

Dokumentation VR-Projekt

Mystaze

Repository: https://im-prod.hdm-stuttgart.de/repos/Labyrinth_WS2021_DELSS22

Gameplay:

https://www.hdm-stuttgart.de/stage/projekt_detail/projekt_details?projekt_ID=3738



Team:

Mascha Weis (mw213, Matrikelnummer: 37052)

Susanne Weiß (sw191, Matrikelnummer: 37404)

Alexander Kraus (ak271, Matrikelnummer: 37034)

Dennis Heuwieser (dh099, Matrikelnummer: 38127)

Betreuer:

Prof. Dr. Jens-Uwe Hahn

Zeitraum:

Wintersemester 21/22

Inhaltsverzeichnis

Einleitung	3
Idee	3
Technik	4
Steuerung	6
Sicht	6
Schuhe	6
Umsetzung	8
Aufbau	8
Erste Überlegung:	8
Zweite Überlegung:	9
Programmierung	14
Enemy-Charakter (KI)	14
Blackboard	14
Behavior Tree	15
Weitere Funktionen	17
Objekt einsammeln	17
Savepoints	18
Menü	19
Schwierigkeiten	21
Modellierung	23
Licht	25
Design	26
Sound	30
Erweiterungen	31
Abschluss	32

Einleitung

Im Rahmen unseres VR-Praktikums ist das Spiel „Mystaze“ entstanden. Der Name besteht aus einem Wortspiel der Wörter „mystery“ und „maze“, denn genau das wollten wir entwickeln: Ein geheimnisvolles, mysteriöses Labyrinth, das zum Nachdenken anregt und Fragen aufwirft, die nicht direkt beantwortet werden können.

Da einige spannende Features unseres Spiels in dieser Doku verraten werden, sollte die Doku erst nach der ersten Spielrunde gelesen werden, um den Überraschungseffekt der ersten Spielrunde zu gewährleisten.

Ein Labyrinth wird oftmals nur eine Runde gespielt. Durch diverse Extras, wie eine KI als Verfolger und einen Schlüssel, der gefunden werden muss, möchten wir den Anreiz schaffen, das Spiel öfter zu spielen und dennoch den Spannungsfaktor aufrecht zu erhalten.

Um die Steuerung realitätsgetreuer zu gestalten und mehr Spielenden die Möglichkeit zu bieten, dass es ihnen beim Spielen nicht schlecht wird, haben wir uns dazu entschieden, Cybershoes zu verwenden. Es handelt sich hierbei um eine Art Schuhe, die der Spieler anzieht und mit denen er Schrittbewegungen ausführt. Dabei sitzt der Spieler auf einem drehbaren Hocker, kann somit in alle Richtungen laufen. Die Schritte werden im Spiel ausgeführt.

Idee

Die Idee, ein Labyrinth als VR-Spiel zu erstellen, entstand beim Brainstorming und hat uns direkt alle überzeugt. Dabei handelte es sich um ein Spiel, bei dem wir die unterschiedlichen Gestaltungswünsche unserer Gruppe gut kombinieren konnten. Die Schwierigkeit bestand darin, dass ein Teil unserer Gruppe gerne ein Horrorspiel erstellen wollte und der andere Teil ein verspieltes, schön aussehendes Spiel mit rosa Schweinchen.

Unser Labyrinth ist unterirdisch gebaut und besitzt eine KI, die den Spieler verfolgt, um den Gruselfaktor zu erfüllen. Gleichzeitig besteht das Labyrinth aus unterschiedlichen Themengebieten und wurde sehr schön, aber gleichzeitig auch möglichst unheimlich dekoriert.

Dabei war uns wichtig, ein eindrucksvolles Spiel zu erstellen, das nicht nur einmal, sondern öfter einen Anreiz zum Spielen bietet. Zusätzlich sollte sich das Spiel für den Spieler anfühlen, wie ein Ausflug in eine andere Welt. Dieses Ziel haben wir unserer Meinung nach durch die vielen Dekorationen und unterschiedlichen Themengebiete im Labyrinth erreicht.

Technik

Wir hatten mehrere Möglichkeiten das Spiel Mystaze umzusetzen. Von den zwei bekanntesten Game Engines (Unity und Unreal Engine), haben wir uns dazu entschieden, Unreal Engine als Entwicklungsumgebung zu verwenden. Uns war es wichtig, einen sehr realistischen Stil zu kreieren. Hierfür gefiel uns das Erscheinungsbild und die große Auswahl an Assets der UE4 am besten. Weitere Assets haben wir mithilfe von Blender modelliert. Blender ist eine kostenlose und quelloffene 3D-Entwicklungsumgebung. Es unterstützt die gesamte 3D-Pipeline: Modellierung, Rigging, Animation, Simulation, Rendering, Compositing und Motion Tracking.

VR-Brille: Die Brille wurde entwickelt, um den Anforderungen der anspruchsvollsten VR Anwender von heute zu genügen - von weitläufigen Office Umgebungen und stark frequentierten Arcade- und Erlebniswelten bis hin zum Komfort des eigenen Wohnzimmers. Beim Spielen von Mystaze, müssen sich die Spielenden sehr oft und schnell umschauen. Dank der hohen Auflösung von 4896 x 2448 Pixeln, der hohen Frequenz von 90hz und einem weiten FoV (Field of View), kann sie größtenteils der Motion Sickness entgegen wirken.

Um sich innerhalb des Spiels fortzubewegen, gab es folgende Möglichkeiten:

Möglichkeit 1:

Die Bewegung findet automatisch statt. Der Spieler schaut nur in die Richtung, in die er laufen möchte.

Nachteile: Diese Art der Fortbewegung wirkt sehr unrealistisch und unecht. Außerdem sind die meisten Spielenden bei dieser Art der Fortbewegung von Motion Sickness befallen.

Möglichkeit 2:

Der Spieler bewegt sich mit den VIVE-Controllern (ähnlich wie ein Nintendo Switch Controller).

Nachteil: Der Spieler steuert zwar die Bewegungsrichtung selbst, aber der Körper bewegt sich dennoch nicht. Daher sind auch hier die Spielenden von Motion Sickness befallen.

Möglichkeit 3:

Der Spieler bewegt sich durch die "Teleportier-Funktion" des Controllers. Dabei visiert der Spieler einen Punkt an und springt zu diesem Punkt.

Nachteil: Durch diese Art der Fortbewegung sind die Spielenden zwar nicht von Motion Sickness befallen, aber der Anreiz sich selbst durch das Labyrinth zu bewegen geht dabei verloren. Außerdem ergibt die Verfolgung durch eine KI keinen Sinn mehr, wenn der Spieler sich einfach wegteleportieren kann.

Möglichkeit 4:

Der Spieler bewegt sich mit den "Cybershoes". Dabei handelt es sich um Schuhe, die der Spieler überzieht. Diese reagieren auf die Bewegungen des Spielers:

Vorteil: Der Spieler führt eine reale Bewegung aus. Daher sind die Spielenden nicht so schnell von Motion Sickness befallen und es fühlt sich deutlich realer an. Wenn man beispielsweise dem Wolf entkommen muss, muss man seine Füße schneller bewegen.

Nachteil: Durch die Schuhe muss man im Sitzen spielen. Dadurch wird das Gefühl des echten Laufen etwas gemindert.

Möglichkeit 5:

Der Spieler bewegt sich auf einem omnidirektionalen Laufband. Obwohl der Spieler eigentlich nur auf der Stelle läuft, kann er sich in alle Richtungen selbstständig fortbewegen.

Vorteil: Das Laufen auf einem Laufband fühlt sich nahezu real an und bekämpft Motion Sickness.

Möglichkeit 6:

Ein quadratischer Bereich von ein paar Metern wird abgemessen. Der Spieler kann sich darin frei umherbewegen und die Bewegungen finden analog dazu im Spiel statt.

Vorteil: Die Bewegung ist komplett real.

Nachteil: Leider war diese Möglichkeit schwer in unsere Idee integrierbar. Das Labyrinth hätte sich auf einen äußerst kleinen Bereich konzentrieren müssen und lange Verfolgungsjagden wären unmöglich gewesen.

Nach Abwägung der verschiedenen Möglichkeiten haben wir uns für die Wahl der Cybershoes entschieden. Es hat uns gereizt eine relativ neue und, für die Beteiligten, unbekannte Technologie zu testen. Natürlich würde ein omnidirektionales Laufband Bewegungen noch realistischer darstellen, aber dies war im Rahmen dieses Projekts nicht möglich. Auch nach Abschluss des Projekts können wir nach wie vor bestätigen, dass die Cybershoes eine gute Wahl waren. Die Bewegung mittels Kontroller hat bei einigen von uns schon nach ein paar Sekunden Spielzeit zu Motion Sickness geführt, wobei wir mit den Cybershoes mehrere Minuten spielen konnten. Außerdem macht es deutlich mehr Spaß. Dadurch kann der Spieler jederzeit nach eigenem Ermessen stehen bleiben und sich umsehen oder auch bei der Verfolgung durch den Wolf die Geschwindigkeit selbst wählen.

Steuerung

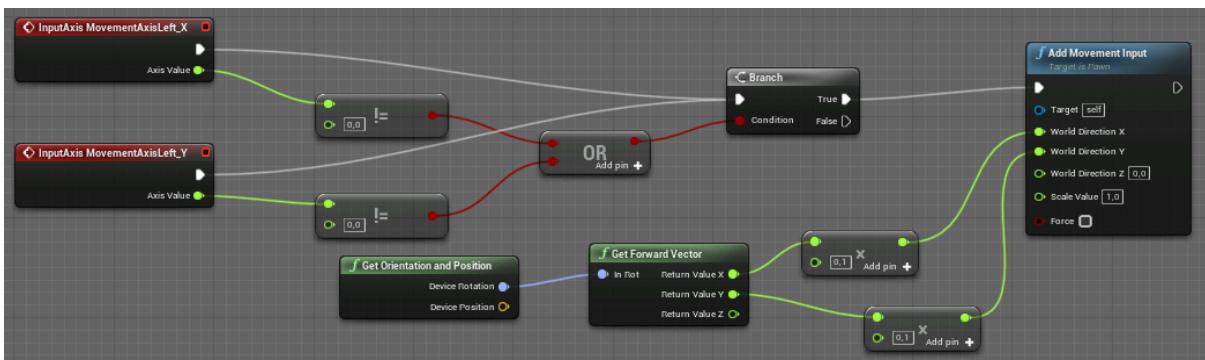
Sicht

Wir nutzen eine HTC Vive Pro in Verbindung mit SteamVR. Damit lässt sich die VR-Brille in Unreal Engine leicht einbinden und benutzen. Weitere Einstellungen waren in Unreal nicht nötig, da die Position des HMD automatisch getrackt wird. Die Position und Rotation ist leicht ab zu fragen, was wir uns oft zunutze gemacht haben (s. Programmierung - Objekt einsammeln). Vorab musste die VR-Brille in SteamVR kalibriert werden, was gut erklärt und einfach auszuführen ist.

Beim ersten Mal spielen ist uns dann aufgefallen, dass man mit der Brille durch Wände schauen kann. Der Player-Charakter wird zwar theoretisch von der Wand geblockt, wenn man sich mit dem Oberkörper aber nach vorne lehnt, kann man den Kopf "durch die Wand stecken". Ein Lösungsansatz war die Verwendung von so genannten "Blocking Volumes", die der Spieler nicht betreten kann. Wenn man diese um die Wand legt, kann der Spieler nicht mehr so nahe an die Wand heran treten und müsste, um durch die Wand blicken zu können, sich extrem nach vorne lehnen, was im Spielgeschehen doch eher unwahrscheinlich ist. Da man diese Situation aber natürlich nicht ausschließen kann, war eine weitere Idee den Bildschirm schwarz zu färben, wenn sich der Kopf des Players in der Wand befindet. Diesen Zeitpunkt, wenn der Kopf in die Wand eintritt, müsste man abfangen. Das ist uns leider nicht gelungen. Die Position des Players ab zu fragen ist kein Problem, allerdings steht dieser ja vor der Wand. Auch der Versuch, es über die im Spieler integrierte Kamera zu lösen, scheiterte. Wie man die Position und das Objekt des Kopfes abgreift, haben wir im Rahmen dieses Projekts leider nicht mehr herausfinden können.

Schuhe

Um das Spielerlebnis zu verbessern nutzen wir, wie vorher beschrieben, die Cybershoes. Deren Einbindung und Nutzung stellten uns vor allem am Anfang vor einige Herausforderungen. Zum einen ist deren Dokumentation nicht sonderlich umfangreich, zum anderen reagieren sie nicht immer gleich. Durch ausprobieren haben wir die Namen der Achsen herausgefunden, die in Unreal aktiviert werden, wenn man die Schuhe bewegt.



Bewegt man die Schuhe vor oder zurück beginnt der Spielende sich in die Richtung zu bewegen in die er schaut. Wir finden, dass ein versehentliche Rückwärtsbewegung im Spiel

eher hinderlich wäre und verwandeln deshalb ein Vor und Zurückbewegen der Schuhe in eine Vorwärtsbewegung.

Leider war es uns nicht möglich herauszufinden, wie man die Position der Schuhe tracken kann. Laut der offiziellen Webseite sollte das und somit auch die Möglichkeit des Hüpfens bzw. Treppenlaufens möglich sein.

Umsetzung

Aufbau

Bei unseren ersten Überlegungen haben wir entschieden, ein relativ großes, rechteckiges Labyrinth zu gestalten, damit eine Verfolgungsjagd des Wolfs möglich wird. Im Labyrinth soll ein sogenannter „Saferoom“ sein, in welchem der Spieler sicher vor dem Wolf ist. Um aus jeder Richtung die Chance zu haben, in den Saferoom zu fliehen, soll dieser ungefähr in der Mitte des Labyrinths liegen.

Erste Überlegung:

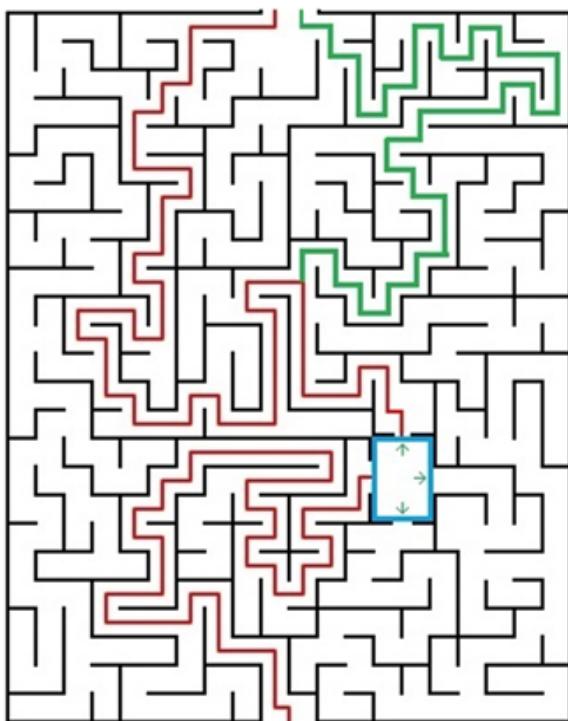


Abb: erste Idee – Aufbau des Labyrinths

Unsere erste Idee enthielt noch sehr viele Sackgassen. Eigentlich sind lange Irrwege, die man wieder zurück laufen muss, gewollt und sorgen in einem Labyrinth für Spannung und einen höheren Schwierigkeitsgrad. Allerdings hat der Spieler in einer Sackgasse keine Chance hat dem Wolf zu entkommen und sitzt somit in der Falle. Aus diesem Grund haben wir uns dazu entschieden, keine langen Sackgassen einzubauen. Kurze, einsehbare Sackgassen sind weiterhin enthalten. Um dennoch die Spannung eines Labyrinths zu erhalten und zu verhindern, dass es nicht nur einen richtigen Weg gibt, haben wir viele Schleifen eingebaut, sodass der Spieler, wenn er nicht aufpasst, im Kreis läuft.

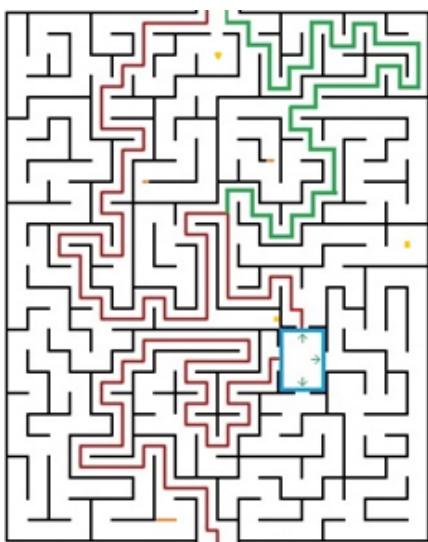
Zweite Überlegung:

Abb: Aufbau des Labyrinths – ohne Sackgassen

Wir haben uns dazu entschieden, dass der Saferoom im ersten Drittel liegt, sodass der hintere Teil schwerer wird. Aus dem Saferoom heraus führen zwei Türen in die Irre und nur eine Tür führt zum Ziel. Nach dem Saferoom gibt es zwei richtige Wege.

Nachdem wir angefangen hatten, das Labyrinth zu erstellen, ist uns aufgefallen, dass wir deutlich mehr Zeit zum Bauen des Labyrinths benötigen als gedacht. Deshalb haben wir uns vorerst auf einen Hauptweg zum Ende konzentrieren und seitliche Nebenbereiche erst einmal vernachlässigen. Um zu verhindern, dass das Labyrinth zu eintönig oder langweilig wird, wurde jeder Bereich im Labyrinth in einem anderen Stil gebaut und mit Besonderheiten versehen.

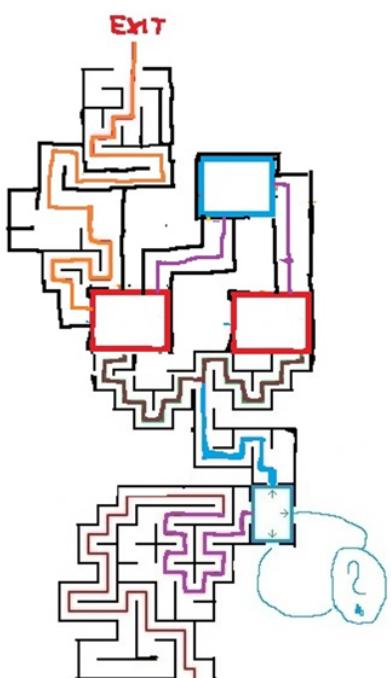


Abb: Aufbau des Labyrinths - Sonderbereiche

Finales Ergebnis:

Letztendlich ist unser Labyrinth groß und sehr abwechslungsreich geworden

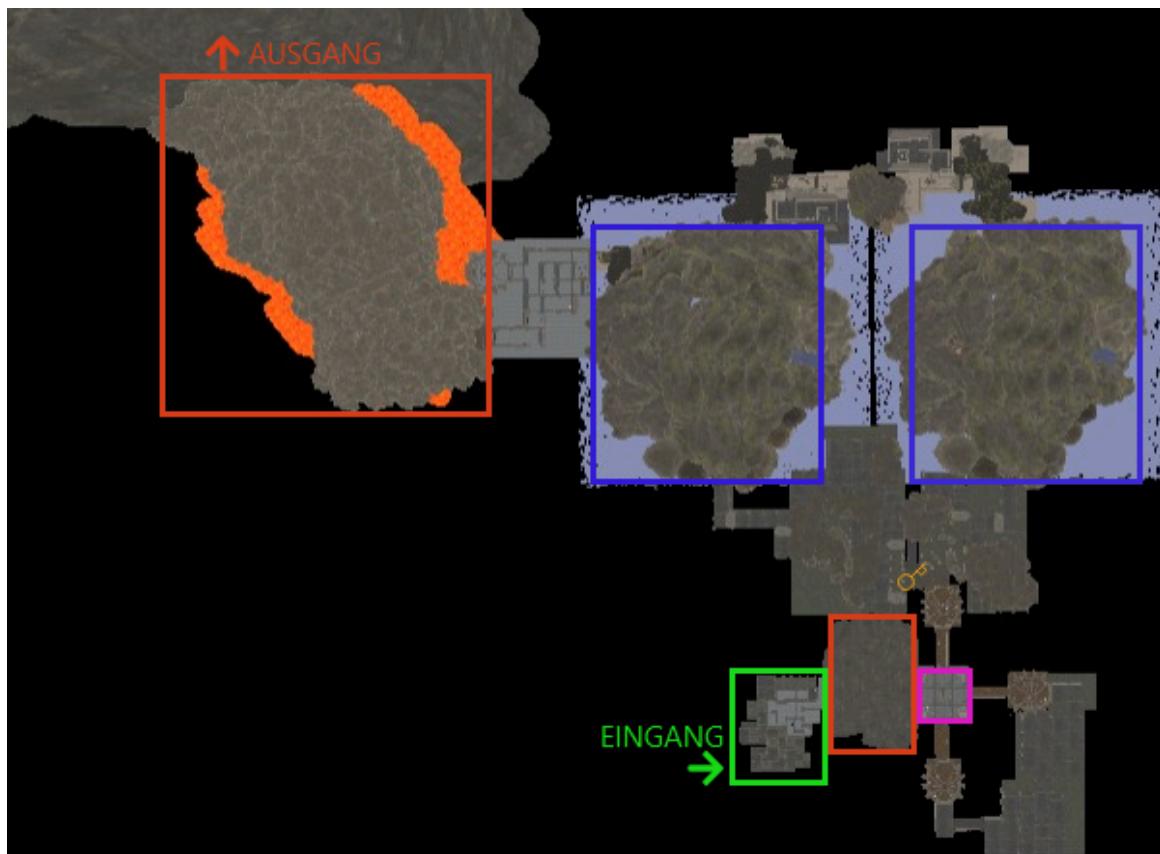


Abb: Ansicht des Labyrinths aus der Vogelperspektive

- **Grün - Einführungsbereich:** Hier beginnt das Labyrinth. Es gibt nicht viele Möglichkeiten sich zu verirren. In diesem Teil haben wir nicht darauf geachtet Sackgassen zu vermeiden, da hier der Wolf nicht unterwegs ist. Dieser Bereich dient dazu, dem Spieler die Möglichkeit zu geben, in aller Ruhe sich an die Steuerung (besonders an die Schuhe) zu gewöhnen.
- **Orange – Feuerwelt:**
 - Es gibt einen Lava-Bereich am Anfang, dessen Aussehen den Spielenden beeindrucken soll. Die Gefahr des Wolfs lauert auch hier noch nicht. Der Spieler muss beim Überqueren der Brücke nur aufpassen nicht abzustürzen. Anschließend an die Brücke folgt der Saferoom.
 - Am Ende folgt nochmals ein Lava-Bereich. Hier lauern schon mehr gefahren. Der Spieler muss aufpassen nicht von den verzwickten Wegen abzukommen und nicht vom Wolf erwischt zu werden. Der Spieler ist kurz davor das Ziel erreicht zu haben.

- **Pink – Saferoom:** Hier war der Gedanke, wie bereits erwähnt, eine sichere Zone vor dem Wolf zu schaffen. Der Spielende kann aus allen Richtungen zurück zum Saferoom fliehen, um dem Wolf zu entkommen. Der Saferoom ist zur Verwirrung symmetrisch aufgebaut, sodass der Spielende genau aufpassen muss, welche der vier Türen die Richtige ist. Eine der Türen, durch die der Spielende hereinkommt, führt zurück zum Eingang. Zwei Türen führen in die Irre. Der Spielende läuft im Kreis und kommt wieder zum Saferoom zurück. Die vierte Tür führt in die richtige Richtung.
- **Blau - Wasserwelt:** Die Wasserbereiche sind ein eigenes Labyrinth für sich. Innerhalb einer riesigen, unterirdischen Wasserhöhle ist ein Labyrinth aus vielen Brücken gebaut. Auch hier ist eine weitere Gefahr, dass nicht alle Brücken Geländer haben und der Spielende abstürzen kann. Beide Wasserwelten sind fast identisch aufgebaut, um den Spielenden zu verwirren. Auf den ersten Blick wirkt die zweite Wasserwelt auf den Spielenden wie die Erste, sodass dieser denkt, er wäre im Kreis gelaufen. Erst bei genauerer Betrachtung der zweiten Wasserwelt fällt der richtige Ausgang auf, der in der ersten Wasserwelt nicht vorhanden war.
- **Schlüssel:** Der Schlüssel ist ein besonderes Extra. Er liegt im Saferoom auf dem Tisch und muss vom Spielende einsammeln werden. So hat er die Möglichkeit den kurzen Weg (rechter Wasserbereich wird übersprungen) zum Ziel zu nehmen. Nachdem sich der Spielende für den linken Weg, heraus aus dem Saferoom, entschieden hat, gibt es noch einmal eine entscheidende Gabelung. Die Tür, in der Übersicht markiert durch den Schlüssel, ist nur geöffnet, wenn der Schlüssel zuvor im Saferoom eingesammelt wurde. Wenn man durch diese Tür geht, kommt man direkt durch den linken Wasserbereich zur letzten Feuerwelt und letztendlich zum Ausgang. Falls man den Schlüssel nicht eingesammelt oder die Abzweigung verpasst hat, folgt ein deutlicher Umweg. Der linke Weg geht durch beide Wasserwelten und eine Verbindung dazwischen.
- **Savepoints:** Da es fast unmöglich ist, aus dem großen Labyrinth in einem Durchlauf ohne zu sterben zu entkommen, haben wir ein paar Savepoints eingefügt. Falls der Spieler durch einen falschen Schritt in die Lava oder das Wasser stirbt oder vom Wolf erwischt wurde, muss er nicht von vorne anfangen, sondern kann beim zuletzt durchschrittenen Savepoint starten. (Die genaue Funktion wird im Abschnitt Programmierung erläutert.)

Abbildungen der beschriebenen Bereiche:

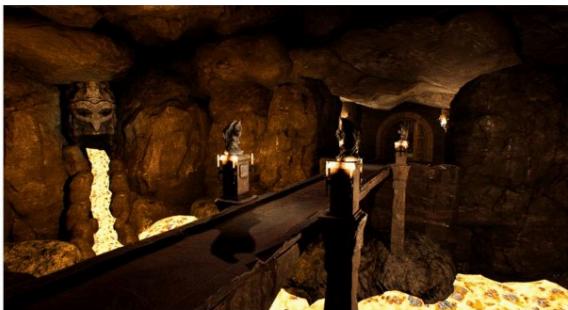


Abb: erster Feuerbereich



Abb: Saferoom



Abb: Wasserbereich



Abb: zweiter Feuerbereich

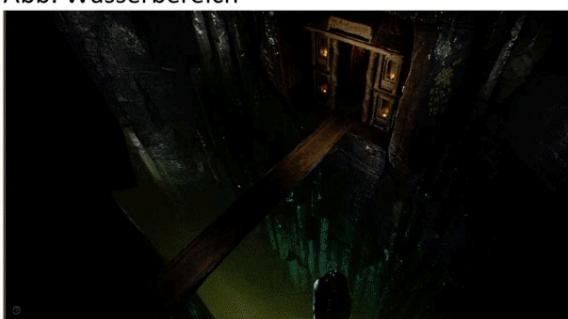


Abb: Ausgang



Abb: Mausoleum

Nicht nur die besonderen Bereiche sollen den Spielenden beeindrucken, sondern auch die Gänge zwischen den einzelnen Bereichen. Um dennoch eine Einheit zu schaffen und einen Bereich nicht stillos mit verschiedenen Materialien zu überhäufen, sind die Gänge auch in Bereiche eingeteilt und bleiben einem Stil treu. Anschließend folgen ein paar Bilder der Gänge, um einen ersten Eindruck zu vermitteln:

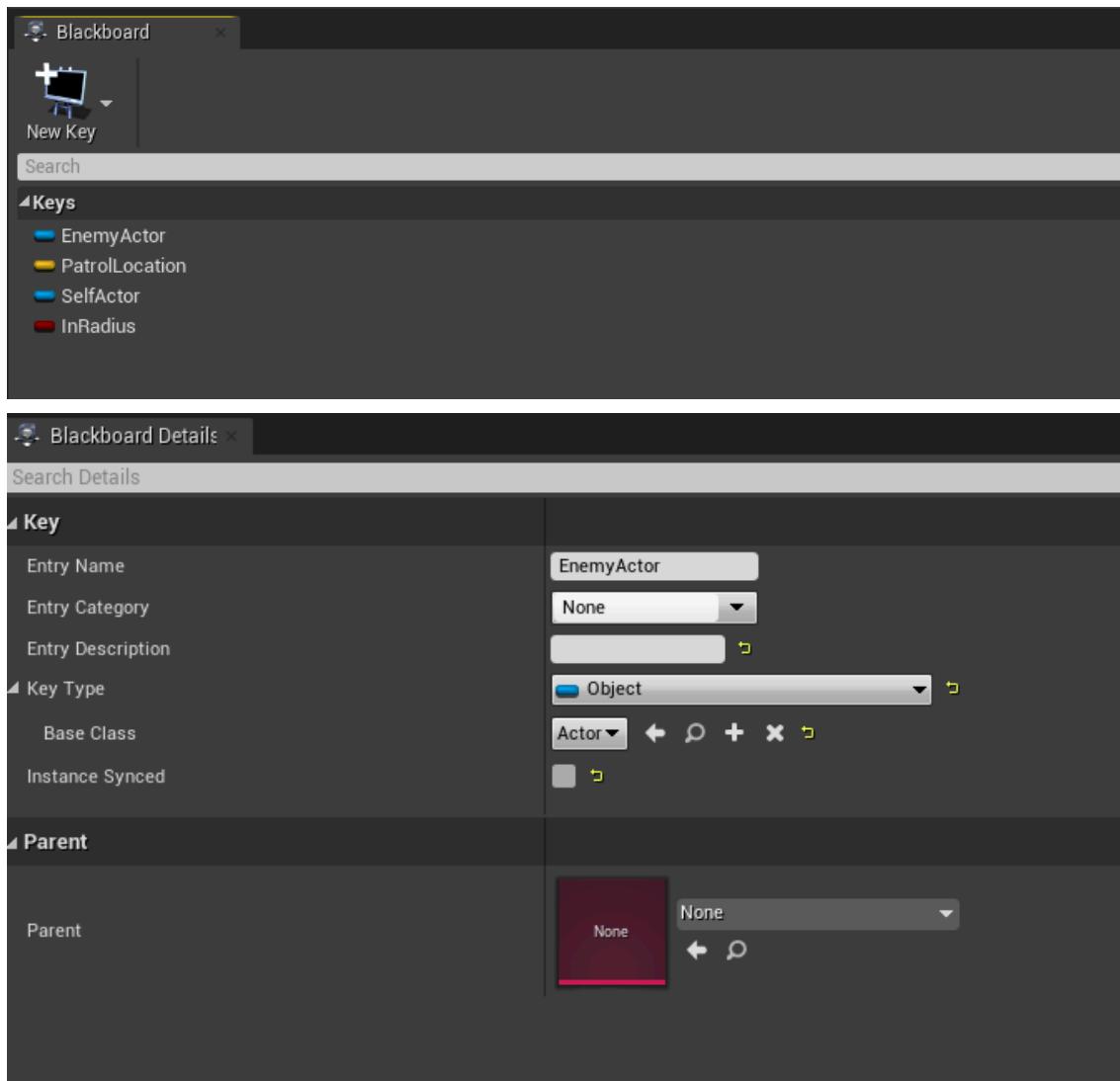


Programmierung

Enemy-Charakter (KI)

Um eine künstliche Intelligenz zu erstellen gibt es in Unreal Assets, die die Programmierung deutlich erleichtern. Zum einen Nutzen wir den "Behavior Tree" in Verbindung mit einem "Blackboard". Während der "Behavior Tree" genau das ist, was er vorgibt zu sein, nämlich ein Baum mit einzelnen Logik-Zweigen, die später das Verhalten widerspiegeln werden, ist das "Blackboard" das "Gehirn", das hinter dem Tree steht.

Blackboard



Im Blackboard werden sogenannte "Keys" hinterlegt, die in Blueprint-Skripten gesetzt und im Behavior Tree abgefragt werden können. Jeder Key hat einen Namen und einen Typ. Als Typ können dabei alle gängigen Typen von Unreal verwendet werden. Im Folgenden werden die einzelnen Keys in Bezug auf ihren Nutzen erklärt:

EnemyActor

Typ: Object der Klasse Actor

Repräsentiert den Spieler, da der Spieler den Gegner der KI darstellt

PatrolLocation

Typ: Vector

Repräsentiert den Punkt, zu dem der Enemy-Charakter als nächsten patrouillieren soll

SelfActor

Typ: Object der Klasse Actor

Repräsentiert den Enemy-Charakter. Da er der "Besitzer" des Trees ist

InRadius

Typ: Boolean

Gibt an, ob sich der Spieler innerhalb des Radius befindet, in dem der Enemy anfängt den Spieler zu jagen

Behavior Tree

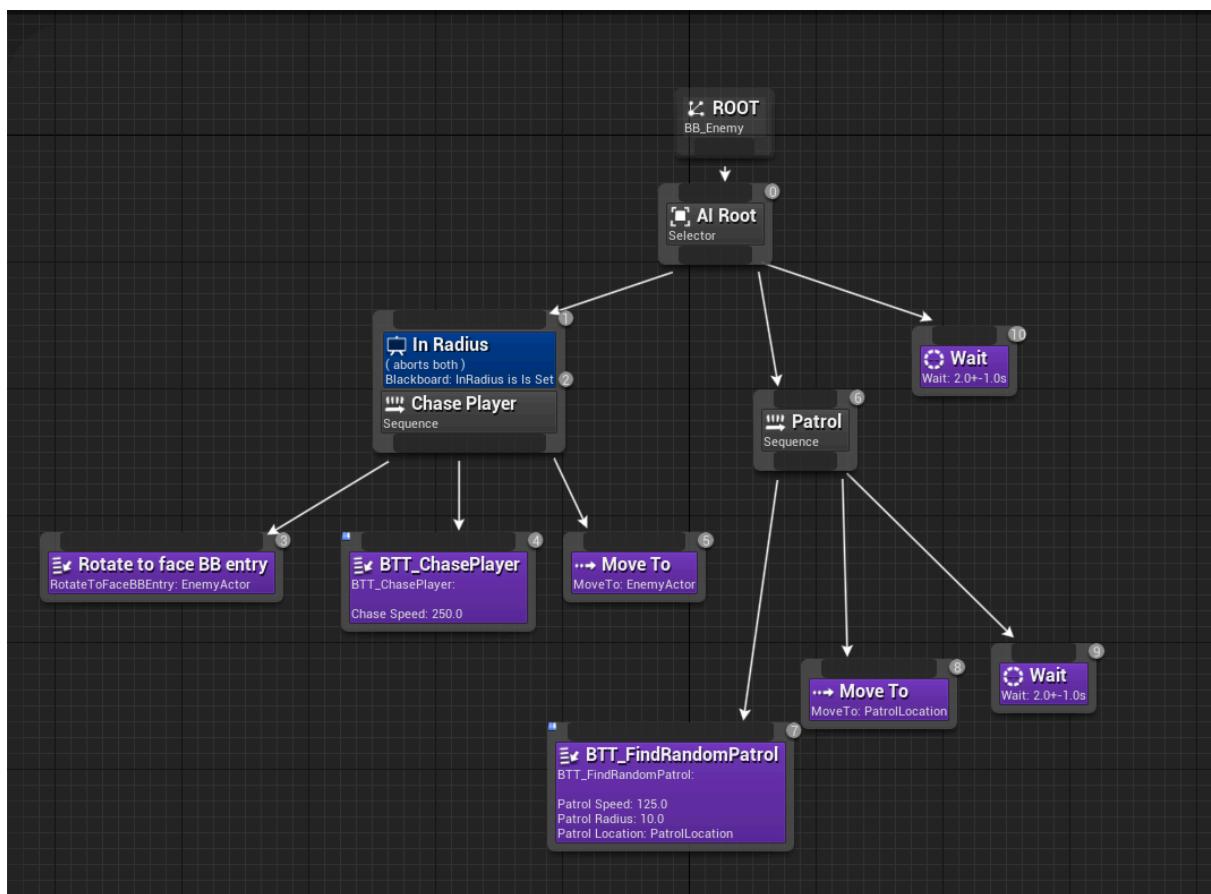


Abb Behavior Tree

Ein "Behavior Tree" beginnt immer mit einer Root, die auf das dazugehörige "Blackboard" verweist. In unserem Fall "BB_Enemy". Unserer Root-Node folgt eine Selector-Node. Diese führt die direkten Verzweigungen unter ihr von links nach rechts aus. Sollte sie auf einen

Zweig stoßen, der im Moment nicht ausgeführt werden kann, springt sie automatisch zum nächsten.

Der erste Zweig von links, beschreibt das Verhalten, wenn der Enemy-Charakter den Spieler jagd. Damit das passiert, muss sich der Spieler in der Nähe des Enemy-Charakters aufhalten. Realisiert wird diese Bedingung im Behavior Tree mit der Sequenz "Chase Player" kombiniert mit dem Decorator "In Radius".

Eine Sequenz-Node funktioniert im Prinzip wie die vorher erwähnte Selector-Node, mit dem Unterschied, dass sie ihre Kind-Knoten nur so lange ausführt bis einer fehlschlägt. In unserem Fall wird keiner der Kind-Knoten fehlschlagen. Die Sequenz wird also so lange ausgeführt, bis der Selector einen neuen Zweig auswählt, was natürlich passieren soll, wenn der Spieler sich nicht mehr im gewünschten Bereich befindet. Um das zu gewährleisten nutzen wir hier einen Decorator, den man wahrscheinlich besser als Bedingung beschreiben sollte. Er gibt an, abhängig davon, ob eine bestimmte Bedingung erfüllt wird oder nicht, ob der darunter liegende Baum ausgeführt werden kann oder nicht.

Sollte sich der Spieler im gewünschten Bereich befinden, dreht sich der Enemy in Richtung des Spielers, passt seine Geschwindigkeit an und bewegt sich auf ihn zu. Verlässt der Spieler den Bereich, in dem er verfolgt werden soll, wird "In Radius" false, der Zweig gilt als nicht ausführbar und der Selector springt einen Zweig weiter. Nun beginnt der Enemy-Charakter zu patrouillieren. Das bedeutet er sucht sich einen zufälligen Punkt innerhalb der Area, in der er sich bewegen darf, läuft dorthin und wartet. Das wiederholt er so lange, bis "In Radius" wieder true ist und er beginnt den Spieler zu jagen.

Im folgenden werden die verwendeten Blatt-Nodes erklärt:

Rotate to face BB entry

Der Enemy-Charakter dreht sich in Richtung des Spielers, in dem er den Blackboard-Key "EnemyActor" abfrägt.

BTT_ChasePlayer

Dient der Berechnung der Geschwindigkeit des Enemy, in Abhängigkeit der Entfernung des Spielers zum Enemy. Je näher der Spieler, je größer die Geschwindigkeit. Dabei muss beachtet werden, dass der Spieler immer eine Chance behält, dem Enemy zu entkommen.

MoveTo

Der Enemy-Charakter bewegt sich mit der vorher bestimmten Geschwindigkeit zum Spieler, in dem er den Blackboard-Key "EnemyActor" abfrägt.

BTT_FindRandomPatrol

Der Enemy-Charakter findet durch die Funktion "GetRandomReachablePointInRadius" einen Punkt, zu dem er als nächstes patrouillieren muss.

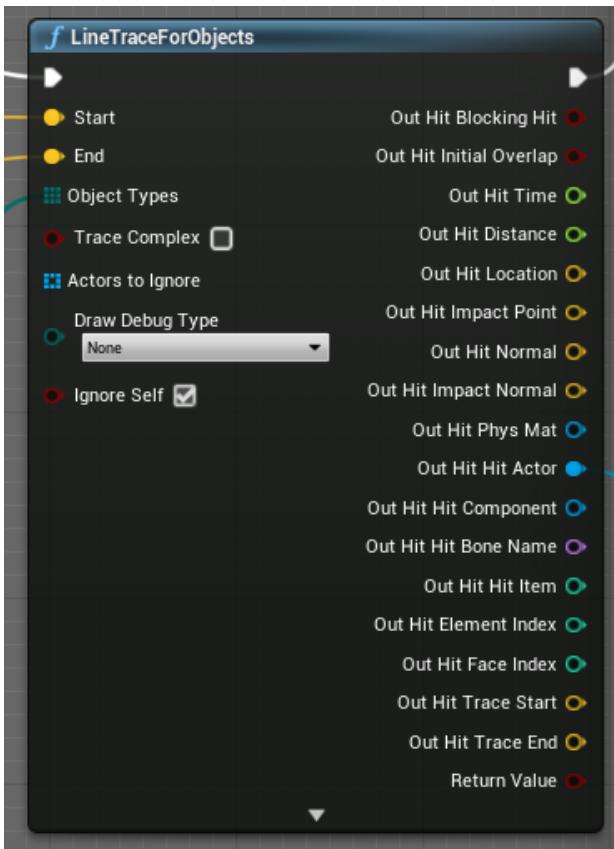
Weitere Funktionen

Objekt einsammeln

Um den "Saferoom" zu verlassen und in die Wasserwelten zu gelangen, muss man einen Schlüssel finden und ihn aufheben, damit sich eine bis dahin verschlossene Tür öffnet. Um den Schlüssel aufzuheben, muss der Spieler 3s auf den Schlüssel schauen. Als Feedback beginnt der Schlüssel zu leuchten.

Auf was schaut der Spieler?

Um zu ermitteln, was der Spieler gerade anschaut benutzen wir die Funktion "LineTraceForObjects".



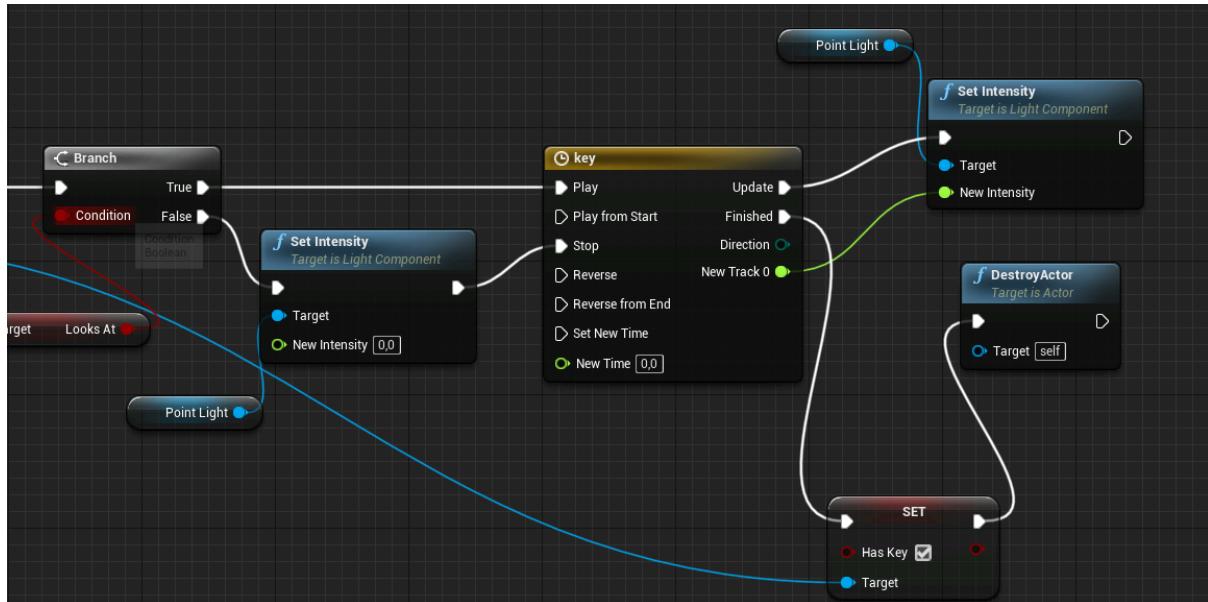
Als Eingabe wird eine Start- und Endposition der "Trace" erwartet. Dies ist bei uns die aktuelle Position des Players und der Endpunkt eines Vektors, der in dieselbe Richtung schaut wie der Spieler.

Wenn die Ausgabe "Out Hit Hit Actor" den Schlüssel ausgibt, wird die Global Variable "Looks At" auf true gesetzt.

Wann beginnt der Schlüssel zu leuchten?

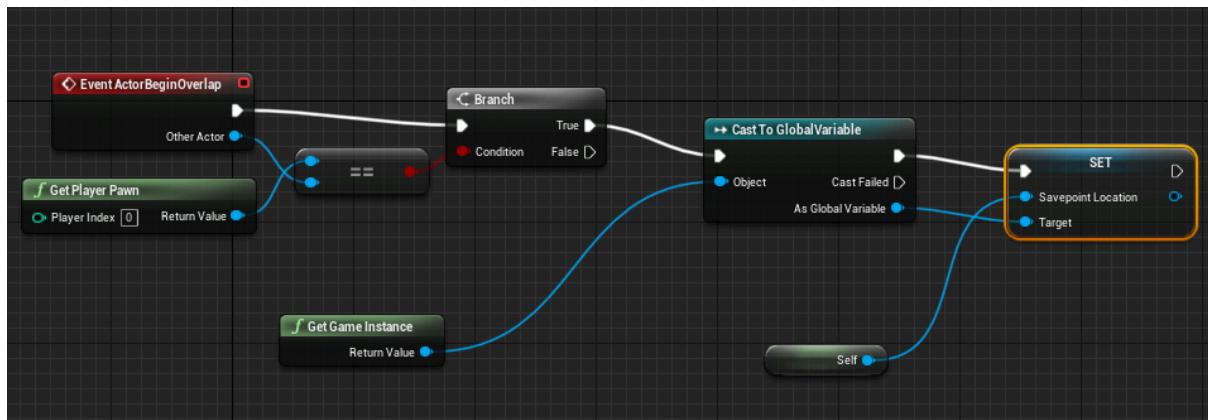
Der Schlüssel prüft jeden Tick, ob die "Looks At"-Variable true ist. Sollte dies der Fall sein, wird die Helligkeit eines integrierten "Point Light" innerhalb von 3s erhöht. Danach verschwindet der Schlüssel.

Um die Helligkeit zu verändern wird hier eine Timeline verwendet:



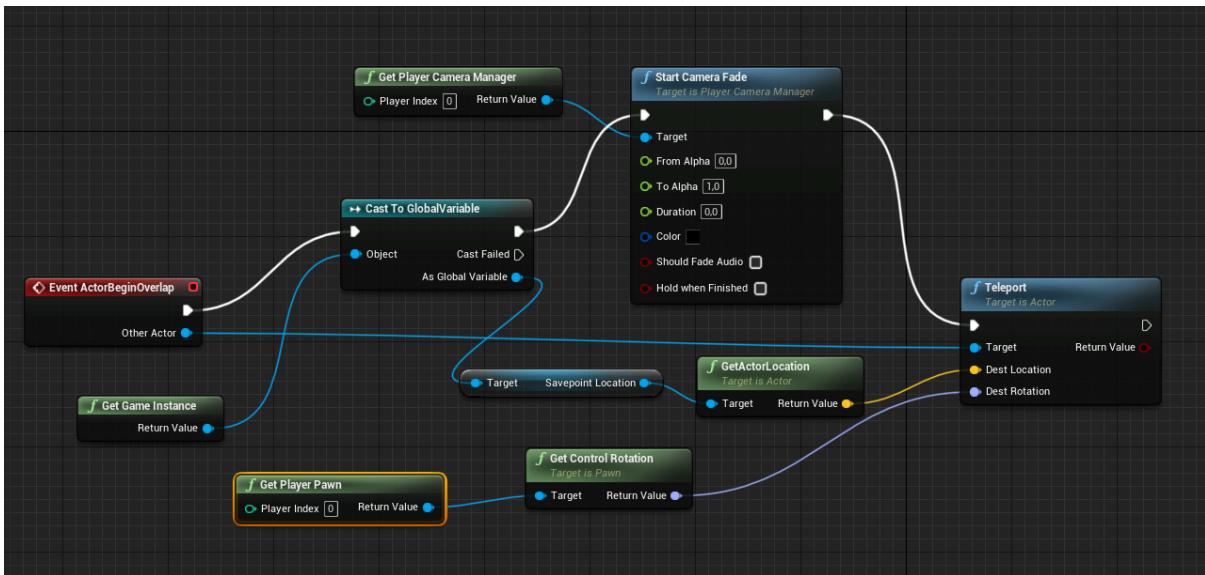
Savepoints

Um einen Savepoint zu aktivieren, muss der Spieler durch ihn hindurch laufen.



Wir das Event "ActorBeginOverlap" ausgelöst, und es handelt sich um den Player und nicht um den Enemy-Charakter, wird der betreffende Savepoint als Objekt Global gespeichert. Läuft der Spieler zu einem späteren Zeitpunkt durch einen weiteren Savepoint, so ersetzt dieser den vorherigen. Damit wird der Spieler immer an dem zuletzt durchschrittenen und somit hoffentlich auch am nächstgelegenen Savepoint weiterspielen können, sollte er sterben.

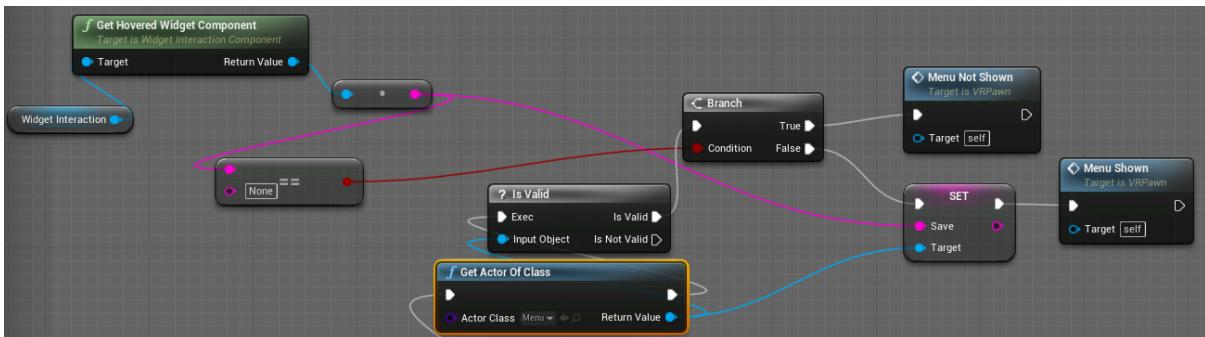
Sterben tut der Spieler zum Beispiel dann, wenn er von einer Brücke ins Wasser bzw. in die Lava läuft.



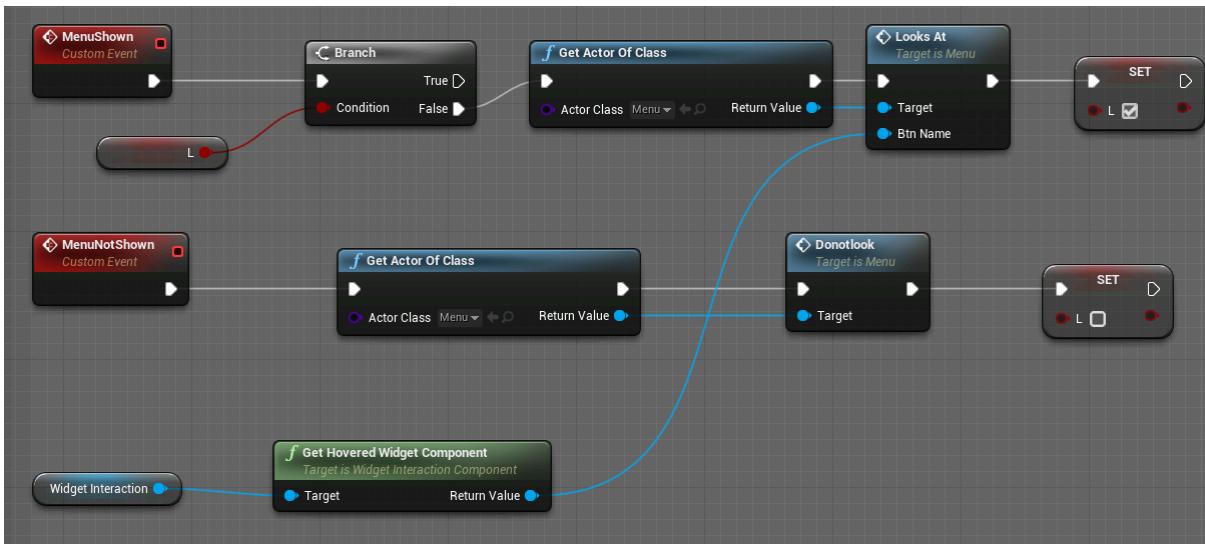
Betrifft der Spieler die kritischen Stellen, wird er an die Position des letzten durchlaufenen Savepoints teleportiert.

Menü

Beginnt das Spiel, erscheint ein Menü mit 2 Buttons. "Play" und "Exit". Einen Button auswählen kann der Spieler in dem er 3s lang auf einen Button schaut. Als Feedback wird der angeschauten Button in dieser Zeit dunkler. Bei "Play" verschwindet das Menü, und das Spiel beginnt. Bei "Exit" wird das Spiel wieder beendet.



Um herauszufinden, was der Spieler gerade anschaut, gibt es einen "Widget Interaction Component". Wenn dort ein Widget und nicht "none" zurückgegeben wird, wird der Name des Widgets in einer Globalen Variable "Save" gespeichert (diese wird später benötigt) und das Event "Menu Shown" ausgelöst. Sollte der "Widget Interaction Component" auf nichts zeigen, startet das Event "Menu Not Shown". Das alles kann nur ausgelöst werden wenn es ein valides Objekt "Menu" gibt.

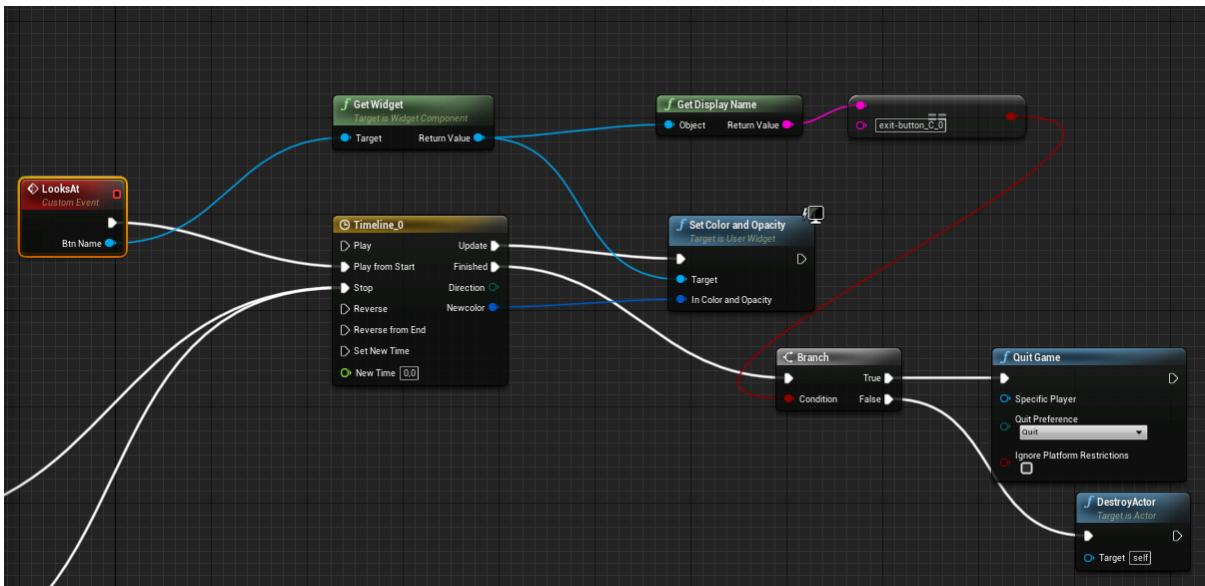


Wird das Event “MenuShown” ausgelöst, wird zunächst überprüft, ob der Spieler schon auf das Menü schaut (boolean L). Ist dies nicht der Fall, wird das Event “Looks At” ausgelöst und L auf true gesetzt.

Ist das Event “MenuNotShown” getriggert, wird das Event “donotlook” ausgelöst und L auf false gesetzt.

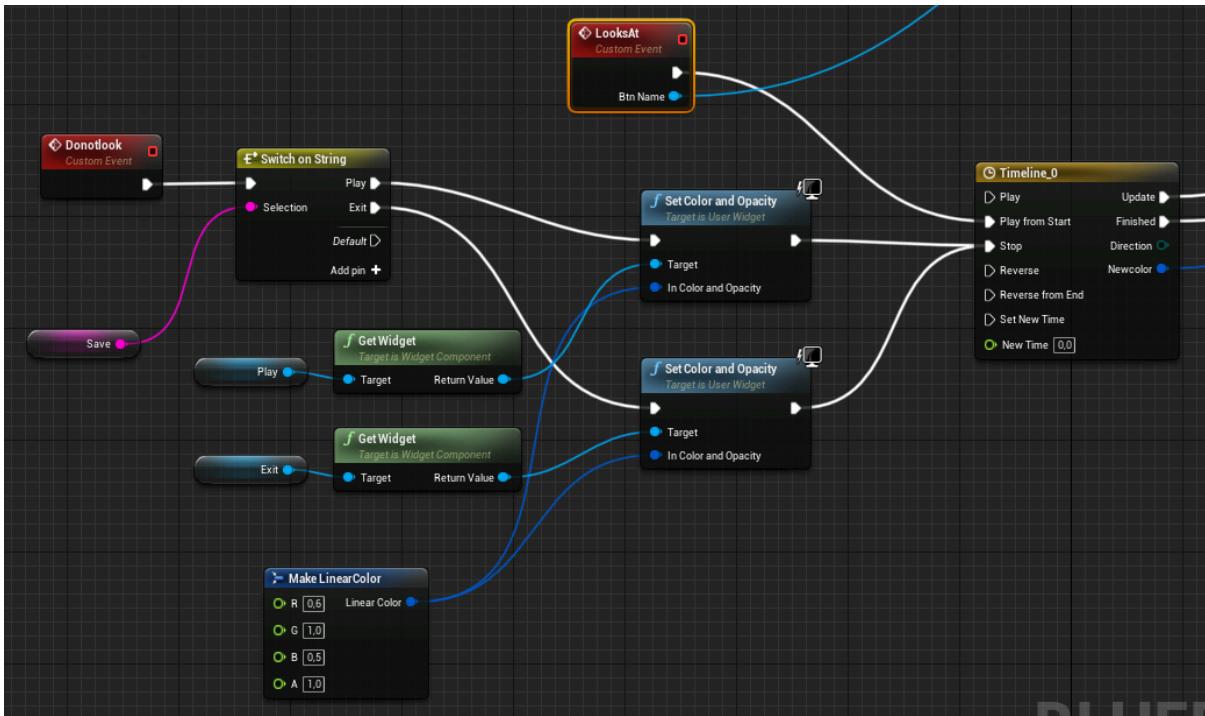
Im Menü-Blueprint wird auf die jeweiligen Events gewartet.

Looksat



Die Timeline wird gestartet. Sie verändert über 3s die Farbe des ausgewählten Buttons. Sind die 3s vorbei, wird überprüft, ob es sich um den Exit-Button handelt. Sollte das der Fall sein, wird das Spiel beendet, ansonsten wird das Menü-Objekt entfernt

Donnotlook



Hier wird zunächst überprüft um welchen Button es sich handelt. Danach wird die Farbe des jeweiligen zurück auf die ursprüngliche Farbe gesetzt und die Timeline gestoppt. Dies hat den Nutzen, dass, wenn der Spieler vom Button wegschaut, er nicht aktiviert wird und in seinen Ursprungszustand zurückgesetzt wird.

Schwierigkeiten

Die wahrscheinlich größte Schwierigkeit war, dass keine von uns vorher mit der Blueprint-Programmierung Kontakt hatte. Somit stießen wir oft auf das bekannte Problem, wenn man eine neue Programmiersprache anwendet. "Ich weiß eigentlich was ich machen will, nur wie?" Da half nur viel ausprobieren und im Internet nach Lösungsansätzen zu suchen.

Das Resultat ist eine Programmierung, die im Großen und Ganzen das macht, was sie soll, aber stellenweise unsauber programmiert wurde. So sind die Variablen und Eventnamen oft nicht mehr passend, da das Programm oft geändert, die Namen aber nicht angepasst wurden. Außerdem fanden wir recht spät heraus, wie man Events einsetzen und Variablen direkt in einem anderen Programm setzen kann, ohne sie global zwischen zu speichern. Um das in allen Programmteilen am Schluss einheitlich zu gestalten, fehlte leider die Zeit.

Eine recht große Herausforderung waren die Savepoints. Getestet haben wir die Funktion in einer anderen Umgebung, damit wir den Player problemlos mit der Tastatur bedienen und so auch von daheim aus daran arbeiten konnten. In diesem Szenario war die Funktion schnell implementiert und funktionsfähig. Beim Übertragen in das Labyrinth wurde der Spieler plötzlich immer irgendwo hin teleportiert, aber nicht an den angegebenen Punkt. Gelöst wurde das Problem einigermaßen, in dem anstelle von nur den Koordinaten zu speichern (wie wir das vorher gemacht haben) das komplette Objekt übergeben wird und davon dann die Koordinaten abzufragen. Leider klappt auch das nur bedingt. Man springt grundsätzlich

einige Einheiten zu weit nach hinten oder zur Seite. Woran das liegt konnten wir uns bis jetzt noch nicht erklären, es könnte aber sein, dass es mit den vielen verschiedenen Bereichen (Volumes genannt) zusammenhängt und die Teleport-Funktion so mit den Koordinaten durcheinander kommt. Das ist aber nur eine ungeprüfte Theorie.

Was dieser Theorie im Wege steht, ist, dass die gleiche Programmierung scheinbar völlig problemlos funktioniert, wenn der Wolf einen eingefangen hat. Dann springt man direkt zur Savepoint-Stelle.

Modellierung

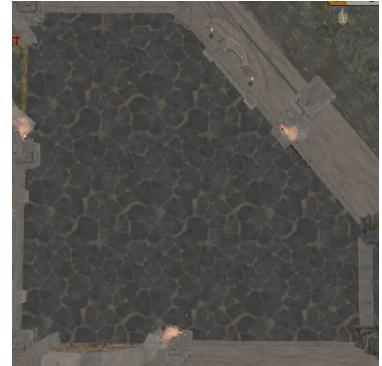
Die meisten unserer Modelle/ Assets stammen aus dem Unreal Engine Marketplace. Da wir nicht jede Idee für das Labyrinth mit vorgefertigten Assets umsetzen konnten, haben wir uns mit Blender vertraut gemacht. Dadurch konnten wir verschiedene Assets wie Tunnel, Böden und Decken modellieren. Viel Zeit ging beim Import und Export der Modelle verloren. Deshalb sind wir auf das Plugin "Mesh Editing" gestoßen, wodurch wir in der Unreal Engine Objekte bearbeiten konnten. Leider ermöglichte uns das Plugin nicht, Modelle in der Entwicklungsumgebung zu erstellen. Bei komplexeren Gängen konnten wir unsere bestehenden Modelle nicht verwenden.



Hier benötigen wir einen Boden, welcher einem Trapez ähnelt. Um effizienter zu arbeiten benutzen wir "Procedural Meshes".

Bei der prozeduralen Geometrie wird die Geometrie in einem Code modelliert. Anstatt 3D-Meshes von Hand mit Software wie Blender zu erstellen, wird das Mesh mit Vertices, Triangles und UV0 erstellt. Diese bestehen aus einem Array von Vektoren, Integers und Vektor 2D Strukturen.

Zu den Vorteilen der prozeduralen Erzeugung von Meshes gehören:



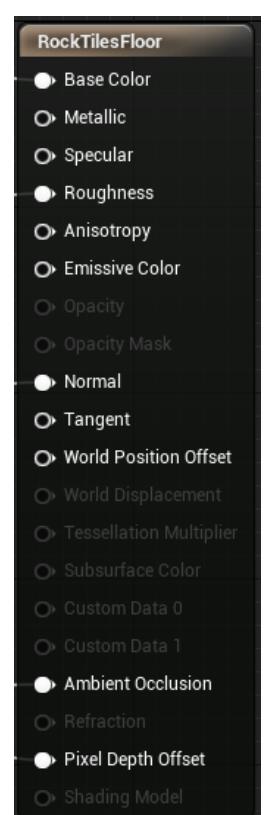
- Variation: Meshes können mit zufälligen Variationen erstellt werden, was bedeutet, dass die Wiederholung von Geometrien vermieden werden kann.
- Skalierbarkeit: Netze können mit mehr oder weniger Details generiert werden.
- Geschwindigkeit: Viele Varianten eines Objekts können einfach und schnell erstellt werden.

Texturen:

Die meisten Texturen haben eine Auflösung von 1-4K. Da uns die Performance des Spiels wichtig ist, haben wir für weit entfernte Objekte niedrig Auflösende (1K) Texturen und für nähere Objekte bis zu 4K Texturen verwendet. Um die Texturen besser der Umgebung anzupassen, haben wir die Helligkeit und Sättigungswerte geändert.

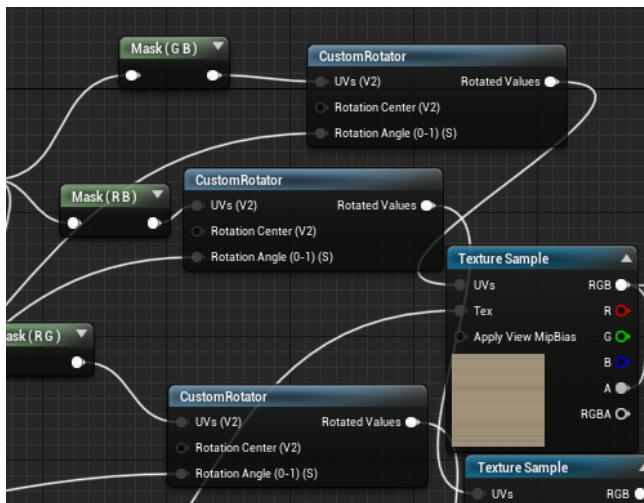
Die meisten unserer Materials bestehen aus den Texturen:

- Base Color (Bestimmt die Farbe des Materials)
- Roughness (steuert, wie rau oder glatt die Oberfläche eines Materials ist. Raue Materialien streuen das reflektierte Licht in mehr Richtungen als glatte Materialien)
- Normal (Die Normalen-Eingabe nimmt eine Normalen-Karte auf, die verwendet wird, um der Oberfläche signifikante physikalische Details zu verleihen, indem die "Normale" oder die Richtung jedes einzelnen Pixels gestört wird)



- Ambient Occlusion (Ecken, Ritze oder andere Merkmale werden abgedunkelt, um ein natürlicheres, realistischeres Aussehen zu erzielen)

Wie oben erwähnt benutzen wir auch Procedural Meshes. Dadurch werden die Objekte beliebig groß skaliert. Da Texturen ab einer gewissen Größe sich verzerrten oder unscharf werden, benutzen wir die Funktion "World Align Texture". Sie dient dazu, eine Textur über die Oberfläche eines Objekts in der Umgebung zu kacheln, unabhängig von der Größe oder Drehung des Objekts. Die Funktion bietet keine Möglichkeit die Materialien zu rotieren. Deshalb haben wir das Blueprint der "WorldAlignTexture" verändert und eine "CustomRotator" Funktion hinzugefügt. Dadurch können wir die Textur beliebig rotieren.



Das war vor allem für die Holzbretter Textur notwendig um diese dem Raum anzupassen.



Licht

Das Spiel befindet sich unterirdisch und hat kein Globales Licht (Directional Light), welches das Level natürlicher erscheinen lässt. Dies war eine sehr schwierige Herausforderung. Wir benötigten für jeden Raum und Gang genügend Licht. Wir haben verschiedene Fackel-Variationen eingesetzt, die sich hauptsächlich von der Lichtfarbe, Lichtstärke und dem Radius unterschieden haben. Anfangs hatten wir beliebig viele Lichter platziert. Dadurch sind wir auf folgende Probleme gestoßen:

- Welche Mobility (Static, Stationary oder Movable)
- Lichtquellen überschneiden sich = keine Schatten mehr (wirkt nicht realistisch)
- Performance (zu viele Lichtquellen)

Statische Lichter sind Lichter, die zur Laufzeit in keiner Weise verändert oder verschoben werden können. Stationäre Lichter sind Lichter, die in einer Position verbleiben sollen, sich aber auf andere Weise verändern können, z. B. in ihrer Helligkeit und Farbe. Dies ist der Hauptunterschied zu statischen Laternen. Es können sich maximal vier Stationäre Lichter überschneiden. Bewegliche Lichter werfen völlig dynamisches Licht und Schatten. Sie können Position, Drehung, Farbe, Helligkeit, Falloff, Radius und so ziemlich jede andere Eigenschaft, die sie haben, ändern. Keines der Lichter, die sie werfen, wird in die Lightmaps gebacken, und sie unterstützen keine indirekte Beleuchtung ohne eine dynamische globale Beleuchtungsmethode.

Da wir sehr viele Fackeln in unserem Spiel benutzen, haben wir uns hauptsächlich auf die Nutzung von statischen Laternen beschränkt. Diese haben die geringste Auswirkung auf die Performance. Um Schatten zu erzeugen benutzen wir stationäre Lichter statt bewegliche. Diese haben den Nachteil, dass kein Schatten auf bewegliche Objekte (Wolf) geworfen wird. Dafür haben sie eine geringere Auswirkung auf die Performance als bewegliche Lichter.

Design

Beim Design unseres Labyrinths haben wir uns für ein unterirdisches Labyrinth in einem zum Teil natürlichen und zum Teil gebauten Stil entschieden. Dabei war uns wichtig, nur realistisch wirkende Materialien zu verwenden und nicht in einen Cartoon oder Stylized-Stil zu verfallen. Hierbei haben wir uns auf Texturen konzentriert, die natürlichen Stein in Grau- und Brauntönen repräsentieren. Teilweise sehen die Gänge aus, als ob sie aus einem großen Steinblock herausgeschlagen wurden. Des Weiteren haben wir verschiedene hohe und breite Gänge gebaut, sowie einige Höhlen und Räume, um eine gewisse Abwechslung, nicht nur durch verschiedene Texturen, sondern auch durch den Baustil zu kreieren.

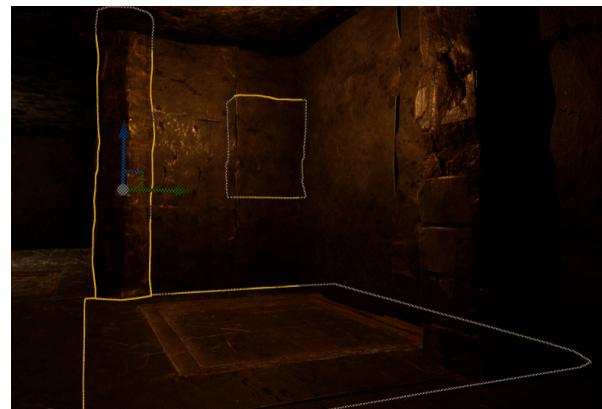
Im Labyrinth sorgen ausschließlich natürliche Lichtquellen für die Beleuchtung der Gänge. Darunter befinden sich verschiedene Fackeln, die ein warmes Licht ausstrahlen. Durch natürlich vorkommende Felsspalten, tritt außerdem Licht aus der Umgebung ein und sorgt dadurch für einen weiteren Lichtton.



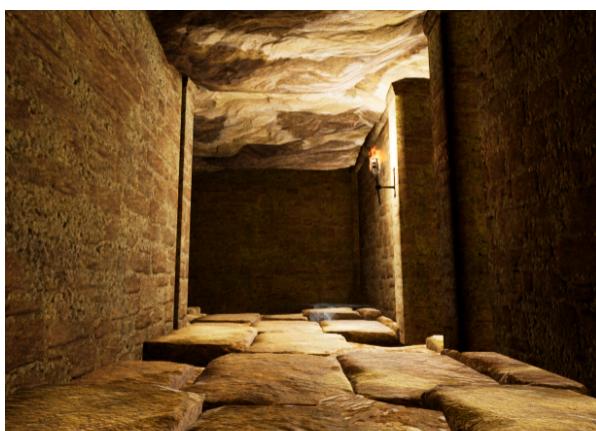
Wie bereits erwähnt möchten wir für mehr Spannung im Labyrinth durch eine KI sorgen. Diese soll ebenfalls eine natürliche, realistische Kreatur sein. Deshalb haben wir uns für ein tierisches Wesen entschieden, da ein Monster, ein Drache oder ein Troll nicht in das realistische Bild unseres Labyrinths passen würde. Da das Tier eine entsprechende Größe haben muss, um in den hohen Gängen Aufmerksamkeit zu erlangen und einen gewissen Gruselfaktor ausstrahlen muss, haben wir uns für einen Wolf entschieden. Die roten Augen des Wolfs kündigen diesen bereits in der Dunkelheit an.



Zu Beginn bauten wir das Labyrinth nur aus Wänden in einen schiefer-ähnlichen Design. Dabei merkten wir schnell, dass das Labyrinth sehr eintönig wirkt. Um dem entgegenzuwirken, bauten wir aus diesen Platten verschiedene Variationen und fügten Säulen ein.



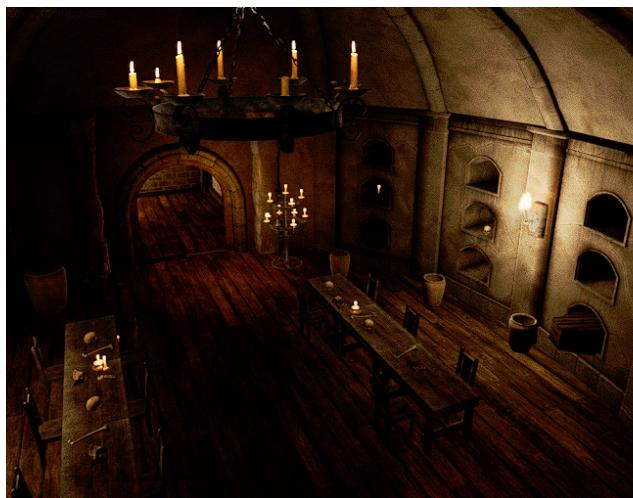
Um weitere Variationen in das Labyrinth einzubauen, haben wir weitere Texturen unterschiedlicher Steinarten und Mauern (Bauweisen) herausgesucht und damit weitere Gänge gebaut. Im Folgenden sind einige Kombinationen abgebildet:



Durch Fenster und Türen haben wir für Abgrenzungen zwischen den Räumen gesorgt, um starke stilistische Brüche zu vermeiden.



Weitere Dekoartikel wie Kerzen, Skelette, Statuen und Kelche, verleihen dem Labyrinth einen Charakter und lassen dem Spieler einen Interpretationsspielraum, wie das Leben in der Vergangenheit in den Höhlen und Gängen des Labyrinths verlief.



Da wir bis auf den Wolf und die Fackeln nur statische Objekte in unserem Labyrinth haben, wollten wir dieses durch etwas Bewegung auflockern. Wir haben uns dazu entschieden Wasser und Lava zu verwenden, da diese Elemente ebenfalls unter der Erde vorkommen. Hierbei war es eine große Herausforderung für uns, realistische Wasser- und Lavabereiche zu bauen, die dem Labyrinth-Stil treu bleiben. Die Höhlen sollten nicht aus einem großen, offenen Raum bestehen und dadurch zu einsichtig sein, aber dennoch nicht erdrückend wirken.

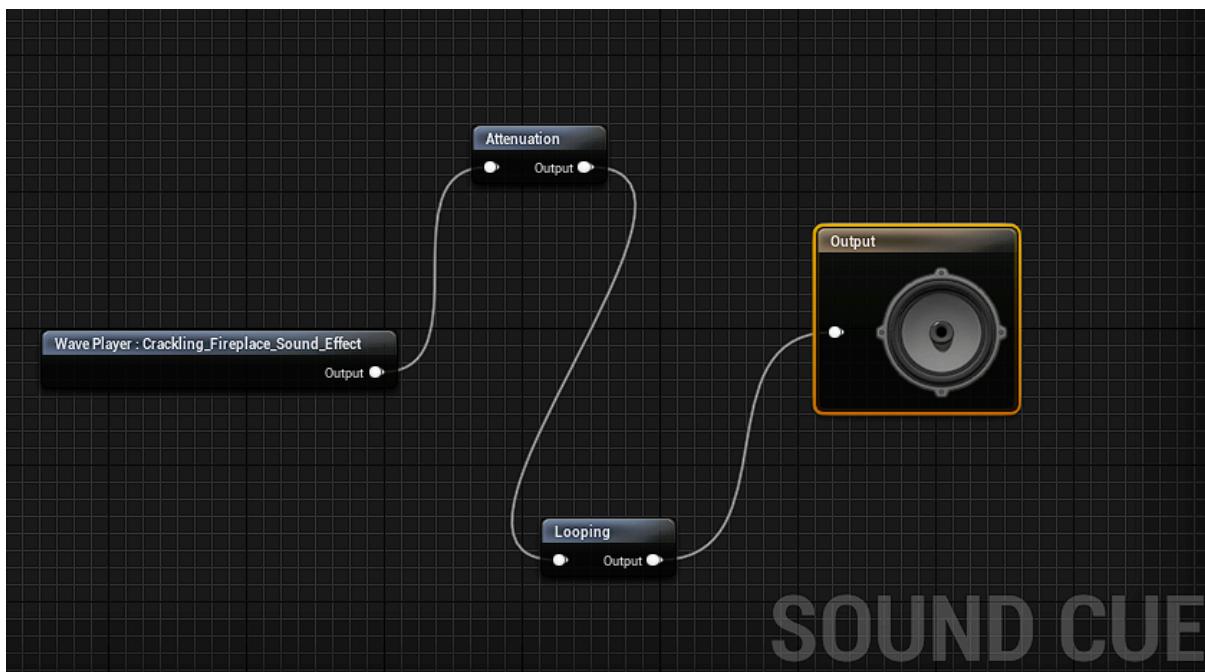


Insgesamt war es uns sehr wichtig dem realistischen und natürlichen Stil beim Bau des Labyrinths treu zu bleiben. Das Spiel soll nicht nur spielaffine Personen ansprechen, sondern auch durch die Bauweise und die Geschichte, die das Labyrinth erzählt, weitere Personen, die in die virtuelle Welt entfliehen möchten, überzeugen.

Sound

Wo macht es Sinn im Labyrinth Sound einzusetzen? Da es sich um ein VR-Spiel handelt, sollte der Spieler nicht mit zu vielen Reizen gleichzeitig überfordert werden. Aus diesem Grund haben wir nur speziell ausgewählten Objekten Sound verliehen. Dem Wolf haben wir Sound verliehen, um den Spieler vor ihm zu warnen. Die Wolfsschritte werden lauter, je näher der Wolf zu einem kommt. Um dem Spieler den Eindruck zu vermitteln, dass er sich tatsächlich in einer Höhle befindet, haben wir den Fackeln und den Wasserfällen Sound verliehen. Auf weitere Hintergrundgeräusche wie das Rauschen des Windes, Geräusche durch weitere Tiere und die eigenen Fußschritte, haben wir aus oben genannten Gründen verzichtet.

Sounddateien werden in der Unreal-Engine im wav-Format eingefügt. Dies hat den Nachteil, dass die Geräusche nicht bearbeitet werden können (z.B. dem Fackelsound einen Radius hinzufügen). Deshalb verwendet man sogenannte .cue-Dateien. Dort können weitere Funktionen wie Attenuation, Loops oder eine zufällige Soundauswahl hinzugefügt werden. Attenuation ist kurz gesagt der Radius, in welchem der Sound wahrgenommen werden kann. Hierbei gibt es zwei wichtige Funktionen, den InnerRadius und die FallOffDistance. Der InnerRadius gibt an, ab welcher Entfernung man die volle Lautstärke hören kann. Die FallOffDistance gibt an, bis zu welcher Entfernung die Lautstärke noch zu hören ist. In diesem Bereich nimmt die Lautstärke mit zunehmender Distanz ab.



Erweiterungen

Während der Entwicklung des VR-Spiels hatten wir noch zahlreiche Ideen um dieses zu erweitern. Vor allem möchten wir mit diesen Erweiterungen auch den Anreiz schaffen, dass das Spiel spannend bleibt und nicht nach einmaligem Durchspielen langweilig wird. Folgende Ideen ließen sich zeitlich leider nicht mehr umsetzen:

Da unser Level inzwischen recht groß wurde, haben wir Savepoints eingebaut, sodass man beim Verlust seines Spiellebens nicht von vorne anfangen muss. Eine weitere Idee wäre, das Spiel in mehrere kleinere Level einzuteilen. Dies hätte außerdem den Vorteil, dass ein Level aus mehreren kleineren Teilen zufällig zusammengesetzt werden könnte. Dies wäre eine einfachere Umsetzung der eigentlichen Idee ein Level dynamisch erstellen zu lassen. Dadurch würde bei jedem Spieldurchlauf ein individuelles Labyrinth aufgebaut werden und der Spieler könnte die Wege nicht auswendig lernen. Da für unser jetziges Labyrinth das visuelle Zusammenspiel eine sehr wichtige Rolle spielt und manche Materialien und Gegenstände stilistisch nicht zusammenpassen würden, wäre im momentanen Stil eine komplett zufällige Generierung des Labyrinths schwierig. Eine weitere Idee, die wir diesbezüglich hatten war, einzelne Wände zufällig anzuzeigen oder auszublenden beziehungsweise zufällig die Wand nach rechts oder links zu schieben, sodass der rechte oder linke Gang geöffnet wird.

Eine Hilfestellung zum Erreichen des Ausgangs wäre eine Karte. Hierfür wäre unsere Idee, einzelne Kartenteile im Labyrinth zu verstecken. Der Spieler kann diese Kartenausschnitte einsammeln. Die eingesammelte Karte bietet eine Hilfestellung um den Ausgang zu finden. Je mehr Teile gefunden werden, desto größer ist der Kartenausschnitt und desto einfacher wird es den Ausgang zu finden. Parallel hierzu könnte man die Geschwindigkeit des Wolfs erhöhen, je mehr Kartenteile gefunden wurden, sodass es nicht zu einfach wird den Ausgang des Labyrinths zu finden. Dies steigert die Herausforderung des Spiels, sodass der Spieler zwischen den Vor- und Nachteilen abwägen muss, die Kartenteile einzusammeln.

Um außerdem den Anreiz zu schaffen, das aktuell bestehende Labyrinth häufiger zu spielen und den Schlüssel einzusammeln, um die Tür zum optimalen Weg zu öffnen, könnten ein Timer sowie eine Highscoreliste eingefügt werden.

Um auch das Spiel ohne Cybershoes spielen zu können, sollte zudem die Steuerung mit den Controllern eingerichtet werden. Falls bei möglichen Erweiterungen weitere Gegenstände eingesammelt werden müssen, wäre eventuell das Einsammeln der Gegenstände mithilfe der Controller einfacher, als über die Blicksteuerung. Um das Gefühl des realistischen Laufens weiter zu optimieren, sollte die Laufrichtung nicht aus der Blickrichtung (VR-Brille) entnommen werden, sondern anhand Schrittrichtung der Cybershoes getrackt werden.

Damit das Spiel unterschiedlichen Ansprüchen genügt, sollte die Schwierigkeit am Anfang des Spiels eingestellt werden können. Die Schwierigkeiten könnten sich beispielsweise durch die Schnelligkeit des Wolf und die Anzahl der KI-Gegner unterscheiden.

Des Weiteren wäre ein Mehrspielermodus eine spannende Erweiterung. Dabei gäbe es folgende Möglichkeiten:

- Spieler vs. Spieler: Ein Spieler ist der Wolf und muss seinen Gegner fangen und der andere Spieler versucht aus dem Labyrinth zu entkommen.
- Miteinander gegen den Wolf
- Gegeneinander vor dem Wolf entkommen - wer erreicht des Ausgang als Erster?

Weitere Ideen, die optional eingebaut werden könnten:

- Fallen, z.B. Brücke, die einstürzt
- Rätsel, z.B. um eine Tür zu öffnen
- Weitere Gegenstände einsammeln
- Weitere, verschiedene Gegner (anstatt nur einen Wolf)
- Powerups, z.B. Items, die zufällig rumliegen um beispielsweise 10 Sekunden schneller laufen zu können

Abschluss

Bei Mystaze haben wir ein Projekt begonnen, welches nie enden muss und immer weiterentwickelt werden kann. Für den aktuellen Zeitpunkt haben wir unser Ziel erreicht, ein funktionsfähiges VR-Spiel entwickelt. Das Spiel stellt ein relativ großes Level zur Verfügung, welches der Spieler durchschreiten kann und was durch diverse Extras, wie die Verfolgung des Wolfs nicht langweilig wird.

Leider ruinierte auch dieses Jahr die Pandemie eine MediaNight in Präsenz. Wir hatten uns bereits alle darauf gefreut, das Spiel vorzuführen und die unterschiedlichen Reaktionen zu erfahren. Leider war es nicht möglich den Besuchern die Möglichkeit zu geben, das Spiel selbst zu testen.

Im Nachhinein gibt es wie bei jedem Projekt einige Punkte die wir beim nächsten Mal anders handhaben würden. Durch das riesige Labyrinth mussten wir zusätzliche Features wie die Savepoints einbauen, um dennoch den Spielspaß aufrecht zu erhalten, da ein kompletter Durchlauf ca. 40 Minuten in Anspruch nimmt. Außerdem haben wir dadurch Performanceprobleme. Könnten wir nochmal neu anfangen, würden wir uns für kleinere Maps und mehrere Level entscheiden. Leider haben wir erst gegen Ende herausgefunden, dass sich von den Materialien Instanzen erzeugen lassen. Dadurch hätten wir mehrmalige Kopien der Dateien vermeiden können. Im Nachhinein war es schwer alle Objekte sinnvoll zu benennen. Darauf sollte direkt von Anfang an geachtet werden.

Es hat uns sehr viel Spaß gemacht das Spiel zu entwickeln. Wir konnten unseren Ideen freien Lauf lassen und waren in keiner Weise eingeschränkt. Vielen Dank für diese Freiheit! Am Ende ist ein eindrucksvolles Labyrinth entstanden, in welches man gerne aus der Realität entflieht. Der Spielspaß ist durch diverse Extras ebenfalls gegeben.