ist und wenn diese Funktion alle Bedingungen erfüllt sind, wird sie auch aufgerufen.

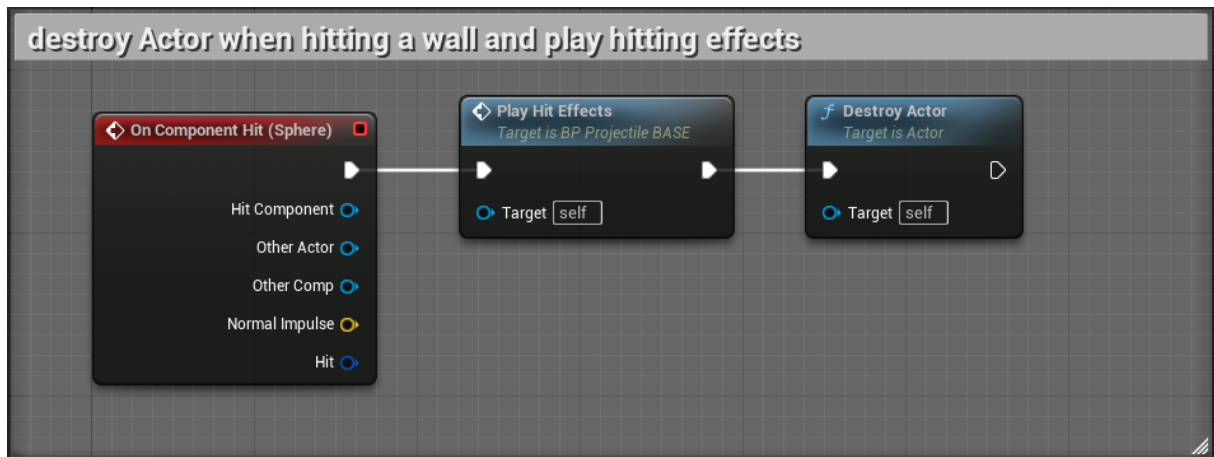


Abbildung 2 - Beispiel Blueprint

Quelle: Nina Hartmann 2024

2.3 Koop-Spiele

Ein Koop-Spiel, auch bekannt als *Cooperative Game*, ist ein Spiel, bei dem mehrere Spieler zusammenarbeiten, um ein gemeinsames Ziel zu erreichen. Im Gegensatz zu Wettbewerbs- oder Einzelspielen, bei denen die Spieler gegeneinander antreten und versuchen, sich gegenseitig zu besiegen, arbeiten die Spieler in Koop-Spielen zusammen, um Herausforderungen zu überwinden, Rätsel zu lösen oder gemeinsam ein Ziel zu erreichen.

Dieser Abschnitt gibt eine Einführung in die Kategorie von Kooperationsspielen und einen kurzen Einblick in NTU- und TU-Spiele geben.

2.3.1 Kategorien von Koop-Spielen

Aus Sicht dieser Arbeit ist eine Aufteilung der Koop-Spiele in drei Hauptkategorien unterteilt:

- Puzzle-Koop-Spiele
- Adventure-Koop-Spiele
- Strategie-Koop-Spiele

Puzzle-Koop-Spiele konzentrieren sich auf das Lösen von Rätseln und Puzzles, wobei die Spieler zusammenarbeiten müssen, um das Ziel zu erreichen. Beispiele hierfür sind *Little Big Planet* und *Portal 2*, wie man in Abbildung 3 erkennen kann. Bei diesen Spielen müssen die Spieler zusammenarbeiten, um komplexe Rätsel zu lösen.



Abbildung 3 - Portal 2

Quelle: https://store.steampowered.com/app/620/Portal_2/

Adventure-Koop-Spiele kombinieren Elemente von Abenteuer und Kooperation. Die Spieler müssen zusammenarbeiten, um durch eine Geschichte zu navigieren, Rätsel zu lösen und Herausforderungen zu überwinden. *New Super Mario Bros. Wii* und *Halo 3: ODST* sind Beispiele für Spiele, die in diese Kategorie fallen.

Strategie-Koop-Spiele erfordern die Zusammenarbeit von Spielern, um strategische Ziele zu erreichen, oft in einem Echtzeit-Strategiespiel. *Age of Empires* und *StarCraft 2* sind Beispiele für Spiele, die in dieser Kategorie fallen.

2.3.2 NTU und TU-Games

NTU- (Nontransferable utility) und TU-Spiele (transferable utility) beziehen sich auf Konzepte aus der Spieltheorie, die sich auf die Analyse von Spielen mit Kommunikationsstrukturen beziehen. Die Spieltheorie und insbesondere die Analyse von Spielen mit Kommunikationsstrukturen ist ein komplexes Feld, das sich mit der Untersuchung von Spielen beschäftigt, in denen Spieler miteinander kommunizieren können, um ihre Strategien zu koordinieren. [9]

NTU-Spiele sind eine Art von Spielen, bei denen der Nutzen, beispielsweise Chips beim Pokern, nicht zwischen Spielern übertragen werden kann. Jeder Spieler hat eigene Nutzenfunktionen, die nicht durch Verhandlungen verändert werden können. Ein gutes Beispiel hierfür ist das Kartenspiel *Poker*, bei dem jeder Spieler Chips besitzt, die nicht übertragbar sind. Jeder Spieler hat seine eigenen Ressourcen und

verfolgt individuelle Strategien, um für sich selbst zu gewinnen, wie man auf der Abbildung 4 auf der linken Seite erkennen kann.

TU-Spiele sind eine spezifische Art von Spielen, die in der Spieltheorie untersucht werden. In TU-Spielen versuchen die Spieler, ihre Ziele zu maximieren, indem sie Entscheidungen treffen, die auf den Informationen basieren, die sie über die Entscheidungen der anderen Spieler haben. Dabei kann der Nutzen auf andere Spieler übertragen werden. Ein gutes Beispiel für ein TU-Spiel ist *Overcooked*, wo Aufgaben und Ressourcen zwischen den Spielern übertragen und gemeinsam genutzt werden, um ein gemeinsames Ziel zu erreichen, wie man auch hier auf dem rechten Teil der Abbildung 4 erkennen kann. [10], [11]

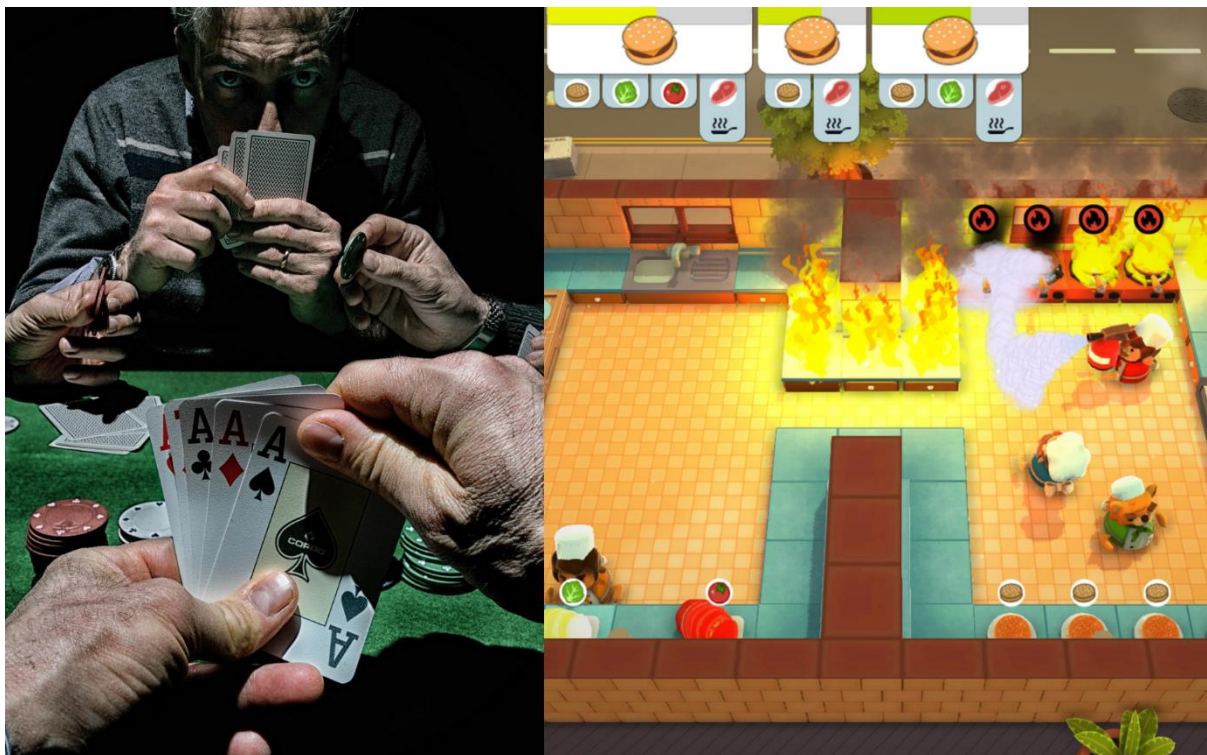


Abbildung 4 - Poker und Overcooked

<https://www.goodfon.com/situations/wallpaper-poker-karty-igra-1.html>

<https://store.steampowered.com/app/448510/Overcooked/>

3 Spielkonzept

Dieses Kapitel beschreibt die Konzeption und grundlegende Spielidee des zu entwickelnden Spiels. Weiteres werden Lösungsansätze für konkrete Anforderungen und technische Herausforderungen beschrieben. Zuvor wird jedoch ein Blick auf die Spielidee und das Spielziel geworfen.

3.1 Spielidee

Die Spielidee lehnt sich an die beliebte Spielreihe von Nintendo, *Super Mario Maker*, an, bietet jedoch eine zusätzliche Komponente: ein Kooperationspiel, das Teamarbeit und logisches Denken in den Vordergrund stellt. Im Gegensatz zu *Super Mario Maker*, das hauptsächlich für Einzelspieler konzipiert ist, zielt dieses Spiel darauf ab, mindestens zwei Spieler zu aktivem Spielen zu ermutigen, wobei die Zusammenarbeit, gutes Jump and Run und Kommunikation zum Erfolg führen.

Das Spiel ist ausschließlich für zwei Spieler konzipiert, wobei der *erste Spieler* als *Jump-and-Run-Charakter* fungiert. Der *zweite Spieler* ist ein *Mouse-Only-Charakter*, der ausschließlich durch die Maus gesteuert wird und erfüllt die Aufgabe, Blöcke zu verschieben und in bestimmte Positionen zu bringen. Die spezifischen Fähigkeiten der beiden Spieler werden im folgenden Unterkapitel *Spielmechaniken* detailliert erläutert. Jedes Spiel benötigt eine Herausforderung, und die Gegner liefern diese, indem sie den Spielern verschiedene Aufgaben auferlegen und Herausforderungen bieten, wodurch die Fähigkeiten und die Teamarbeit unter Beweis gestellt werden.

Abbildung 5 zeigt, wie die Spielidee aufgebaut ist. Ein Hindernis stellt sich dem Jump-and-Run-Charakter in den Weg, dass allein nicht überwunden werden kann. Er benötigt die Unterstützung des Mouse-Only-Charakters, der einen Block herbeiruft und diesen in die richtige Position bringt, um das Hindernis gemeinsam zu meistern.

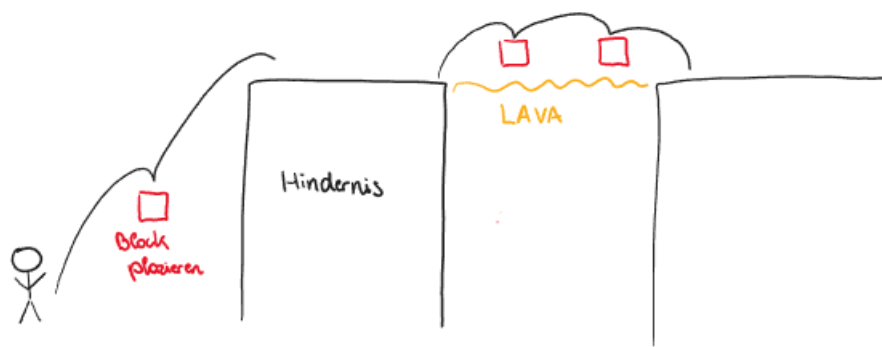


Abbildung 5 - Spielidee

Quelle: Nina Hartmann, 2024

3.2 Spielmechaniken

In diesem Abschnitt werden die zentralen Spielmechaniken der beiden Hauptcharaktere des Spiels vorgestellt. Dabei fungiert der Jump-and-Run-Charakter als der körperliche Aspekt, während der Mouse-Only-Charakter als der intellektuelle Kopf des Spielers agiert. Durch ein koordiniertes Zusammenspiel müssen beide Charaktere gemeinsam Rätsel lösen und so das Ziel erreichen.

3.2.1 Jump-and-Run-Charakter

Die Aufgabe des Jump-and-Run-Charakters besteht im Überwinden von Hindernissen, Besiegen von Gegnern und Durchqueren von Levels. Diese Art von Charakter ist in einer Vielzahl von Videospielgenres zu finden, wobei die grundlegenden Mechaniken das Laufen, Springen und manchmal auch das Klettern, Fliegen oder das Schießen mit einer Waffe umfassen. Ein Beispiel für einen Jump-and-Run-Charakter ist Mega Man aus der Spielreihe *Mega Man*, in der die Spielfigur durch ihre Fähigkeiten und Waffen die Herausforderungen des Spiels meistern.

Im in dieser Arbeit konzipierten Spiel ist dieser Charakter als Spieler 1 identifiziert und verfügt über grundlegende Mechaniken wie das Springen, Laufen und Schießen. Zusätzlich ist er mit einer Waffe ausgestattet, die ihm hilft, sich gegen seine Gegner zu behaupten.

3.2.2 Mouse-Only-Spieler

Die Aufgabe des Mouse-Only-Spieler ist die Bewegungen und Interaktionen ausschließlich durch die Bedienung der Computermouse gesteuert werden. Es gibt eine Vielzahl von Spielen, die diese Mechanik nutzen, wobei einige Spiele den Charakter durch das Klicken auf einen Ort zum Ziel bewegen, wie zum Beispiel *Poly Bridge*. Andere Spiele, wie *Islanders*, erfordern, dass der Spieler durch das Klicken der Maus das Spiel durchlebt und Entscheidungen trifft.

Im Spiel ist dieser Charakter als Spieler 2 bekannt und spielt eine entscheidende Rolle bei der Unterstützung des Jump-and-Run-Charakters. Er hat die Fähigkeit, einen bestimmten Block an Orte im Level zu platzieren, was ihm ermöglicht, das Spielgeschehen zu beeinflussen und zu steuern. Diese Blöcke sind der Schlüssel zu den Rätseln, die die beiden Charaktere gemeinsam lösen müssen, um das Ziel zu erreichen.

4 Umsetzung

In diesem Kapitel wird die Umsetzung des Spiels beschrieben. Zunächst werden die Tools und die ausgesuchte Spiel-Engine (bzw. hast du unten Spieleentwicklungsumgebung) vorgestellt, die für die Implementierung des Spiels verwendet wurden. Anschließend wird der Fokus auf die eigentliche Implementierung gelegt.

Zu Beginn des Kapitels 4.2 wird die Implementierung der beiden Hauptcharaktere des Spiels beschrieben. Neben den Hauptcharakteren werden auch die vier Hauptgegner des Spiels behandelt.

Zum Abschluss der Implementierung wird das Hauptlevel des Spiels detailliert beschrieben. Das Hauptlevel veranschaulicht, wie die verschiedenen Spielmechaniken und Herausforderungen umgesetzt werden. Die detaillierte Beschreibung enthält, welche Elemente der Hauptlevel umfasst und wie die Spieler diese erleben sollen.

4.1 Tools und Spielentwicklungsumgebung

Für die Entwicklung des Spiels wurde die Unreal Engine 5.3.2 ausgewählt, hauptsächlich aufgrund bestehender Vorerfahrungen und der Tatsache, dass die Engine alle definierten Anforderungen erfüllt. Zudem bieten die Blueprints eine alternative Möglichkeit zur Beschreibung des Spielablaufs. Die Wahl der neuesten Version 5.3.2 erfolgte aufgrund eines Updates, das Funktionen bereitstellt, die besonders nützlich für die Entwicklung eines Spiels mit Blueprints sind.

Durch Plugins, wie *Paper 2D*, wird die Entwicklung von 2D-Spielen erleichtert, da die Unreal Engine ursprünglich nicht optimal für die Erstellung solcher Spiele konzipiert wurde. Laut der offiziellen Webseite ist das Plugin eine Lösung für die fehlende Unterstützung der Unreal Engine für 2D-basierte Spiele. [12] Das Plugin enthält einen eigenen Blueprint für Animationen, der eine ähnliche Implementierung wie bei einem 3D-Charakter bietet. Der *Animation Blueprint* verfügt über einen eigenen Animationsgraphen, der eine einfach zu verfolgende Animationslogik für jeden Charakter ermöglicht.

Abbildung 6 zeigt einen *AnimGraph* für den Spieler 1. Dieser ist dafür da, die Animationen und mögliche Übergänge vom Zustand des Charakter abzuspielen. In jedem dieser Nodes wird ein bestimmter Sprite abgespielt, der die Bewegungen des Charakters widerspiegelt. Beispielsweise wäre *Idle* der Zustand, in dem der Charakter sich nicht bewegt, und *Run* der Zustand, in dem der Charakter läuft. Es gibt auch Animationen für Sprünge und Fallbewegungen. Wie genau dieser AnimGraph funktioniert, wird in der Implementierung des Jump-and-Run-Charakter noch genauer erläutert.

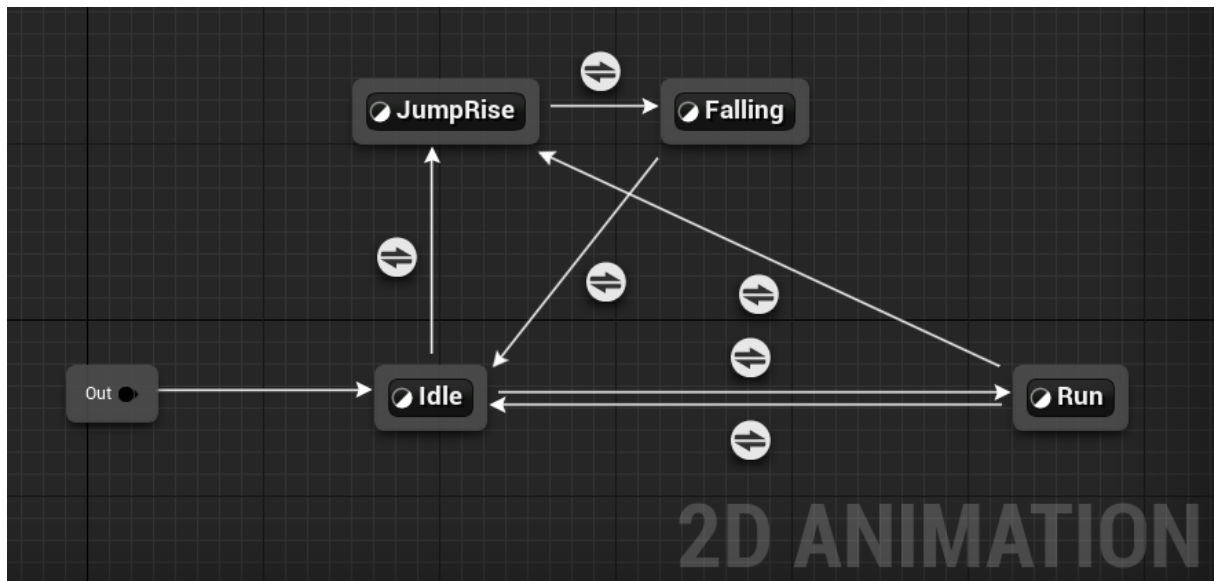


Abbildung 6 - AnimGraph Beispiel

Quelle: Nina Hartmann, 2024

4.2 Implementierung

In diesem Abschnitt wird auf die Implementierung der Hauptfiguren und Gegner, die visuelle Darstellung von Level und Spieler, sowie die Lebensanzeige und Checkpoints behandelt.

4.2.1 Grundlegende Einstellungen

Zu Beginn der Implementierungsphase wurde ein neues Projekt in Unreal Engine mit spezifischen Einstellungen konfiguriert. Es wurde für den Desktop-Bereich optimiert, wobei die Programmiersprache Blueprint gewählt wurde, um die Entwicklung zu erleichtern und zu beschleunigen. Die Qualitätseinstellungen wurden so konfiguriert, dass sie skalierbar sind, was bedeutet, dass das Spiel auf verschiedenen Geräten mit unterschiedlichen Konfigurationen (Performance) reibungslos laufen kann.

Ein wichtiger Schritt in der Entwicklung war die Anpassung der Default Pixel pro Unreal Unit von 1,0 auf 0,25. Diese Änderung beeinflusst die Größe der Objekte im Spiel und ermöglicht es, den Charakter später an die gewünschte Größe anzupassen. Dies ist besonders wichtig für die visuelle Darstellung und Spielmechanik, da es die Interaktion zwischen dem Charakter und der Umgebung beeinflusst.

Der Spielablauf wird mittels eines *GameMode* konfiguriert der für die Auswahl des Standardlevels, der Definition der Standardklassen und der Konfiguration des HUDs verantwortlich ist. Die Flexibilität des GameModes ermöglicht es, verschiedene Spielszenarien zu erstellen und anzupassen, was die Entwicklung von Levels und die Integration von Spielmechaniken erheblich vereinfacht.

Im GameMode kann auch festgelegt werden, welcher Charakter standardmäßig gespawnt wird, wenn der Spieler das Spiel startet. Dies bietet eine Möglichkeit, den Spieler direkt in die Spielwelt einzuführen und die Interaktion mit dem Spiel zu beginnen.

4.2.2 Jump-and-Run-Charakter

Abbildung 7 zeigt die verschiedenen Blueprint-Typen, die für den Charakter verwendet werden. Die drei Hauptoptionen die Charakterentwicklung zur Verfügung stehen, sind:

Der Actor ist ein grundlegendes Element in Unreal Engine, dass in die Spielwelt platziert oder erzeugt werden kann. Ein typisches Beispiel hierfür wäre ein Projektil, das von einer Waffe abgefeuert wird.

Der Pawn erweitert die Fähigkeiten eines Actors, um die Möglichkeit, Spielereingaben zu empfangen. Der Pawn kann dadurch direkt von einem Spieler gesteuert werden. Ein gutes Beispiel dafür wäre ein Fahrzeug.

Der Character ist eine spezielle Art von Pawn, der zusätzlich die Fähigkeit besitzt, sich in der Spielwelt zu bewegen. Dadurch ist er besonders für Charaktere geeignet, die sich frei in der Spielwelt bewegen können, wie zum Beispiel den Spielercharakter in einem Jump-and-Run-Spiel.

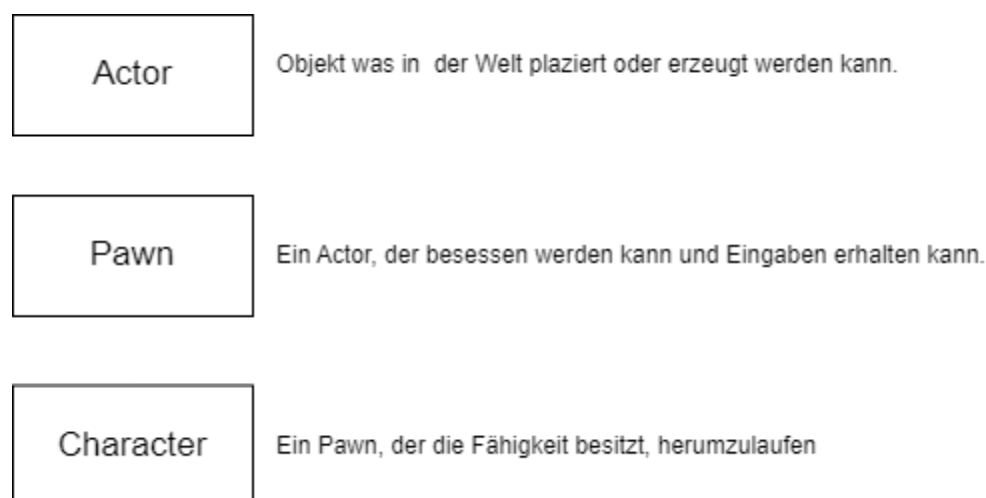


Abbildung 7 - Unterschied zwischen Blueprints

Quelle: Nina Hartmann, 2024

Für die Entwicklung des Spielercharakters wurde ein Pawn Blueprint erstellt.

4.2.2.1 Movement

Für die Implementierung der Bewegungsmechanik wurden Input Actions und Input Mapping Context genutzt, um die Steuerung des Charakters zu definieren.

Für jede einzelne Bewegung, die einen anderen Tastendruck erfordert, werden separate Input Actions erstellt. In diesem Fall wurden Input Actions für Jump, Run und Shoot definiert.

Für die Jump-Aktion wurde der Value Type als Boolean festgelegt, da nur ein einzelner Tastendruck erforderlich ist, um den Sprung auszulösen. Ähnlich wurde für die Shoot-Aktion der Value Type als Boolean definiert, da auch hier nur ein Tastendruck für den Schuss ausreicht.

Die Run-Aktion erfordert eine komplexere Handhabung, da in einem 2D-Spiel die Bewegung in beide Richtungen möglich ist. Hier wurde der Value Type als Axis1D (ein Float) festgelegt, um sowohl die Bewegung nach rechts als auch nach links zu ermöglichen.

Im Input Mapping Context werden die entsprechenden Tastenbelegungen für die verschiedenen Aktionen definiert. Für den Jump werden mehrere Tasten als Optionen festgelegt, darunter die Space Bar, die W-Taste und die K-Taste. Die Wahl der K-Taste ergab sich aus der Nähe zur Taste für Shoot (J), was für einige Spieler angenehmer sein könnte. Für die Run-Aktion wurden die Tasten D für Rechts und A für Links als Standardbelegungen festgelegt.

Um sicherzustellen, dass der Charakter auch nach links laufen kann, wurde der Float-Wert für die Run-Aktion als *Negate* deklariert. Dies bedeutet, dass der Charakter in den Minusbereich des Axis1D-Wertes wechselt, wenn der Spieler nach links laufen möchte, was wiederum eine präzise Steuerung der Bewegungsrichtung des Charakters ist.

Um den Charakter bewegen zu können, ist es notwendig, das Movement auf das Player 1 Blueprint zu übertragen. Dies erfordert den Zugriff auf den Input Mapping Context und jede Input Action. Um Zugang zum Input Mapping Context zu erhalten, muss auf den Standard PlayerController zugreifen, der bereits von Unreal Engine bei der Erstellung des Projekts hinzugefügt wurde. Von dort aus kann auf das Enhanced Input Local Player Subsystem zugreifen und mit der Funktion *Add Mapping Context* um den passenden Input Mapping Context auswählen.

Um die Eingabeaktionen zu definieren, wird ein Node *EnhancedInputAction* zusammen mit den entsprechenden Input Actions verwendet. Abbildung 8 zeigt die Standardwerte für diese Aktionen dargestellt. Für die Sprungmechanik ist es besonders wichtig, dass der Charakter höher springt, wenn der Sprungknopf lange gedrückt wird. Daher wird der Link nicht bei *Triggered*, sondern bei *Started* plziert. Die Node *Jump* ist eine bereits von Unreal Engine implementierte Funktion, die einfach aufgerufen werden kann, um zu springen. Ebenso ist *Stop Jumping* eine Funktion, die den Sprung des Charakters stoppen kann. Wenn der Spieler die Taste loslässt, soll der Charakter fallen und nicht noch höher springen. Daher gibt es zwei Links von *Canceled* zu Stop Jumping und auch wenn die maximale Höhe erreicht ist.

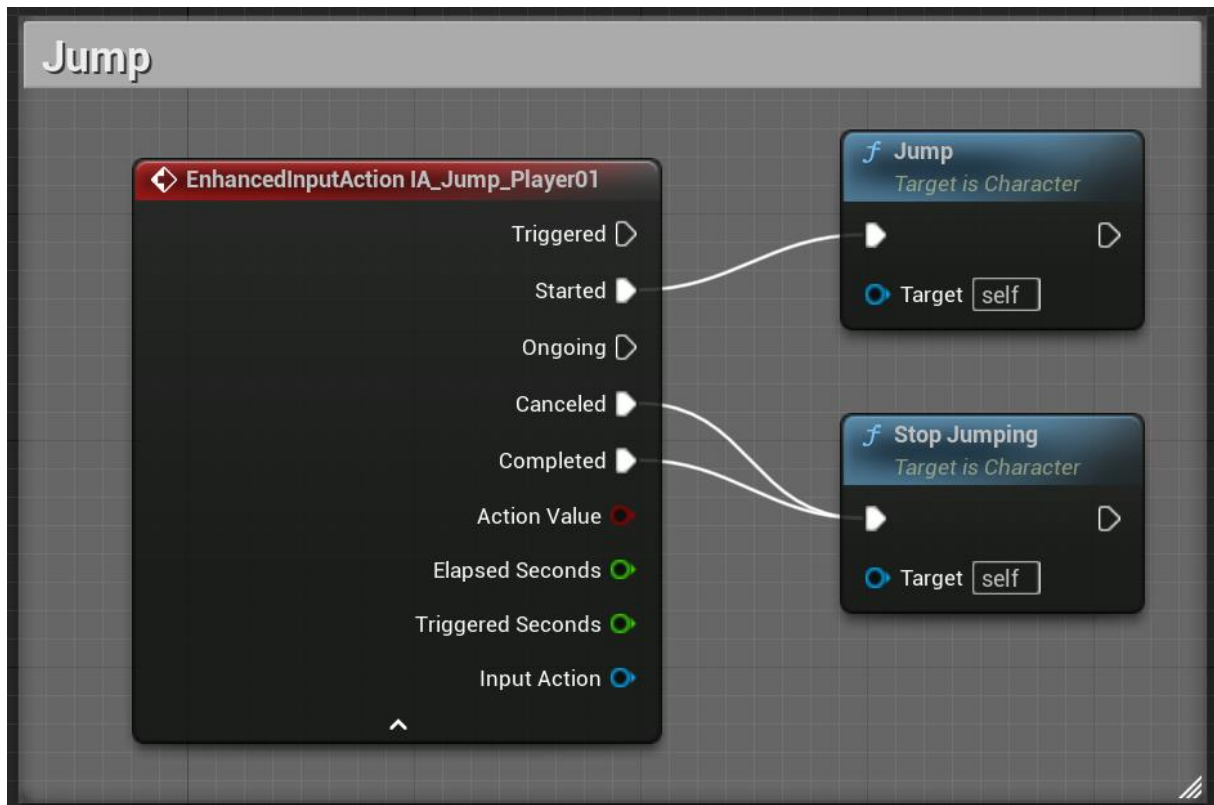


Abbildung 8 - Jump Implementation

Quelle: Nina Hartmann, 2024

Für das Movement wird die Funktion *Add Movement Input* genutzt, die bereits von Unreal Engine bereitgestellt wird. Als Standardwert wird der Float *Action Value* verwendet, der angibt, ob der Float positiv ist, was bedeutet, dass der Charakter nach rechts läuft, oder negativ, was bedeutet, dass der Charakter nach links läuft. Anschließend wird dieser von dem Input Action mit der Funktion verknüpft. Da es sich um ein 2D-Spiel handelt, wird zusätzlich die Rotation eingefügt. Dies bedeutet, dass das visuelle Erscheinungsbild des Charakters entsprechend der Bewegungsrichtung angepasst wird, indem die Funktion *Update Rotation* hinzugefügt wird.

Die Abbildung der Schlussmechanik gestaltet sich komplexer. Es soll eine Begrenzung für die Anzahl der Schüsse geben, um zu verhindern, dass diese willkürlich abgefeuert werden können. Wenn diese Begrenzung erreicht ist, soll sich die Shot Energy wieder auf den ursprünglichen Wert zurücksetzen. Auch hier ist der Zugriff auf die Input Action erforderlich. Es wird eine Variable erstellt, die bestimmt, wie viele Schüsse als Begrenzung eingegeben werden sollen. In diesem Fall sind es drei Schüsse. In Abbildung 9 ist zu sehen, dass, wenn die Variable *Shot Energy* kleiner als 0 ist, wird nichts passieren und kein weiteren Schuss abgefeuert werden kann. Wenn die Shot Energy jedoch größer 0 ist, wird die Variable *Shot Energy* um eins verringert und ein Schuss ausgeführt.

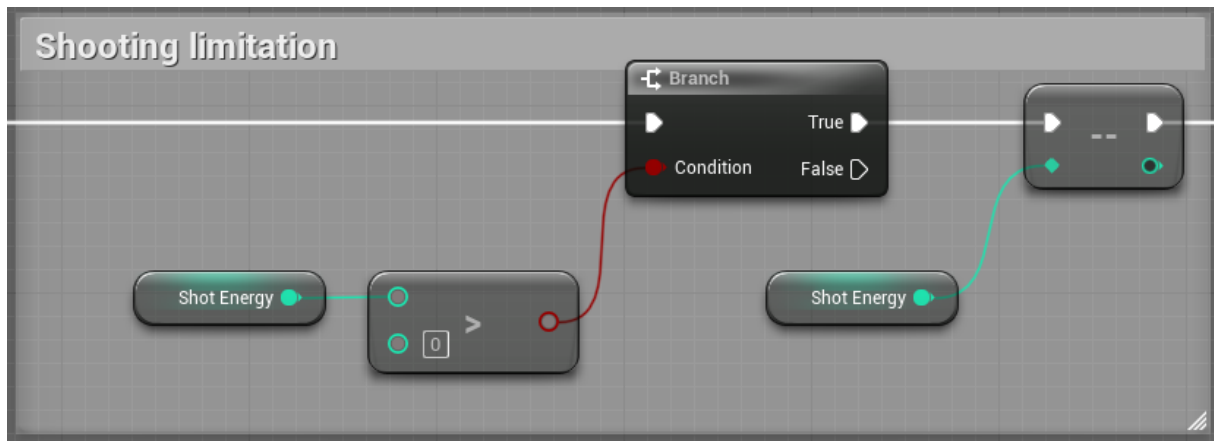


Abbildung 9 - Schuss Limitation

Quelle: Nina Hartmann, 2024

Anschließend wird die Variable *IsShooting* auf True gesetzt, damit später die Anzeige des korrekten Sprites erfolgen kann. Danach wird ein Projektil mittels der SpawnActor-Funktion erzeugt. Um das richtige Projektil zu spawnen, müssen die korrekten Position und Rotation festgelegt werden. Im linken Bild in Abbildung 10 ist zu sehen, dass die Projektile derzeit an der falschen Stelle erscheinen, da die Position des Charakters verwendet wird. Dieses Problem kann gelöst werden, indem eine Szene erstellt und an der gewünschten Stelle platziert wird, an der der Schuss erscheinen soll. Durch die Verwendung dieser Szene kann die Position und Rotation des Projektils korrekt eingestellt werden, wie in der Abbildung 10 rechts zu sehen ist.



Abbildung 10 - Schuss Location

Quelle: Nina Hartmann, 2024

4.2.2.2 Visuelle Darstellung

Die visuelle Repräsentation des Charakters erfolgt mittels eines animierten Sprites. Dieses setzt sich aus mehreren Bildern zusammen, die verschiedene Bewegungsphasen des Hauptcharakters darstellen, wie Abbildung 11 zeigt. Es konzentriert sich auf die zentralen Bewegungen, die für die Interaktion und Navigation des Charakters im Spiel entscheidend sind.



Abbildung 11 - Charakter-Sprites

Quelle: <https://ansimuz.itch.io/super-grotto-escape-pack>

Diese animierten Sprites werden dann in den AnimGraph integriert, um sie in die entsprechenden Animationen zu überführen. Ein Beispiel hierfür ist die Idle Animation. In Abbildung 12 ist zu sehen, wie mittels Blueprints die richtige Animation gewählt werden kann. Zunächst wird auf den Player verwiesen, und anschließend kann aus dem besitzenden Player heraus überprüft werden, ob die Taste für das Schießen der Waffe gedrückt wird. Wenn dies der Fall ist, soll die *Idle Shoot* Animation abgespielt werden. Andernfalls wird die einfache Idle Animation verwendet.

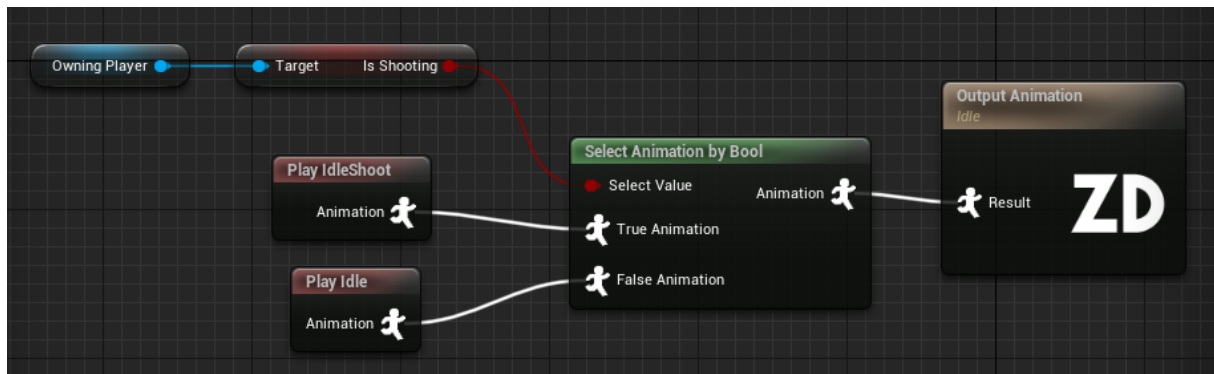


Abbildung 12 - Idle Animation

Quelle: Nina Hartmann, 2024

4.2.3 Mouse-Only-Charakter

Für die Interaktion mit dem zweiten Spieler ist es notwendig, die Position der Maus auf dem Bildschirm zu erfassen, um zu erkennen, welche Komponente angezeigt wird, wenn daraufgeklickt wird. Dies wird durch die Funktion *Convert Mouse Location To World Space* ermöglicht. Abbildung 13 zeigt, dass die gezogene Linie auf die Komponente zieht und auch diese durch das Rite Quadrat erkannt wird. Anschließend wird die Richtung mit einem Wert multipliziert, da der Weg von der Kamera zur Komponente gezogen werden muss. Abschließend wird diese berechnete Position zur Mausposition addiert, um den Start- und Endpunkt der Linie festzulegen.

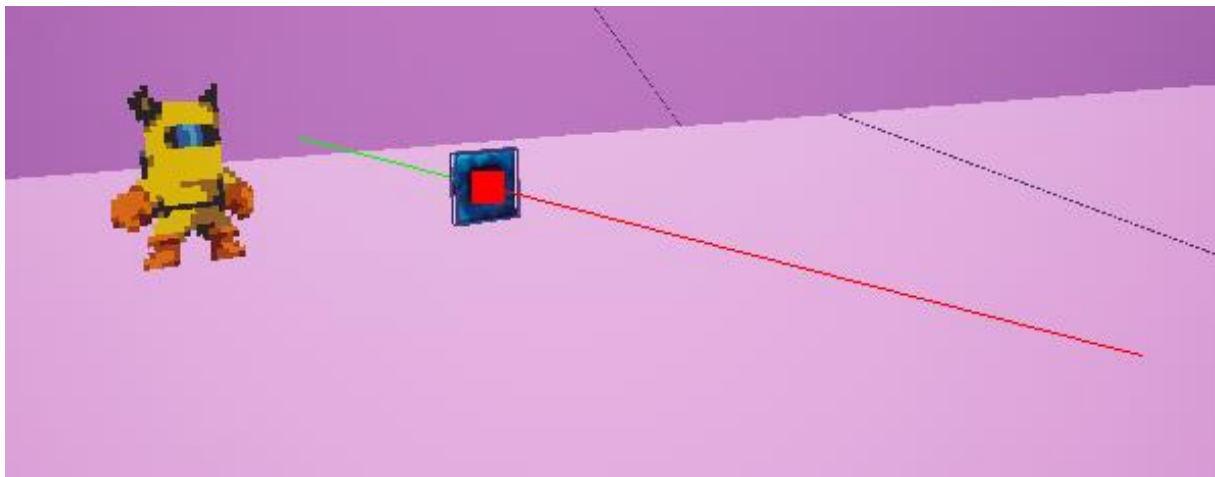


Abbildung 13 - Line Trace

Quelle: Nina Hartmann, 2024

Sobald eine Komponente identifiziert ist, wird der genaue Treffer durch *Break Hit Result* verfeinert, um den getroffenen Bereich zu bestimmen. Dadurch kann man feststellen, welche Komponente getroffen wurde. Anschließend wird überprüft, ob die getroffene Komponente beweglich ist. Ist sie nicht beweglich, wird keine Aktion ausgeführt. Ist diese aber beweglich, hängt die weitere Vorgehensweise davon

ab, ob sie physikalische Eigenschaften aufweist oder nicht. In diesem Projekt werden nur Komponenten ohne Physik zu bewegen.

Die Position der getroffenen Komponente wird in einem Vektor gespeichert und während jedes *On Event Tick* aktualisiert, um die Position alle Millisekunden zu verfolgen. Die Funktion *Set World Location* wird verwendet, um die Komponente zu bewegen. Wenn die Maustaste losgelassen wird, wird die Komponente an ihrer aktuellen Position fixiert, indem die Funktion *Release Component* aufgerufen wird.

Zusätzlich zur Möglichkeit, Blöcke zu bewegen, fehlt noch die Funktion, einen neuen Block zu spawnen. Dies wird durch einen Button ermöglicht, wie in Abbildung 14 in der rechten unteren Ecke dargestellt. Beim Klicken auf diesen wird die Position des Spielers ermittelt, da die Augen des Spielers typischerweise auf den Charakter gerichtet sind. Mit der *SpawnActor*-Funktion wird ein Block dann erstellt und damit der Block nicht im Spieler erscheint, wird für die Höhe noch ein Wert von 100 hinzugefügt, damit der Block über dem Spieler erscheint.

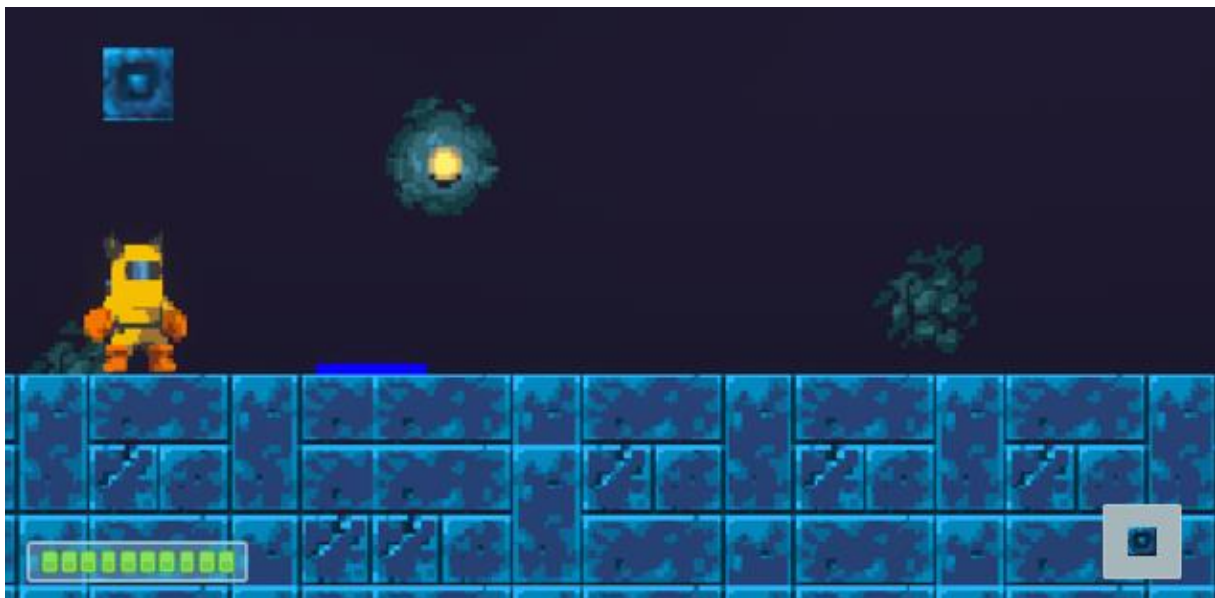


Abbildung 14 - Zweiter Spieler

Quelle: Nina Hartmann, 2024

4.2.4 Leben und Schaden System

Für die Implementierung der Lebensfunktionen in einem Spiel wird eine Actor-Component benötigt. Diese Komponente ermöglicht es, das Leben des Jump-and-Run-Charakters und der Gegner zu verwalten. Drei Hauptfaktoren sind erforderlich: maximale Gesundheit (Float), aktuelle Gesundheit (Float) und der Status, ob der Charakter besiegt wurde oder nicht (Boolean). Diese Faktoren werden innerhalb des

Actor Components implementiert und beim Start des Spiels (Event Begin Play) werden sie initialisiert, wobei die aktuelle Gesundheit auf die maximale Gesundheit gesetzt wird.

Abbildung 15 zeigt die Funktion Receive Damage. In dieser Funktion wird der erhaltene Schaden (Float) von der aktuellen Gesundheit subtrahiert. Durch die Verwendung der Funktion Clamp wird sichergestellt, dass die Gesundheit zwischen 0 und der maximalen Gesundheit bleibt, um ein Unterschreiten von 0 zu verhindern. Anschließend wird die aktuelle Gesundheit entsprechend aktualisiert, und mittels eines Verzweigungspunkts (Branch) wird überprüft, ob die aktuelle Gesundheit kleiner oder gleich 0 ist.

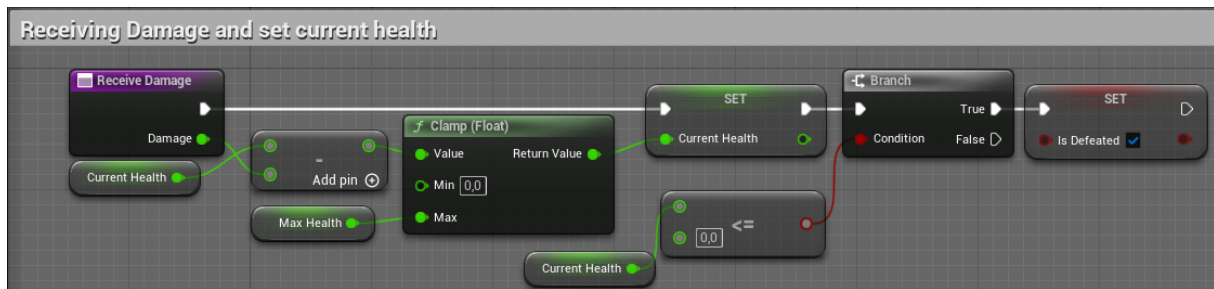


Abbildung 15 - Gesundheit Implementierung

Quelle: Nina Hartmann, 2024

Um Schaden als Spieler zu verursachen, müssen Projektile Schaden anrichten. Hierbei ist darauf zu achten, dass im Projektil-Actor die Kollisionen überlappen. Das relevante Event dafür ist *On Component Begin Overlap*. Wenn diese Kollisionen auftreten, geschieht Folgendes:

Jeder Charakter erhält ein Enum (Aufzählungstyp), um zwischen Spieler und Gegner zu unterscheiden, damit ein Charakter sich nicht selbst verletzen kann. Wenn das Projektil und die Kollision ein Gegner betreffen, wird die vorhandene Funktion *Apply Damage* aufgerufen. Diese Funktion ermöglicht es, dass beispielsweise im Gegner-Actor das Event *Any Damage* aufgerufen und an die Apply Damage Funktion angepasst wird. Anschließend wird festgelegt, wie viel Schaden das Projektil verursachen soll, standardmäßig 1 Schaden, und schließlich wird das Projektil zerstört, sobald es eine andere Kollision berührt.

Das Event Any Damage wird verwendet, um zu überprüfen, ob das Ziel bereits besiegt wurde. Falls ja, geschieht nichts. Falls das Ziel noch nicht besiegt ist, wird die oben beschriebene *Receive Damage* Funktion aufgerufen. Wenn das Ziel ein Spieler ist, wird das Event *Trigger Flash* ausgelöst, um anzuzeigen, dass der Spieler getroffen wurde und für eine kurze Zeit unverwundbar ist. Wenn das Ziel ein Gegner ist, blinkt dieser kurz gelb auf, um den Treffer zu zeigen. Falls der Spieler besiegt wird, wird die Funktion *Call Defeated* aufgerufen.

Damit Gegner dem Spieler Schaden zufügen können, benötigt man auch im Spieler-Actor ein On Component Begin Overlap Event, das die Funktion Apply Damage aufruft, um den Schaden zu verarbeiten.

Hier wurde auch die Funktion Knockback erstellt, um den Spieler bei der Berührung in die entsprechende Richtung zu schieben und so die Benutzeroberfläche zu verbessern.

4.2.5 Checkpoints und Respawn

Verliert der Spieler ein Leben wird er an einem Checkpoint wiederbelebt. Hierfür verwendet man einen Blueprint, der einen Trigger und ein dünnes Rechteck enthält.

Um sicherzustellen, dass der Checkpoint nur für den Spieler gilt, nutzt man das Event On Component Begin Overlap in Verbindung mit einem *Cast To*, um auf alle Elemente des Blueprints zugreifen zu können. Da ein Checkpoint nur einmal aktiviert werden sollte, wird die Methode *Do Once* verwendet. Sobald diese aktiviert wird, wird im Game Mode der aktive Checkpoint festgelegt und der Würfel auf Blau gesetzt, damit er gut sichtbar ist.

Im Game Mode existiert die Methode *Respawn Player*. Zunächst wird überprüft, ob ein aktiver Checkpoint vorhanden ist. Wenn ja, wird die Position und Rotation des aktiven Checkpoints genommen und mit *SpawnActor* der Spielercharakter erneut gespawnt. Es gibt jedoch das Problem, dass der gespawnte Charakter nur ein Actor ist und man ihn mit einer speziellen Funktion namens *possess* wieder steuern müssen.

Falls der Spieler keinen Checkpoint aktiviert hat, ist der Ablauf ähnlich, nur dass die Position und Rotation auf den Startpunkt des Spiels festgelegt werden.

4.2.6 Gegner

Es gibt insgesamt vier Gegner, von denen jeder eine bestimmte maximale Gesundheit hat, die geringer ist als die des Spielers. Beispielsweise hat die Krabbe nur 4 HP im Vergleich zu den 10 HP des Spielers. Die *Crab* verfügt über die spezielle Fähigkeit, sich von links nach rechts zu bewegen und umzukehren, wenn sie auf eine Wand oder Schlucht trifft. Abbildung 16 zeigt, wie es mit zwei Kollisionsboxen ermöglicht. Die erste Kollisionsbox befindet sich an der Schere der Krabbe und erkennt, wenn die Krabbe auf eine Wand trifft oder auf einen anderen Gegner stößt. In diesem Fall wird die Rotation auf 180 Grad gesetzt, und die Krabbe dreht sich um. Die zweite Kollisionsbox ist ein Eckendetektor, der überprüft, ob sich unter der Krabbe ein Objekt befindet. Wenn nicht, dreht sich die Krabbe ebenfalls um.



Abbildung 16 - Krabbe

Quelle: Nina Hartmann, 2024

Ein weiterer Gegner ist die *Bat*. Die Implementierung der Fledermaus war anspruchsvoller. Die Fledermaus soll sich von einem Punkt zum anderen bewegen können, wobei diese Punkte frei festgelegt werden können. Abbildung 17 zeigt die zwei Szenen verwendete Szenen, *TopPos* (links) und *BottomPos* (rechts). Zu Beginn des Spiels übernimmt die Fledermaus mit dem Event *Begin Play* die Positionen dieser beiden Punkte. Dann wird festgelegt, wie schnell sich die Fledermaus zwischen den beiden Positionen bewegen soll, dies wird mit der Funktion *Set Play Rate* erreicht. Das Event *Move Forward* bewegt die Fledermaus von einer Position zur anderen. Dabei wird eine Animationssequenz (*MoveAnim*) abgespielt, bei der die Fledermaus zum *TopPos* fliegt und am Ende zur *BottomPos* zurückkehrt. Dieser Ablauf wird mit einem Flip-Flop kontinuierlich wiederholt.

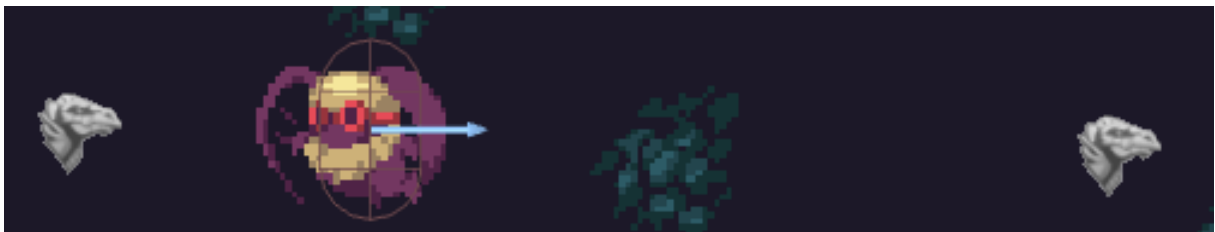


Abbildung 17 - Fledermaus Movement

Quelle: Nina Hartmann, 2024

Als nächstes ist der Gegner *Flyeye*. Dieser Gegner hat einen Erfassungsradius, den man in der Abbildung 18 sieht und sobald sich der Spieler innerhalb dieses Radius befindet, beginnt der Flyeye, den Spieler zu verfolgen, bis dieser besiegt ist. Der Flyeye hat weniger Gesundheit im Vergleich zu anderen Gegnern, da er als überraschender Gegner konzipiert ist, der sich unerwartet dem Spieler nähert. Die Verfolgung des Spielers wird durch ein selbst erstelltes Event (*MoveTowards*) ermöglicht, bei dem die Flyeye-Position mit der Funktion *Add Movement Input* aktualisiert wird. Die Funktion *Set Actor Rotation* wird verwendet, um die Ausrichtung des Gegner-Sprites anzupassen. Das Event *On Component Begin Overlap* wird genutzt, um den Spieler zu identifizieren, der sich in der Nähe des Flyeye befindet. Anschließend wird eine einfache Verfolgung mit der Funktion *Set Target Player* eingeleitet.

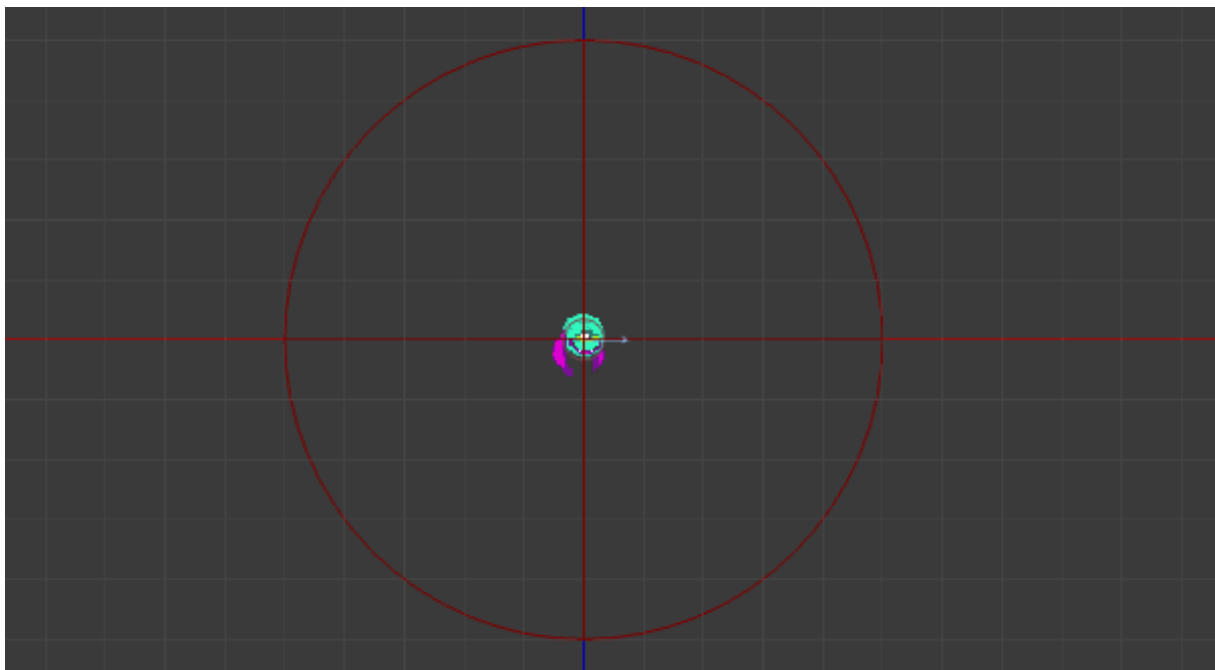


Abbildung 18 - Flyeye Radius

Quelle: Nina Hartmann, 2024

Der letzte Gegner ist der *Lizard*. Dieser Gegner ähnelt dem Spieler, da er Projektile abfeuert, sich jedoch nicht bewegen kann. Ähnlich wie beim Spieler gibt es eine Animationsquelle und einen Animationsgraph. Im Animationsgraph gibt es nur eine Idle-Animation, da sich der Lizard nicht bewegt und nach einer gewissen Zeit ein Projektil abfeuert. Die Funktion *Override Animation* wird verwendet, um nach einer bestimmten Zeit die Schussanimation auszulösen. In Abbildung 19 ist zu sehen, wie mithilfe von Nodes gearbeitet wird, wobei das Projektil beim vierten Frame gespawnt wird.

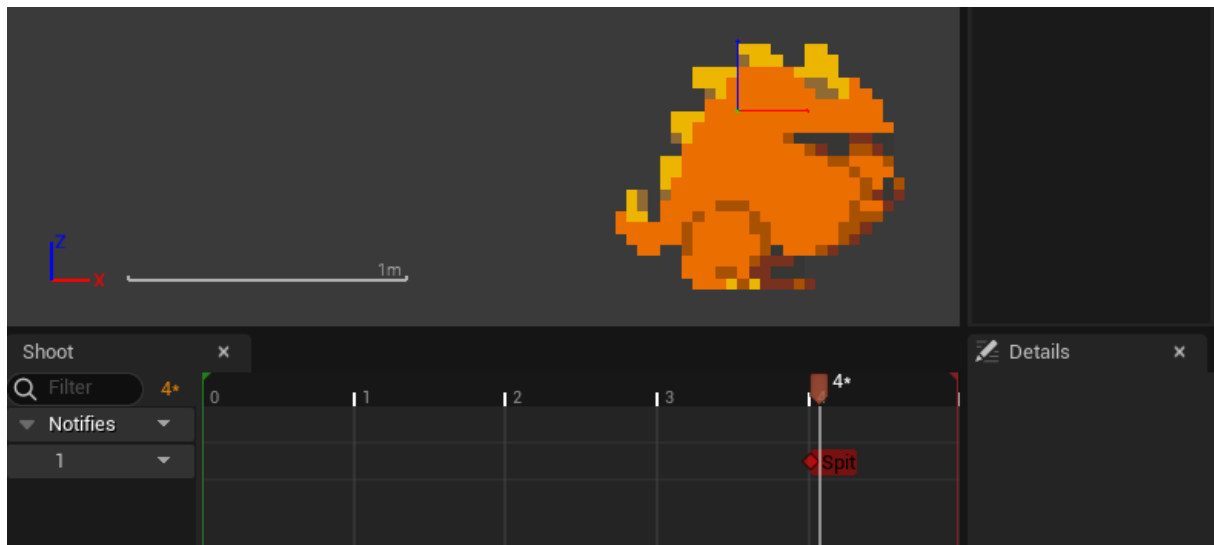


Abbildung 19 - Lizard Schuss

Quelle: Nina Hartmann, 2024

4.2.7 Level Design

Die spielbare Umgebung wird mittels Tilemaps gestaltet, bei dieser werden verschiedene TileSets, die eine Vielzahl von Elementen umfasst, eingefügt. Es ist wichtig, die TileSize richtig einzustellen und die Kollisionen entsprechend zu konfigurieren. Auf der linken Seite der Abbildung 20 ist ein Beispiel für ein TileSet zu sehen. Die TileSize wurde auf 16 festgelegt, und die einzelnen Kollisionseinstellungen, wie auf der rechten Seite der Abbildung 20 gezeigt, wurden vorgenommen.

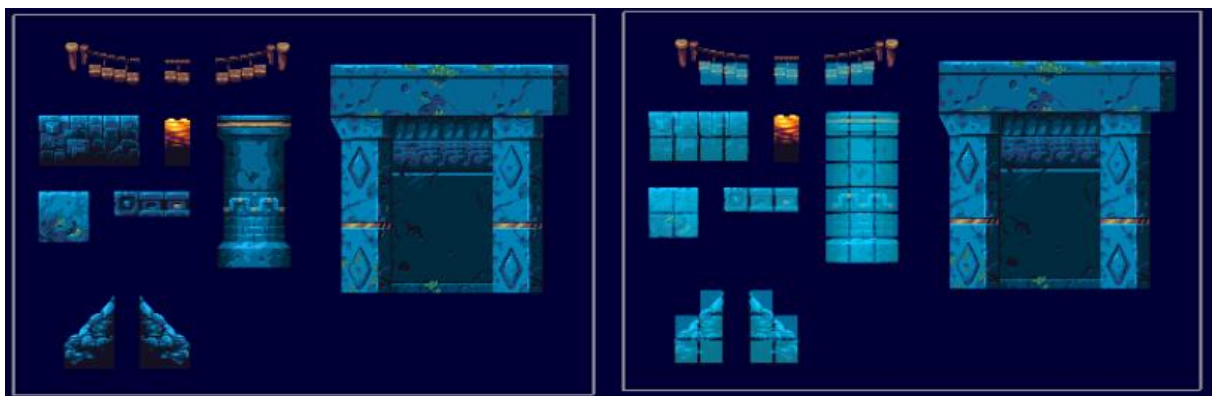


Abbildung 20 - Tile Sets

Quelle: Nina Hartmann, 2024

Für die Erstellung des Levels wird die TileMap verwendet, in der das Level nach Belieben gestaltet werden kann. Vor der Erstellung des Levels wurde eine Skizze angefertigt, wie in Abbildung 21 zu erkennen, um zu planen, wie das Rätsel gelöst werden kann, wo die Gegner platziert werden sollen und wie das

Level strukturiert sein soll. Der Anfang zeigt, den ersten Teil des Levels mit Anweisungen für den Spieler zum Springen, Bewegen und Schießen und hier soll auch kurz erklärt werden, wie der Block plaziert werden kann. Der Rest ist die Fortsetzung des Levels und führt zum Ziel.

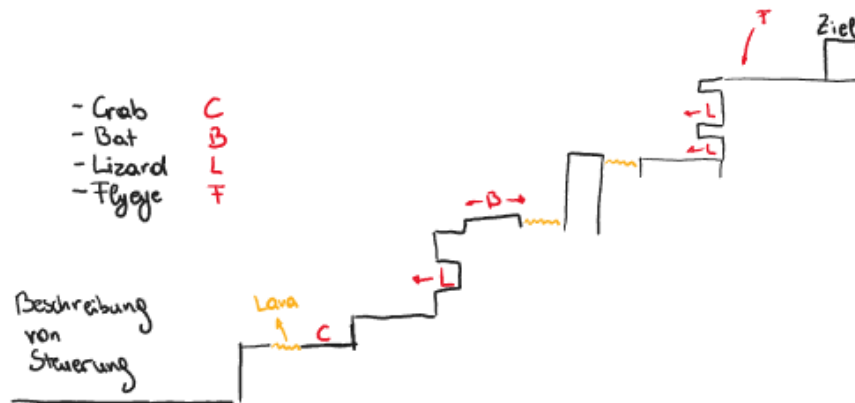


Abbildung 21 - Level-Design

Quelle: Nina Hartmann, 2024

Die Umsetzung der Skizze in die TileMap erfordert viel Platz und Vorarbeit. Im Vorfeld muss die Größe der TileMap festgelegt werden, da sie nachträglich nur nach unten und rechts erweitert werden kann. Ist die Höhe der TileMap zu gering, erfordert die Anpassung erheblichen Aufwand. Daher ist es wichtig, im Voraus sorgfältig zu planen, wie groß die TileMap sein soll. Das Erstellen einer TileMap in Unreal Engine kann umständlich sein, da die einzelnen Tiles einzeln platziert werden müssen. Es ist auch möglich, mehrere Ebenen zu erstellen, wie beispielsweise einen Hintergrund, um die Umgebung zu gestalten.

4.2.8 Sounds

Das optionale Feature der Integration von Sound wurden Geräusche hinzugefügt, die bestimmte Bewegungen begleiten und die Spielerfahrung dynamischer gestalten. Auch für die Gegner wurde Sound hinzugefügt, um es dem Spieler zu erleichtern, sich auf das Kommende vorzubereiten.

Die Sounds wurden auf unterschiedliche Weise implementiert, wobei zwei Methoden verwendet wurden. Die erste Methode besteht darin, zu jedem Sprite, der abgespielt wird, eine Node hinzuzufügen, die einen Sound wiedergibt. Dies ergibt Sinn beispielsweise beim Laufen des Charakters, da dabei mehrere Bilder hintereinander abgespielt werden und der Sound jedes Mal abgespielt wird, wenn der Fuß des Charakters den Boden berührt. Diese Methode wurde beim Spielercharakter angewendet und auch beim Gegner Lizard, der während seiner Schussanimation ebenfalls mehrere Bilder hat und genau dann, wenn er den Schuss abfeuert, einen Sound abspielen soll.

Die zweite Methode erfolgt direkt in den Blueprints selbst. Wenn das Event *Input Actor* für den Charakter ausgelöst wird, wird auch ein entsprechender Sound abgespielt. Für die Gegner ist es wichtig, dass die Sounds abgespielt werden, wenn sich der Charakter in der Nähe befindet, und dass sie immer lauter werden, je näher man dem Gegner kommt. Hierfür wird die Funktion *Spawn Sound at Location* verwendet, wobei die Position des Actors genutzt wird, um den Sound an dieser Stelle abzuspielen und die Lautstärke entsprechend anzupassen. [13]

5 Tests und Evaluation

Dieses Kapitel beschreibt die Evaluierung des entwickelten Spiels. Das Spiel wurde einem umfassenden Testprozess unterzogen, um seine Funktionalität, Benutzerfreundlichkeit und Unterhaltungswert zu bewerten.

Die Evaluierung erfolgte insbesondere durch eine gezielte Umfrage unter ausgewählten Probanden, die das Spiel gespielt haben. Dabei wurden insgesamt 10 Personen, jeweils in Zweiergruppen, gebeten, das Spiel zu testen und im Anschluss detaillierte Rückmeldungen zu geben. Diese Umfrage diente dazu, sowohl quantitative als auch qualitative Daten zu sammeln, um die Leistung und die Reaktionen der Spieler auf das Spiel zu analysieren.

In diesem Kapitel wird die Methodik dieser Testphase erläutert. Dabei wird auf die Auswahl der Testgruppe, die Kriterien für die Bewertung, die Durchführung der Tests sowie die Auswertung der Ergebnisse eingegangen. Der Schwerpunkt liegt dabei darauf, wie die gesammelten Daten dazu beitragen, Erkenntnisse über das Spielerlebnis zu gewinnen und potenzielle Verbesserungen oder Anpassungen für zukünftige Entwicklungen aufzuzeigen.

Der Einsatz von Test- und Evaluationsmethoden spielt eine entscheidende Rolle, um die Qualität und den Erfolg des Spiels zu bewerten und sicherzustellen, dass es den Erwartungen und Bedürfnissen der Zielgruppe entspricht. Die Ergebnisse dieser Tests dienen nicht nur als Grundlage für die Unterstützung dieser Arbeit, sondern auch einen wichtigen Beitrag zur Weiterentwicklung und Optimierung des Spiels leisten.

5.1 Testphasen und -szenarien

Die Testphasen des Spiels wurden in mehreren klar definierten Testphasen durchgeführt, um eine umfassende Bewertung der Funktionalität und Benutzererfahrung sicherzustellen.

1. Grundlegende Spielmechanik und Interaktion:

Es wurde die Kernsteuerung und die Interaktionen zwischen den Spielcharakteren überprüft, um sicherzustellen, dass das Spiel reibungslos läuft und keine technischen Probleme auftreten.

2. Benutzerfreundlichkeit und Navigation:

Die Grafikqualität, Benutzerführung und Navigation im Hauptlevel wurden eingehend untersucht, um die Zugänglichkeit des Spiels zu bewerten und sicherzustellen, dass die Spielerführung angemessen ist.

3. Spielverhalten und Kooperation:

Besonderes Augenmerk lag auf dem Verhalten der Spieler und ihrer Zusammenarbeit im Spiel. Dabei wurden die Möglichkeiten und Herausforderungen der Kooperation zwischen den Charakteren analysiert.

Die Testkriterien, die zur Bewertung des Spiels herangezogen wurden, umfassen:

- Grafikqualität und ästhetische Gestaltung: Beurteilung der visuellen Qualität und Ästhetik des Spiels.
- Spielmechanik und Interaktivität: Überprüfung der Mechanik und Interaktionen zwischen den Spielcharakteren.
- Benutzerführung und Kooperation: Bewertung der Benutzerführung und der Effektivität der Kooperationsmechanismen.
- Technische Stabilität und Performance: Testen der technischen Leistung und Stabilität des Spiels auf dem genutzten Gerät.

Als Testumgebung diente ein HP ENVY x360 Convertible 15-ee0xxx. Alle Tests wurden ausschließlich auf diesem Gerät durchgeführt und getestet, um die Performance und Kompatibilität auf einem typischen Endnutzergerät zu gewährleisten.

Die Ergebnisse dieser Testphasen bieten Einblicke in die Leistung und Benutzererfahrung des Spiels, was wiederum dazu beiträgt, potenzielle Verbesserungsbereiche zu identifizieren und die Spielqualität zu optimieren.

5.2 Evaluation

Die Evaluierung wurde anhand der definierten Testphasen und Szenarien, wie im vorherigen Kapitel erwähnt, durchgeführt, um die Funktionalität, Benutzerfreundlichkeit und Interaktionen zwischen den Spielcharakteren zu bewerten. Unter den befragten Personen befanden sich vier zufällig ausgewählte Menschen, die sich in der FH Vorarlberg aufhielten, sowie ein älteres Paar und zwei gezielt ausgewählte Paare, die sich gut mit Spielen auskennen. Insgesamt wurden somit zehn Personen befragt.

5.2.1 Ergebnisse der Testphasen und Szenarien

1. Grundlegende Spielmechaniken und Interaktionen

Die Überprüfung der Kernsteuerung und Spielmechanik ergab überwiegend positive Ergebnisse. Die Steuerung des Jump-and-Run-Charakters erhielt eine Bewertung von 4,4 von 5 Punkten, während der Mouse-Only-Charakter etwas niedrigere Bewertungen mit 4 von 5 Punkten erzielte. Während der Tests wurde das Spiel auf Bugs und Stabilitätsprobleme getestet. Etwa 40 % der Tester berichteten von Bugs, wie zum Beispiel Verzögerungen bei zu vielen gespawnten Blöcken oder unzureichender Leistung aufgrund der Testumgebung. Einige Tester bemerkten, dass sie Gegner mit dem Block bewegen konnten

oder dass das Spiel abstürzte, wenn sich ein Spieler auf den Block stellte. Zudem wurde der Wunsch geäußert, mehr Varianten von Blöcken und zusätzliche Level hinzuzufügen, da das Spiel, als sehr unterhaltsam und mit einer interessanten Spielidee bewertet wurde. Auch die Kooperationsidee zwischen den Spielern wurde positiv hervorgehoben, obwohl einige Tester meinten, dass es zu viele Leben gab.

2. Benutzerfreundlichkeit und Navigation

Die Benutzerfreundlichkeit und Navigation wurden positiv bewertet. Die Grafikqualität wurde als ansprechend empfunden und die Benutzerführung im Hauptlevel war angemessen gestaltet. Spieler lobten besonders die Einführung am Anfang des Spiels, die als einfach verständlich gelobt wurde. Die Navigation innerhalb der Level wurde als einfach wahrgenommen, da es nur eine Richtung zum Ziel gab und Spieler sich gut orientieren konnten. Das Block-Spawning wurde ebenfalls positiv bewertet, obwohl angemerkt wurde, dass weitere Blöcke möglicherweise alternative Lösungen erfordern würden, abhängig von der gewünschten Spielkomplexität.

3. Spielverhalten und Kooperation

Das Verhalten der Spieler und ihre Zusammenarbeit wurden positiv bewertet. Die Kooperationsmechanik zwischen den Charakteren wurde als effektiv erlebt, um Hindernisse zu überwinden und strategische Vorteile zu erzielen. Allerdings wurde auch festgestellt, dass Spieler den Block manchmal dazu benutzten, um einander zu stören oder zu blockieren, was dem Spiel eine gewisse Komplexität und Spaß verlieh. Spieler mussten sich gut abstimmen und strategisch vorgehen, da das Spiel stark von Teamarbeit abhing. Einige Spieler fanden es jedoch herausfordernd, die Blöcke richtig zu platzieren, da sie die Sprünge des anderen Spielers nicht genau vorhersehen konnten.

Zusammenfassend war das Spiel für die erste Testphase erfolgreich. Es wurden zwar einige Bugs festgestellt, jedoch nichts Unlösbares. Die Spielidee erhielt eine durchweg positive Bewertung von 5 von 5 Punkten und wurde von den Testern sehr empfohlen. Viele glauben, dass das Spiel Potenzial hat, zu einem vollständigen Spiel weiterentwickelt zu werden. Verbesserungspotenzial gibt es bei den Level. Mit mehr und herausfordernderen Levels könnte das Spiel noch unterhaltsamer sein, da das aktuelle Level einfach gestaltet wurde, um eine breite Altersgruppe anzusprechen.

6 Zusammenfassung und Ausblick

Dieses Kapitel fasst die Ergebnisse dieser Arbeit zusammen und gibt einen Ausblick auf mögliche weiterführende Arbeiten.

6.1 Zusammenfassung

Diese Arbeit beschäftigte sich mit dem Thema des kollaborativen Rätsellösens in Spielen in einer sich verändernden, dynamischen Umgebung. Es wurde exemplarisch ein kollaboratives 2D-Spiel Double Trouble entwickelt, welches von zwei Spielern simultan gespielt werden kann. Hauptziel ist dabei kollaborativ Rätsel zu bearbeiten und so Fortschritt im Spiel zu erzielen.

Die Mussziele wurden alle erfolgreich erreicht. ¹Die Implementierung beider Hauptcharaktere verlief erfolgreich: Der Jump-and-Run-Charakter beherrscht das Springen, Rennen und Schießen. Mithilfe von AnimGraph wird der passende Sprite für die Animation abgespielt, und die Bewegungs-, Lebens- und Schadenssysteme wurden korrekt implementiert. Der Mouse-Only-Charakter wird ausschließlich mit der Maus gesteuert und kann Blöcke spawnen und bewegen, um dem Jump-and-Run-Charakter zu helfen. Dies wurde durch eine Funktion erreicht, die die Position der Maus erfasst und es ermöglicht, durch Klicken auf die Komponente den Block zu erkennen und zu verschieben. Die Erstellung des Hauptlevels wurde als erfolgreich angesehen, da es alle Spielmechaniken gut demonstriert und ausprobieren lässt. Es bietet eine umfassende Möglichkeit für Spieler, die Funktionalitäten des Spiels zu testen und zu erleben.

Zusätzlich wurden einige optionale Ziele verfolgt, um das Spielerlebnis zu verbessern. Ein Einführungslevel wurde erstellt, um den Spielern das Spiel und die Gegner näherzubringen, indem man den Spielern erklärt, wie die Steuerung funktioniert und wie man einen Block spawnen und bewegen kann. Soundeffekte wurden für den Spieler und die Gegner hinzugefügt, um das Spielerlebnis zu verbessern und die Spieler zu informieren, welcher Gegner als nächstes auftaucht. Auch eine Gesundheitsanzeige wurde implementiert, um den Spielern ihren aktuellen Gesundheitsstatus anzuzeigen.

Das Ergebnis dieses Projekts ist ein 2D-Koop-Spiel, das darauf abzielt, den Spielern Spaß zu bereiten und sie zur Zusammenarbeit durch unterschiedliche Rätsel, starke Gegner und herausfordernde Level zu ermutigen. Es wurde darauf geachtet, ein vielseitiges und unterhaltsames Spielerlebnis zu schaffen, das Teamarbeit und strategisches Denken fördert. ²

¹ https://github.com/nha5226/Bachelorarbeit_Hartmann

² <https://www.youtube.com/watch?v=5hdFUs3US0k>

6.2 Ausblick

Mögliche Weiterentwicklungen betreffen den Ausbau des Spiels in Bezug auf die Integration von weiteren Leveln und der Integration von Online-Features, um auch eine Online-Zusammenspiel zu ermöglichen

Für die Zukunft des Spiels sind weitere Elemente geplant, die das Spielerlebnis bereichern sollen. Dazu gehört die Implementierung neuer Blöcke, die sinnvoll in das Spiel integriert werden, um den Spielern mehr Spaß und strategische Möglichkeiten zu bieten. Zusätzlich sollen mehr Level erstellt werden, um die Vielfalt und den Umfang des Spielerlebnisses zu erweitern.

Durch die Erweiterung des Spiels mit neuen Inhalten und Features wird die Bindung der Spieler gestärkt und das Spiel zu einem unterhaltsamen und interaktiven Erlebnis für alle werden.

7 Lizenzen

Die verwendeten Grafiken in dieser Bachelorarbeit wurden unter einer Creative Commons-Lizenz veröffentlicht, die die Nutzung dieser Assets ermöglicht. Es ist erfreulich und lobenswert, dass talentierte Künstler und Kreative wie Luis Zuno und die Webseite Kenny solche Assets frei zur Verfügung stellen.

Die Charaktere, Hintergründe und Gegner wurden von Luis Zuno erstellt.³ Die UI-Assets stammen von der Webseite Kenny.⁴

Es ist wichtig, anzuerkennen, dass diese Künstler ihre Arbeit unter Bedingungen veröffentlicht haben, die die Verwendung und Anpassung ihrer Grafiken in akademischen und kreativen Projekten wie dieser Bachelorarbeit ermöglichen. Dies unterstreicht die Bedeutung offener Ressourcen und der Gemeinschaft von Entwicklern und Designern, die durch ihre Arbeit innovative Projekte unterstützen.

³ <https://ansimuz.itch.io/super-grotto-escape-pack>

⁴ <https://kenney.nl/assets/ui-pack-space-expansion>

Glossar

Player	Der Spielercharakter im Spiel, der vom Spieler gesteuert wird. In der Unreal Engine wird der Spieler oft durch eine sogenannte Player Character oder Pawn repräsentiert.
Cooperative Game	
Sprite	Ein 2D-Bild oder eine Grafik, die in der Spielwelt dargestellt wird. In der Unreal Engine können Sprites verwendet werden, um z. B. HUD-Elemente, Icons oder Effekte darzustellen.
Blueprint	Ein visuelles Scripting-System in der Unreal Engine, das es Entwicklern ermöglicht, das Verhalten von Objekten und Charakteren zu definieren, ohne Code schreiben zu müssen. Blueprints sind eine Art visuelle Programmierung.
Movement	Die Bewegung von Objekten oder Charakteren in der Spielwelt. In der Unreal Engine kann die Bewegung durch Physik, Animationen oder Skripting gesteuert werden.
Checkpoint	Ein spezieller Punkt in der Spielwelt, an dem der Spieler seinen Fortschritt speichern kann. Wenn der Spieler stirbt oder das Spiel verlässt und zurückkehrt, kann er am letzten Checkpoint wieder starten.
Game Mode	Eine Klasse in der Unreal Engine, die das Regelwerk und die Logik des Spiels definiert. Der Game Mode legt fest, wie das Spiel startet, welche Regeln gelten und wie Spieler miteinander interagieren können.
Respawn	Der Vorgang, bei dem ein Spieler oder eine Entität nach dem Tod oder dem Verlassen eines Bereichs wieder in die Spielwelt zurückkehrt. Der Respawn-Punkt wird normalerweise vom Spiel festgelegt, z. B. beim Erreichen eines Checkpoints.
Spawn	Der Vorgang, bei dem Objekte, Gegner oder Spielercharaktere in der Spielwelt erzeugt oder platziert werden.
Sound	Audioelemente, die in der Spielwelt abgespielt werden, um Atmosphäre zu erzeugen oder Ereignisse zu signalisieren. In der Unreal Engine können Soundeffekte und Musik über Audiosysteme eingebunden und gesteuert werden.

Literaturverzeichnis

- [1] J. Schell, *The art of game design: a book of lenses*, 3rd edition. Boca Raton London New York: CRC Press, Taylor & Francis Group, 2020.
- [2] „Unity“, Unity. Zugriffen: 14. Juni 2024. [Online]. Verfügbar unter: <https://unity.com/>
- [3] „Unreal Engine“, Unreal Engine. Zugriffen: 14. Juni 2024. [Online]. Verfügbar unter: <https://www.unrealengine.com/de/home>
- [4] G. Engine, „Godot Engine - Free and open source 2D and 3D game engine“, Godot Engine. Zugriffen: 14. Juni 2024. [Online]. Verfügbar unter: <https://godotengine.org/>
- [5] U. Technologies, „Wondering what Unity is? | Unity“. Zugriffen: 30. April 2024. [Online]. Verfügbar unter: <https://unity.com/our-company>
- [6] „The Best Games Made With Unity Game Engine“, Kevuru Games. Zugriffen: 30. April 2024. [Online]. Verfügbar unter: <https://kegurugames.com/blog/top-games-made-with-unity-game-engine/>
- [7] G. Engine, „Showcase | Godot“, Godot Engine. Zugriffen: 1. Mai 2024. [Online]. Verfügbar unter: <https://godotengine.org/showcase/>
- [8] „Unreal Tournament: Game of the Year Edition bei Steam“. Zugriffen: 30. April 2024. [Online]. Verfügbar unter: https://store.steampowered.com/app/13240/Unreal_Tournament_Game_of_the_Year_Edition/
- [9] J. Von Neumann und O. Morgenstern, *Theory of games and economic behavior*, 60. anniversary ed., 4. printing, and 1. paperback printing. in A Princeton classic edition. Princeton, NJ: Princeton University Press, 2007.
- [10] M. J. Albizuri und P. Sudhölter, „Characterizations of the core of TU and NTU games with communication structures“, *Soc. Choice Welf.*, Bd. 46, Nr. 2, S. 451–475, Feb. 2016, doi: 10.1007/s00355-015-0924-1.
- [11] V. Torra, Y. Narukawa, und M. Inuiguchi, *Modeling Decisions for Artificial Intelligence: 6th International Conference, MDAI 2009, Awaji Island, Japan, November 30-December 2, 2009. Proceedings.* in Lecture notes in artificial intelligence, no. 5861. Berlin Heidelberg: Springer-Verlag, 2009.
- [12] „PaperZD Documentation – Critical Failure Studio“. Zugriffen: 6. Mai 2024. [Online]. Verfügbar unter: <https://www.criticalfailure-studio.com/paperzd-documentation/>
- [13] N. Hartmann, „Sound Effekts - YouTube“. Zugriffen: 10. Mai 2024. [Online]. Verfügbar unter: <https://www.youtube.com/watch?v=1RuoBQViAWk>
- [14] *A theory of fun for game design*, 2nd edition. Sebastopol, CA: O'Reilly, 2014.

- [15] Cobra Code, „Make a 2D Action Platformer in Unreal Engine 5“, Udemey. Zugriffen: 2. April 2024. [Online]. Verfügbar unter: <https://www.udemy.com/course/ue-2d-action-platformer/>
- [16] B. Peleg und P. Sudhölter, Introduction to the Theory of Cooperative Games. Springer Science & Business Media, 2007.
- [17] „Unreal Engine 5.4 Documentation | Epic Developer Community“. Zugriffen: 30. April 2024. [Online]. Verfügbar unter: <https://dev.epicgames.com/documentation/en-us/unreal-engine/unreal-engine-5-4-documentation>
- [18] „Die Evolution der Koop-Spiele: Von den späten 1970ern bis heute“, PC GAMES. Zugriffen: 30. April 2024. [Online]. Verfügbar unter: <https://www.pcgames.de/Panorama-Thema-233992/Specials/gute-Koop-Spiele-Geschichte-1369588/>
- [19] „Introduction to Blueprints | Unreal Engine 4.27 Documentation“. Zugriffen: 30. April 2024. [Online]. Verfügbar unter: <https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Blueprints/GettingStarted/>
- [20] „Choosing a game engine: Unity vs Unreal Engine vs Godot“. Zugriffen: 1. Mai 2024. [Online]. Verfügbar unter: <https://cyberglads.com/making-cyberglads-1-choosing-a-game-engine.html>

Anhang

Inhaltsverzeichnis

A „Umfrage zum 2D-Koop-Spiel“ Fragebogen	45
B „Umfrage zum 2D-Koop-Spiel“ Ergebnisse	48

Umfrage zum 2D-Koop-Spiel

1. Wie alt sind Sie? *

- ☐ 0-18
- ☐ 19-30
- ☐ 31-60
- ☐ 60+

2. Haben Sie bereits Erfahrungen mit ähnlichen Koop-Spielen? *

- ☐ Ja
- ☐ Nein

3. Wie würden Sie die Steuerung des Jump-and-Run Charakters bewerten? *



4. Wie einfach oder schwer war es, Blöcke als der Mouse-Only-Charakter zu platzieren? *



5. Mussten Sie bestimmte Strategien entwickeln, um effektiv als Team zu arbeiten? *



6. Gab es Momente in denen Sie sich Aufgrund unterschiedlicher Rollen mit Ihrem Mitspieler abstimmen mussten? *

☐ Ja

☐ Nein

7. Wie unterhaltsam fanden Sie das Spielkonzept? *



8. Würden Sie das Spiel anderen Spielern empfehlen? *



9. Wie wichtig war die Kommunikation mit Ihrem Mitspieler, um im Spiel voranzukommen? *



10. Haben Sie festgestellt, dass eine gute Zusammenarbeit erforderlich war, um das Ziel zu erreichen? *

☐ Ja

☐ Nein

11. Was waren die größten Herausforderungen, die Sie beim Zusammenarbeiten mit Ihrem Mitspieler erlebt haben? *

Ihre Antwort eingeben

12. Hat das Spiel Ihre Fähigkeit zur Zusammenarbeit oder Problemlösung auf die Probe gestellt? *

- ☐ Ja
- ☐ Etwas
- ☐ Nein

13. Gab es technische Probleme während des Spielens? (z.B. Absturz, verschwinden der Maus,...) *

- ☐ Ja
- ☐ Nein

14. Wenn Ja, welche?

Ihre Antwort eingeben

15. Gibt es etwas, das Sie an diesem Spiel besonders mochten? *

Ihre Antwort eingeben

16. Was könnte verbessert werden, um das Spielerlebnis zu optimieren? *

Ihre Antwort eingeben

17. Gibt es Funktionen oder Mechaniken, die die Koop-Erfahrung verbessern könnten? *

Ihre Antwort eingeben


18. Haben Sie sonstiges Feedback, dass Sie teilen möchten? *

Ihre Antwort eingeben

B „Umfrage zum 2D-Koop-Spiel“ Ergebnisse

1. Wie alt sind Sie? (0 Punkt)

[Weitere Details](#)


 Einblicke

0-18	0
19-30	7
31-60	3
60+	0



2. Haben Sie bereits Erfahrungen mit ähnlichen Koop-Spielen? (0 Punkt)

[Weitere Details](#)


 Einblicke

Ja	6
Nein	4

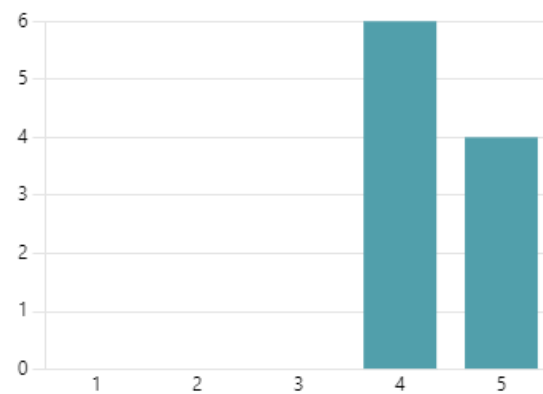


3. Wie würden Sie die Steuerung des Jump-and-Run Charakters bewerten? (0 Punkt)

[Weitere Details](#)

 Einblicke

4.40
Durchschnittliche Bewertung



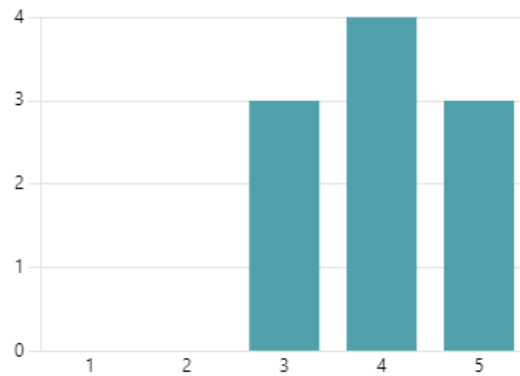
4. Wie einfach oder schwer war es, Blöcke als der Mouse-Only-Charakter zu platzieren? (0 Punkt)

[Weitere Details](#)

 Einblicke


4.00

Durchschnittliche Bewertung



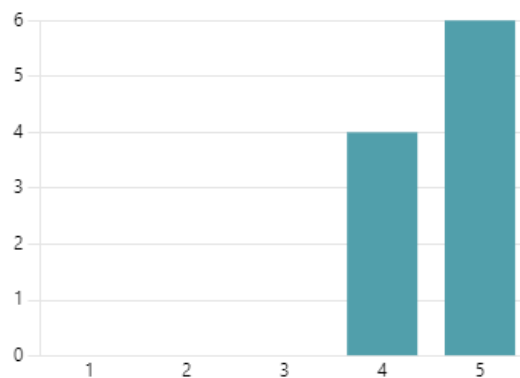
5. Mussten Sie bestimmte Strategien entwickeln, um effektiv als Team zu arbeiten? (0 Punkt)

[Weitere Details](#)

 Einblicke


4.60


Durchschnittliche Bewertung



6. Gab es Momente in denen Sie sich Aufgrund unterschiedlicher Rollen mit Ihrem Mitspieler abstimmen mussten? (0 Punkt)

[Weitere Details](#)

 Einblicke


 Ja 10

 Nein 0



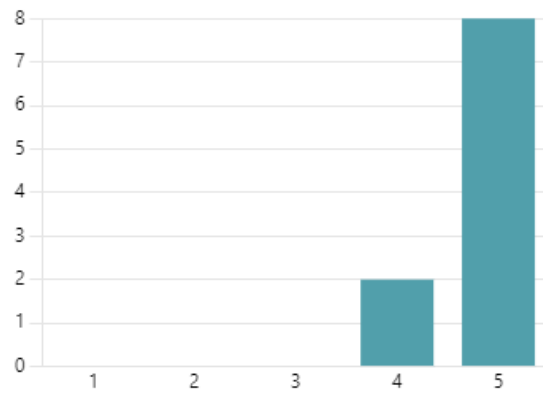
7. Wie unterhaltsam fanden Sie das Spielkonzept? (0 Punkt)

[Weitere Details](#)

 Einblicke

4.80

Durchschnittliche Bewertung



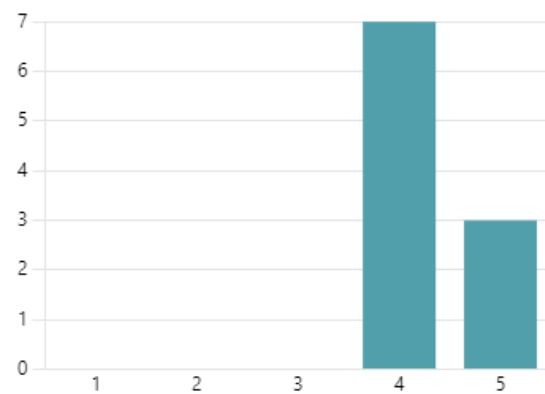
8. Würden Sie das Spiel anderen Spielern empfehlen? (0 Punkt)

[Weitere Details](#)

 Einblicke


4.30

Durchschnittliche Bewertung



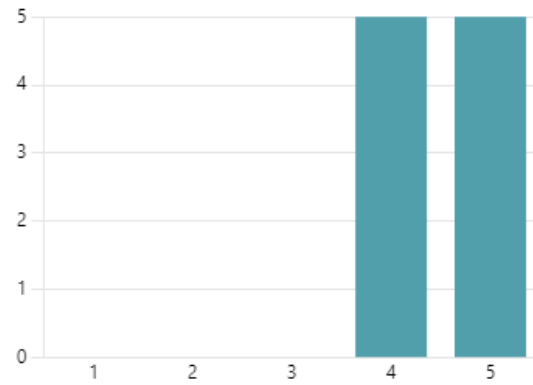
9. Wie wichtig war die Kommunikation mit Ihrem Mitspieler, um im Spiel voranzukommen? (0 Punkt)

[Weitere Details](#)

 [Einblicke](#)

4.50


Durchschnittliche Bewertung



10. Haben Sie festgestellt, dass eine gute Zusammenarbeit erforderlich war, um das Ziel zu erreichen?

(0 Punkt)

[Weitere Details](#)

 [Einblicke](#)

● Ja 10
● Nein 0



11. Was waren die größten Herausforderungen, die Sie beim Zusammenarbeiten mit Ihrem Mitspieler erlebt haben?

10 Antworten

ID ↑	Name	Antworten
1	anonymous	Koordination
2	anonymous	Die Blöcke richtig zu platzieren, wenn man zB über Lava springen muss
3	anonymous	Inkompetenz des Mitspielers
4	anonymous	Richtige Kommunikation mit dem Mitspieler
5	anonymous	Der fliegende Gegner der einen verfolgt hat ziemlich angst gemacht und mein Mitspieler konnte nicht helfen
6	anonymous	Auf die Blöcke zu springen, da der Mitspieler nicht so kooperativ war
7	anonymous	Mit dem Mitspieler richtig zu Kooperieren, da man den springenden Spieler auch behindern kann.
8	anonymous	die Stelle mit den Gegner die Spucken
9	anonymous	Gute Mitspieler haben
10	anonymous	Nicht in die Lava zu springen

12. Hat das Spiel Ihre Fähigkeit zur Zusammenarbeit oder Problemlösung auf die Probe gestellt? (0 Punkt)

[Weitere Details](#)

[Einblicke](#)



13. Gab es technische Probleme während des Spielens? (z.B. Absturz, verschwinden der Maus,...) (0 Punkt)

[Weitere Details](#)

[Einblicke](#)



14. Wenn Ja, welche?

ID ↑	Name	Antworten
1	anonymous	Absturz, des Spiels
2	anonymous	Blöcke sind ohne mausklick gespawnd

15. Gibt es etwas, das Sie an diesem Spiel besonders mochten?

10 Antworten

ID ↑	Name	Antworten
1	anonymous	Der Spaßfaktor
2	anonymous	Zwei Spieler variante
3	anonymous	Multiplayer, Retro Grafik
4	anonymous	Die Spielidee
5	anonymous	Es ist eine coole Spielidee und bin schon darauf gespannt ob es weiter geht oder veröffentlicht wird.
6	anonymous	Spiel hat Potenzial und wäre cool wenn es mehr level hätte
7	anonymous	cooles Spiel, cooler Artstyle
8	anonymous	War lustig den anderen Spieler ab und zu zu blocken und zu nerven
9	anonymous	Teamarbeit war nötig um weiter zu kommen
10	anonymous	Hat spaß gemacht zu Spielen

16. Was könnte verbessert werden, um das Spielerlebnis zu optimieren?

10 Antworten

ID ↑	Name	Antworten
1	anonymous	Schwierigere Level, mehr Optionen von Blöcken
2	anonymous	Mehrere lvl
3	anonymous	Mehrere Varianten von Blöcken
4	anonymous	Mehr Varianten der Blöcke
5	anonymous	unbedingt mehr Level, mehr Gegner und mehr Blöcke zum plazieren
6	anonymous	Mehr Level
7	anonymous	Mehr Level und mehr optionen für Beide Spieler richtig aufeinander einzugehen
8	anonymous	Schwierigere Stellen so wie die eine mit den vielen Gegner an der Wand
9	anonymous	mehr Rätsel
10	anonymous	Mehr coole Gegner

17. Gibt es Funktionen oder Mechaniken, die die Koop-Erfahrung verbessern könnten?

10 Antworten

ID ↑	Name	Antworten
1	anonymous	Nein
2	anonymous	Doppel-Jumps
3	anonymous	Nein
4	anonymous	Zum Beispiel ein Block bei dem man höher Springen kann oder so
5	anonymous	mehr Blöcke
6	anonymous	mehr coole Blöcke wie zum Beispiel eine Kugel die die Gegner zerstören könnte
7	anonymous	Ja, mehr Blöcke, mehr Rätsle, mehr Gegner
8	anonymous	Mehr Funktionen für den ersten Spieler
9	anonymous	nein
10	anonymous	vielleicht ducken?

18. Haben Sie sonstiges Feedback, dass Sie teilen möchten?

10 Antworten

ID ↑	Name	Antworten
1	anonymous	Hat Spaß gemacht :)
2	anonymous	All in all - nice
3	anonymous	Sehr viel Spaß
4	anonymous	Hat viel Spaß gemacht
5	anonymous	Hat richtig viel Spaß gemacht
6	anonymous	War lustig
7	anonymous	war lustig
8	anonymous	war lustig es zu spielen hätte nochmal lust es zu spielen
9	anonymous	cooles Spiel
10	anonymous	Cooler Spiel hat spaß gemacht

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich vorliegende Bachelorarbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Stellen sind als solche kenntlich gemacht.

Die Arbeit wurde bisher weder in gleicher noch in ähnlicher Form einer anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Dornbirn, am Mai 2024

Nina Hartmann