Projekt Game-Design

Sommersemester 2023

Bericht Gruppe 02 - Robo Escape

Jonathan El Jusup (cgt104707) [B246b]

Prince Lare-Lantone (cgt104645) [B246b]

Florian Kern (cgt104661) [B246b]

INHALTSVERZEICHNIS

EINLEITUNG

Robo Escape ist ein 2,5-dimensionaler Puzzle-Platformer, in dem der Spieler in die Fußstapfen eines kleinen Roboters tritt, um aus einer unbekannten Einrichtung zu fliehen, in welcher seinesgleichen zerstört wird. Hierzu macht er sich verschiedene Elemente wie Laser und Spiegel zunutze, um Rätsel zu lösen und mit seinen Platforming-Fähigkeiten die Umgebung zu traversieren, alles mit dem Ziel zu entkommen.

SPIELBESCHREIBUNG

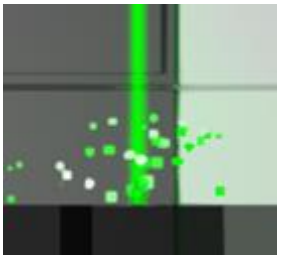
Spielelemente

Es folgt eine kurze Beschreibung aller Spielelemente, mit denen der Spieler interagieren kann. Der Spieler nutzt diese, um Rätsel zu lösen und die Umgebung zu traversieren. Auf dekorative Elemente wie Überwachungskameras, Kabel oder fallende Schrottteile wird hier nicht eingegangen, da diese dekorativen und keinen funktionalen Charakter besitzen.

Laserquellen

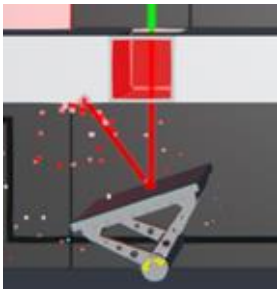
Aus Laserquellen können Laser verschiedenen Typs abgefeuert werden. Eine Laserquelle kann mit einem Trigger, also einem Knopf oder Laser Schalter verbunden werden, welcher diesen konditionell aktiviert oder deaktiviert.

Laser



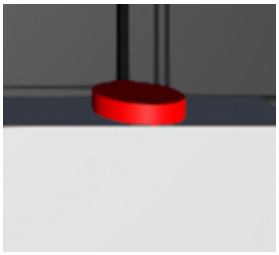
Laser werden von Laserquellen abgefeuert. Es gibt 2 verschiedene Arten von Lasern: Grüne Laser, welche den Spieler keinen Schaden zufügen können und somit sicher sind und rote Laser, welche den Spieler töten, aber auch Schmelzblöcke schmelzen können. Laser können transmittiert, reflektiert oder absorbiert werden.

Spiegel



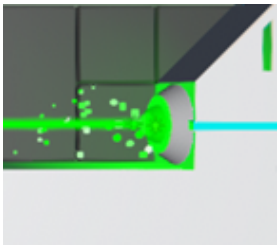
Ein Spiegel reflektiert einen Laser, unabhängig von seinem Typen. Spiegel können mit Knöpfen verbunden werden, damit der Spieler diesen rotieren kann. Dies kann entweder gegen den Uhrzeigersinn, im Uhrzeigersinn oder beides geschehen. Für jede Richtung wird ein Knopf benötigt. Die Rotationsgeschwindigkeit des Spiegels kann eingestellt werden, sodass präzisere Reflektionen möglich sind. Optional kann die Rotation des Spiegels um eine horizontale oder vertikale Verschiebung ausgeglichen werden, damit der Laser immer am gleichen Punkt reflektiert wird, da der Drehpunkt des Spiegels unter der eigentlichen Spiegelfläche liegt. Spiegel können nicht gleichzeitig in beide Richtungen gedreht werden. Wenn beide Trigger gleichzeitig aktiv sind, fängt der Spiegel an zu ruckeln und es fliegen Funken, um dies dem Spieler zu verdeutlichen. Dies ist besonders relevant im 2. Level.

Knopf



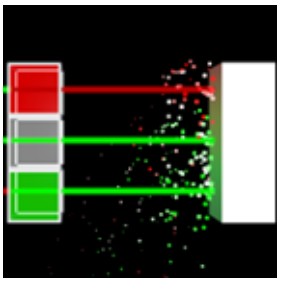
Ein Knopf wird als Trigger betrachtet. Dieser kann mit Effektoren verbunden werden wie Laserquellen oder Spiegeln, um diese zu steuern. Knöpfe haben einen einfachen Zustand, der sich ändert, wenn etwas auf dem Knopf liegt oder nicht.

Laser-Schalter



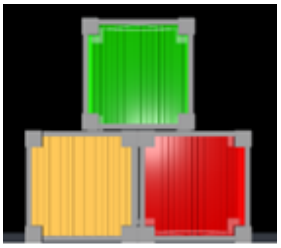
Ein Laser-Schalter ist ein weiterer Trigger, der ebenfalls mit Effektoren verbunden werden kann. Ein Laser Switch besitzt einen internen Timer, welcher den Aktiv-Zustand des Schalters für eine Weile aufrechterhält, bevor sich der Schalter wieder abschaltet. Dieser Timer kann individuell eingestellt werden. Dies stellt sicher, dass der verbundene Effektor leicht verzögert auf Änderungen des Schalters reagiert, insbesondere wenn der Spieler rapide den Schalter aktiviert und deaktiviert.

Glas



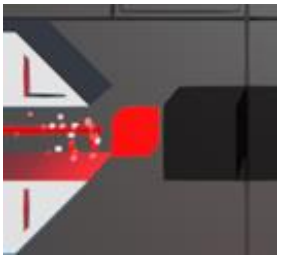
Glasblöcke können Laser passieren lassen. Es gibt Glasblöcke verschiedenen Typs: Der weiße Glasblock lässt Laser durch, ohne dessen Typ bzw. Farbe zu ändern, der grüne Glasblock verändert den Typ entsprechend zu grün. Analog gilt dies für den roten Glasblock.

Kiste



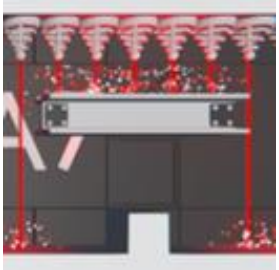
Kisten können benutzt werden, um die Umgebung zu traversieren, um beispielsweise eine Treppe zu bilden, um an höhere Stellen zu gelangen. Außerdem gibt es verschiedene Arten von Kisten: Die Standard Kiste, welche nicht durchsichtig ist und 2 Arten von transparenten Kisten, eine rot und eine grün, welche Laser passieren lassen und dessen Farbe entsprechend ihrer Farbe ändern.

Schmelzblock



Ein Schmelzblock kann von einem roten Laser geschmolzen werden. Ansonsten fungiert dieser wie ein normaler Block, auf dem man laufen kann. Sobald ein roter Laser auf einen Schmelzblock trifft, fängt dieser an, über Zeit zu schrumpfen, bis er komplett geschmolzen ist. Schmelzblöcke können insbesondere für Level genutzt werden, wenn etwas blockiert werden soll (siehe 8. Level), oder die Zeit eine Rolle spielt (siehe 4. Level).

Bewegliche Plattform



Bewegliche Plattformen können genutzt werden, um den Spieler zu befördern, um die Umgebung zu traversieren. Dabei bewegt sich der Spieler auf der Plattform relativ zu dessen Position, rutscht nicht davon ab. Bewegliche Plattformen können ebenfalls genutzt werden, um Laser kurzzeitig zu blockieren und dem Spieler eine Öffnung zu bieten.

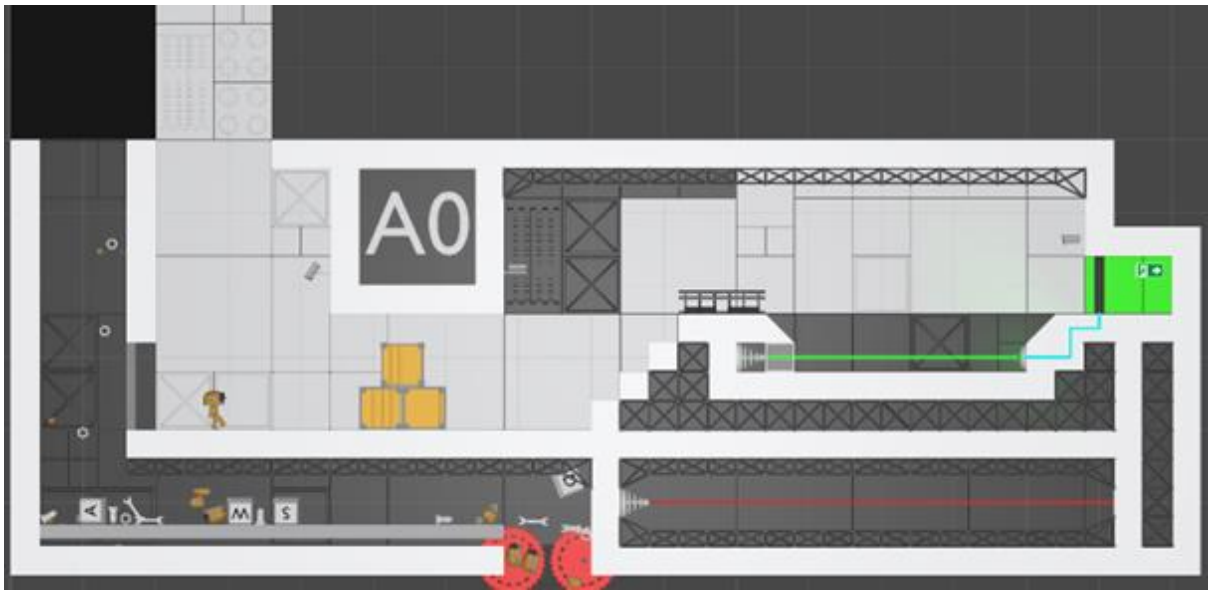
Tür

Türen blockieren den Weg und können von Triggern wie Laser-Schaltern geöffnet werden. Türen werden hauptsächlich genutzt, um den Spieler den Ausgang zu versperren, können aber auch genutzt werden, um Spielelemente wie Laser zu blockieren (siehe 8. Level).

Levelbeschreibung

Das Projekt besteht aus insgesamt 8 Leveln, welche nach und nach neue Mechaniken einführen und zunehmend schwerer werden, um den Spieler herauszufordern. Es folgt eine Beschreibung jedes Levels, wie es aufgebaut wird, wie diese zu lösen sind und was für Gedanken wir uns beim Design dazu gemacht haben.

Level 00 – Intro

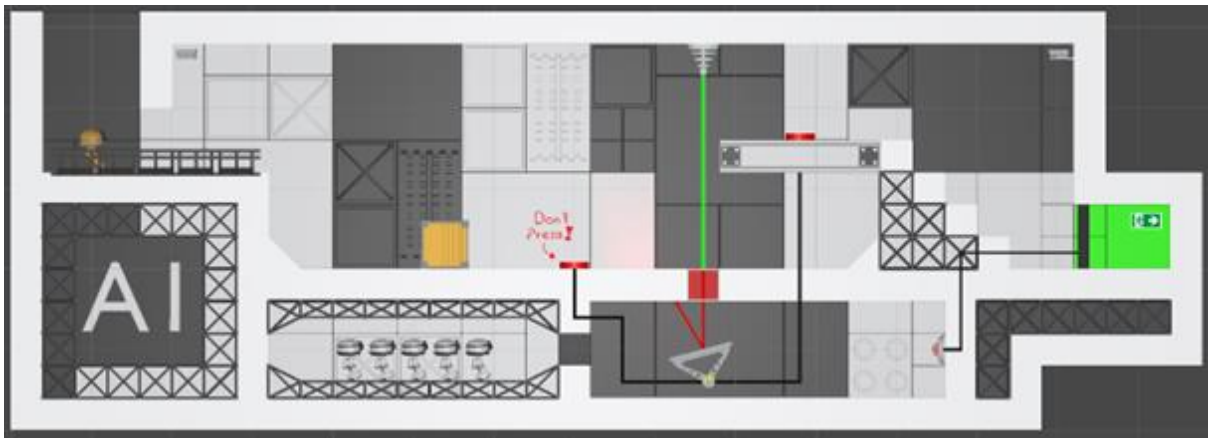


Das Level dient als Einführung und soll keine Herausforderung für den Spieler darstellen. Trotzdem lernt der Spieler die ersten Mechaniken und Steuerungselemente, die später essentiell für ihn sein werden.

Am Anfang fällt der Spieler einen Schacht, gefüllt mit Lasern herab und landet in das erste Level. Links neben ihm befindet sich ebenfalls ein Schacht, in welchem Schrott und Roboterteile herabfallen und geschreddert werden. Dies soll unter anderem zeigen, was mit ihm passiert wäre und dient als narrative Komponente. Außerdem fallen die WASD-Tasten herab, um dem Spieler indirekt zu zeigen, wie er sich bewegen kann, ohne den Spielfluss zu unterbrechen. Angekommen, kann der Spieler Kameras sehen, die ihn anschauen. Dies soll die Situation erwecken, zu jedem Zeitpunkt beobachtet zu werden. Allgemein soll alles zusammen eine feindliche Atmosphäre erzeugen, dem Spieler zeigen, dass er nicht willkommen ist. Um weiterzukommen, muss der Spieler einen Haufen Kisten verschieben, die seinen Weg blockieren. Auch wenn trivial, lernt der Spieler, mit Kisten zu interagieren, diese zu verschieben und auf denen zu springen, um die Umgebung zu traversieren.

In der rechten Hälfte angelangt sieht der Spieler einen grünen Laser am Boden, welchen er nicht umgehen kann. Er soll dadurch gezwungen werden, den Laser zu blockieren, um zweierlei zu realisieren: Der grüne Laser schadet dem Spieler nicht, und der Laser ist für das Öffnen der Tür verantwortlich. Dies wird unter anderem weiter verdeutlicht durch den Laser-Schalter, welcher durch Kabel mit der Tür verbunden ist. Wenn der Schalter durch den Laser aktiviert wird, leuchten die Kabel auch auf. Dies soll dem Spieler Zusammenhänge und aktuelle Stati vermitteln. Wenn sich der Spieler weiter umschaut, sieht er auch einen roten, für ihn noch unzugänglichen Laser. Dadurch soll schon einmal vorgegriffen werden, dass auch andere Lasertypen existieren, aber noch nicht gezeigt werden, was dessen Eigenschaften sind. Am Ende deutlich in grün hervorgehoben befindet sich der Ausgang, den der Spieler betreten muss, um das Level abzuschließen. Zusätzlich wird dieser Bereich durch einen Notausgangsschild markiert, welcher den Namen „Robo *Escape*“ unterstreichen soll.

Level 01 – Execution



Im nächsten Level angekommen wird der Spieler das erste triviale Rätsel zu lösen haben. Er sieht eine Kiste, von der er bereits Bescheid weiß, dass mit dieser interagiert werden kann und einen Button mit der Aufschrift, dass dieser auf keinen Fall gedrückt werden soll. Der Button ist unmittelbar mit einem Spiegel verbunden. Dort läuft noch ein weiteres Kabel nach oben zu einem weiteren Button entlang. Dieser kann aber nicht ohne weiteres erreicht werden. Angenommen der Spieler ist neugierig, dann besteht die große Wahrscheinlichkeit, dass dieser den großen roten Knopf drücken wird. Der Spiegel dreht sich langsam und reflektiert den Laser nach links. Unter dem Spieler sieht er seinesgleichen aneinandergereiht, aber in anderer Farbe, um zwar die Ähnlichkeit zu zeigen, aber auch, dass er anders, der Hauptcharakter ist. Der Spiegel dreht sich weiter und fängt an, einen Block zu schmelzen, welcher den roten Laser und die Roboter Props trennt. Sobald der Block geschmolzen ist, trifft der rote Laser die Roboter Props und sie werden zerstört. Diese ausgehende Gefahr des roten Lasers, soll dem Spieler hierdurch vermittelt werden, zunächst, ohne selbst schaden davon zu nehmen.

Auch wenn der Spieler den Knopf nicht betätigt, muss er dennoch die Box darauf schieben, um auf dieser zu springen, um den oberen Knopf zu erreichen. Das ist aber nicht alles. Wenn der Spieler auf den rechten Knopf tritt, die Box aber noch auf dem linken Knopf liegt, ist der Spiegel blockiert und kann sich nicht bewegen. Dies wird entsprechend visualisiert. Deswegen muss die Box weiter nach rechts geschoben werden, aber auch nicht zu weit, damit diese den Laser nicht blockiert. Nun kann sich der Spiegel nach rechts drehen und den roten Laser auf den Laser-Schalter lenken. Bis hierhin

sollte der Spieler auch realisiert haben, dass die rote Glasscheibe den initial grünen Laser rot gefärbt und somit seine Eigenschaft geändert hat. Somit soll der Laser nur unterhalb des Spielers gefährlich sein. Es handelt sich somit um eine relativ sichere Gegend für den Spieler, in dem er zum ersten Mal die Umgebung bewusst traversieren muss, Knöpfe drückt, um einen Spiegel zu drehen, welcher den roten Laser reflektiert. Ebenfalls lernt er die Eigenschaft des roten Lasers kennen, Blöcke, die von diesen geschmolzen werden können und Gläser, welche die Farbe von Lasern ändern können.

Level 02 – Simple Parkour

Da es sich um einen Puzzle-Platformer handelt, darf es auch nicht an Parkour-Elementen fehlen. Dies ist das erste Level, welches hauptsächlich die Platforming Fähigkeiten des Spielers fordert. Zum ersten Mal kann der Spieler durch einen roten Laser sterben und muss präzise Sprünge tätigen, um das Level zu bestehen. Im Level kommen auch zum ersten Mal bewegliche Plattformen vor, auf denen der Spieler stehen kann. Sobald der Spieler die Platforming Sektion überwunden hat, gibt es noch ein kleines Spiegel Puzzle, in dem der Spieler einen Laser auf einen Laser-Schalter lenken muss, um die Tür zu öffnen. Dieses doch triviale Rätsel soll Parkour- und Puzzle-Elemente mehr miteinander verschmelzen. Bis hierhin könnte der Spieler auch schon bemerkt haben, dass die Laser-Schalter bestimmte Farben haben und somit nur bestimmte Lasertypen annehmen. Dies wird im nächsten Level noch wichtig.

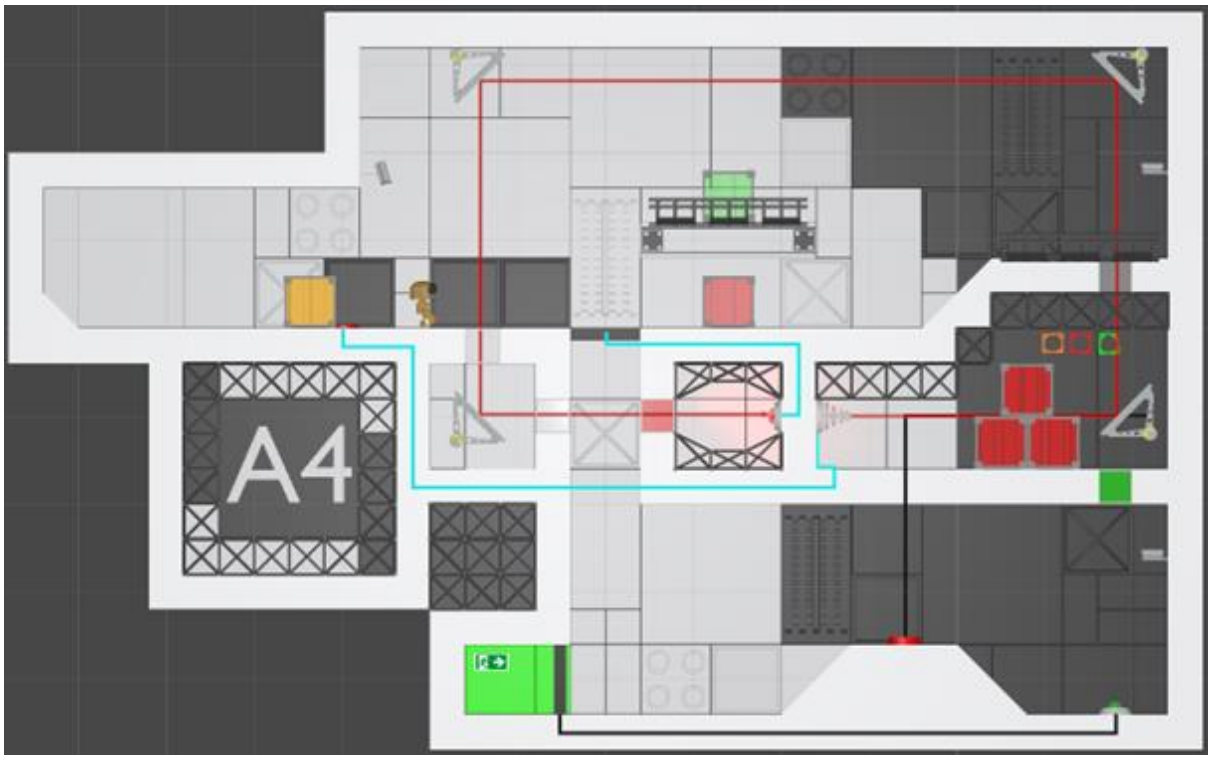
Level 03 – Melting Blocks

Der Spieler hatte bereits erste Erfahrungen mit Schmelzblöcken sammeln können. Nun handelt es sich um ein Level auf Zeit. Unter ihm werden eine Reihe Schmelzblöcke nacheinander geschmolzen. Ohne sie kommt er nicht weiter und muss sich demnach beeilen. Beim Traversieren der oberen Sektion bemerkt der Spieler bestimmt die rote transparente Kiste, mag sich aber vielleicht noch nichts dabei denken. Wenn er diese ignoriert und in die untere Sektion absteigt, wird er einen grünen Laser sehen, welches einen roten Laser Schalter nicht aktiviert. Es wird ein roter Laser benötigt. Um das nochmal extra klar zu machen, findet sich an der Wand ein Hinweis in Form einer roten Kiste. Aber selbst, wenn der Schalter aktiv wäre, könnte der Spieler den Ausgang dennoch nicht erreichen. Demnach erfüllt die rote transparente Kiste einen doppelten Zweck, nämlich den, dass der Spieler den Ausgang erreichen kann.

Wenn der Spieler nun ohne die Kiste unten angelangt ist, steckt er fest. Hier soll dem Spieler klargemacht werden. Dass er das Level neu starten kann. Hierfür befindet sich unter ihm ein Laufband, wieder mit Schrott und Roboter Teilen, die unter ihm befördert werden. Darunter aber auch eine R-Taste, welche zum Neustarten des Levels gedrückt werden kann. Hier soll die Steuerung wieder indirekt klargestellt werden. Beim zweiten Versuch mag der Spieler die Kiste nun nicht mehr ignorieren. Er muss die Kiste von oben nach unten bringen, doch das Loch, durch das er kam ist nicht

groß genug für die Kiste. Demnach muss der Spieler die Kiste an der richtigen Stelle auf den Schmelzblöcken bringen, sodass die Kiste nach unten fällt, sobald die Blöcke geschmolzen wurden. Hierbei muss die Platzierung der Kiste stimmen. Dennoch sollte der Spieler auf die Zeit achten, die ihm ausläuft.

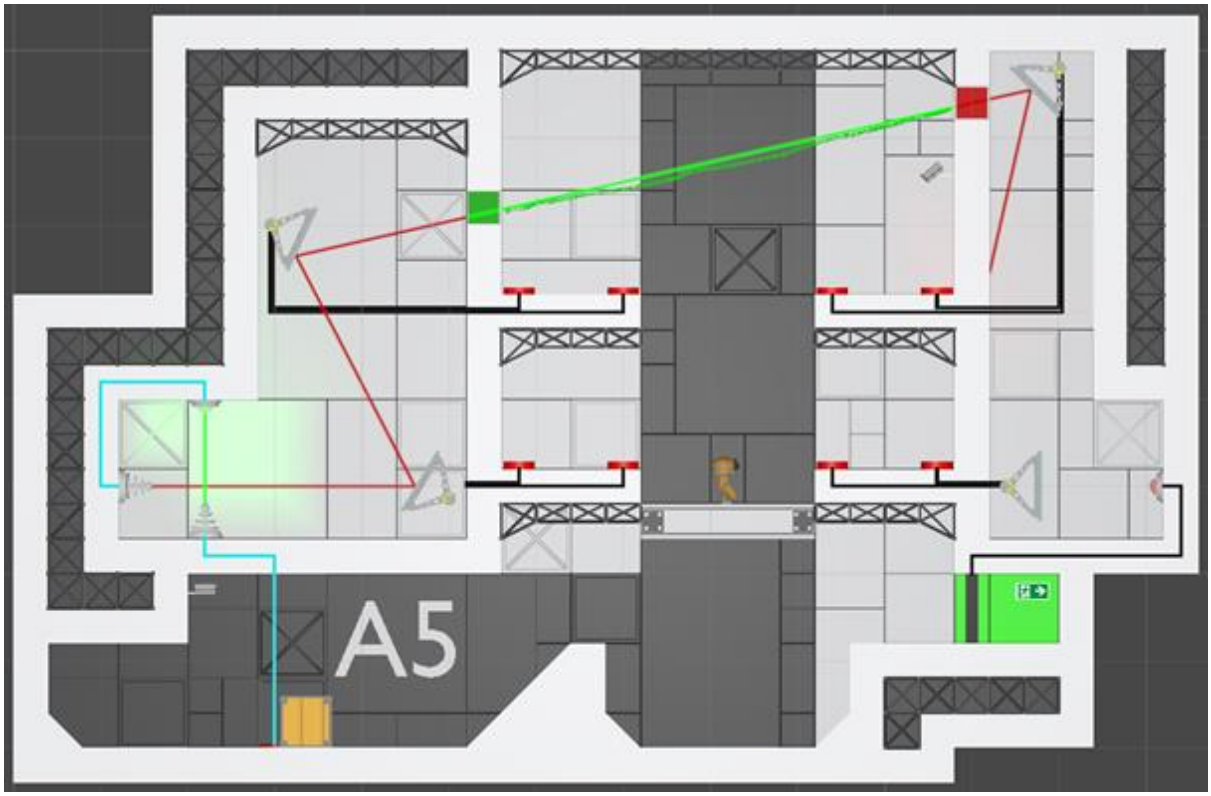
Level 04 – Box Puzzle



Nun hat der Spieler die rote transparente Kiste gelernt und was diese macht. In diesem Level kommt das Gegenstück, die grüne transparente Kiste, hinzu. Hierbei handelt es sich um das erste etwas herausfordernde Level. Im Level befinden sich zunächst eine normale Kiste und ein Knopf. Wenn dieser gedrückt wird, sieht der Spieler einen roten Laser und eine Tür, die sich öffnet. Der Spieler kommt jedoch nicht durch, da der rote Laser seinen Weg blockiert. Geht er weiter, sieht er 2 weitere transparente Kisten; eine ist rot, die andere grün.

Dieses Rätsel erfordert 2 Realisationen. Die erste ist die, dass der Knopf betätigt werden muss, sobald der Spieler schon auf der rechten Seite des roten Lasers angelangt sein muss. Wenn der Spieler dies realisiert hat, mag er darauf kommen, die 3 Kisten nebeneinanderzulegen und von rechts nach links auf den Knopf zu schieben. Wenn er dies erreicht hat, öffnet sich nun die Tür und er kann durch die Tür nach unten springen. Doch gibt es noch ein letztes Hindernis, welches zu überwinden gilt. Wenn der Spieler die 3 Kisten falsch angeordnet hat, mag es sein, dass der Laser unter der Tür ebenfalls rot ist und den Weg nach der Tür wiederum blockiert. Hierbei ist es wichtig, den roten Laser durch die grüne Kiste grün zu machen. Ein Tipp für die richtige Reihenfolge findet sich am rechten Ende des Levels. Sobald der Spieler unten angelangt ist, muss er nur noch einen Spiegel drehen, um die Ausgangstür zu öffnen.

Level 05 – Reflection Puzzle



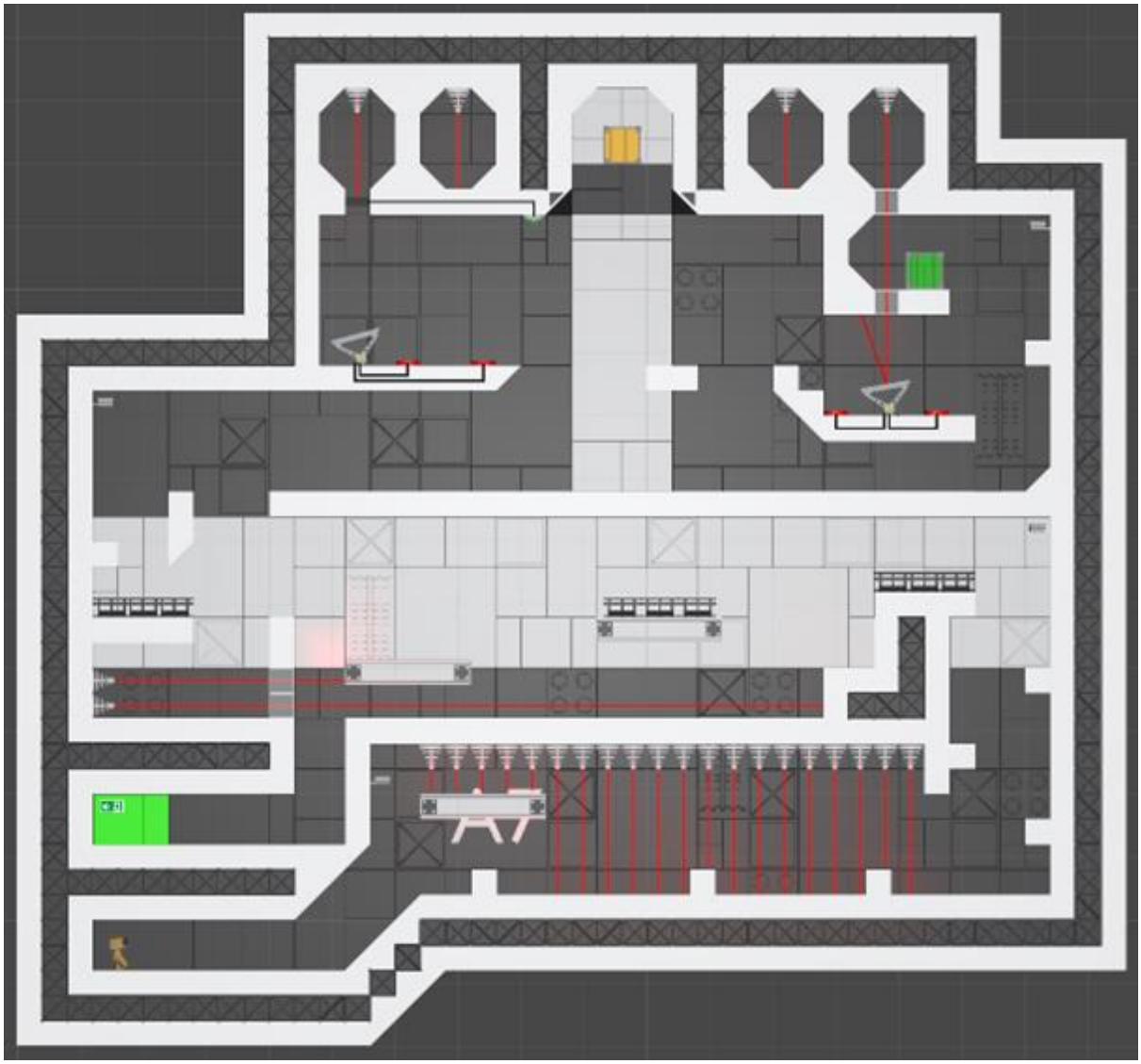
In diesem Level wird der Fokus mehr auf Reflektionen gelegt. Hierbei muss der Spieler 4 Spiegel präzise drehen, um einen Laser von einem Ende zum anderen zu reflektieren, um die Ausgangstür zu öffnen. Es ist wichtig zu verstehen, dass sich falsche Spiegel Rotationen in der Reflektion-Kette akkumulieren. Deswegen existiert an der Hälfte ein schmaler Durchgang, damit der Spieler sich bemüht, den Laser hierdurch zu reflektieren. Dies wird extra durch einen grünen Strahl an der Wand als Tipp markiert. Wenn der Laser schon hierdurch scheint, können auch die restlichen Spiegel eingestellt werden, um am Ende die Tür zu öffnen.

Level 06 – Hard Parkour

Um den Spieler eine Pause von den Rätseln zu geben, darf sich dieser jetzt an einem schweren Parkour-Level versuchen. Die Schwierigkeit liegt hauptsächlich in präzisen Sprüngen und in Geduld, das Level nicht zu schnell zu durchlaufen. Hierfür muss der Spieler die Umgebung über Plattformen und bewegliche Plattformen traversieren und nicht fallen. Wenn er fällt stirbt er durch einen roten Laser am Boden. Nennenswert wäre hier eine bewegliche Plattform, die zeitweilig unter einem roten Laser schwebt. Hierbei muss der Spieler aufpassen und auf das Timing seiner Sprünge achten.

Sobald das Level seine Vertikale erreicht, gelangt der Spieler zu seinem größten Hindernis. Ein roter Laser blockiert seinen Weg, auf eine Plattform zu springen. Hin und wieder wird dieser aber durch eine vertikale bewegliche Plattform blockiert und der Spieler kann auf die Plattform. Was aber nun? Es ist vorgesehen, dass der Spieler mit richtigem Timing auf die bewegliche vertikale Plattform springt, sobald diese den roten Laser blockiert. Da dies schwerer zu realisieren sein könnte, befindet sich an der Wand über der beweglichen Plattform wieder ein Tipp, um den Spieler zu zeigen, dass der diese Plattform nutzen kann und muss, um weiterzukommen. Wenn der Spieler diese Herausforderung überwunden hat, ist es ein Leichtes, den Rest zu überwinden, um das Level abzuschließen und ins finale Level zu gelangen.

Level 07 – Finale



Nun ist der Spieler in das finale Level angelangt. Er wird direkt durch eine Reihe roter Laser begrüßt, welche seinen Weg blockieren. In regelmäßigen Abständen überdacht eine bewegliche Plattform den Spieler und schützt diesen vor den Lasern, aber auf dem Boden befinden sich auch kleine bewegliche Blöcke, welche den Spieler in seiner Bewegung stören sollen. Wenn der Spieler doch diese Sektion überwunden hat, gelangt er an eine ihm vom letzten Level bekannte Stelle.

Er muss auf eine bewegliche Plattform springen, die hin und wieder unter roten Laserstrahlen geht. Hier ist das Timing seiner Sprünge relevant. Hier sieht der Spieler aber auch schon den Ausgang. Dieser ist aber durch gerade diese roten Laser blockiert. Er braucht also etwas, dass diese Laser blockiert, damit der Weg zum Ausgang frei ist. Geht der Spieler weiter, findet er sein passendes Werkzeug, eine Kiste.

Diese Kiste steckt jedoch oben an der Decke fest. Hierfür müssen die Blöcke an der Decke geschmolzen werden. Dafür findet der Spieler 2 Laser. Der rechte darunter kann sofort erreicht werden, der linke wird noch blockiert. Doch kann der Spieler den rechten Laser nicht benutzen, um die Blöcke zu

schmelzen, da der Knopf für die Linksdrehung vom roten Laser blockiert wird. Schaut sich der Spieler sich etwas um, sieht der, dass der Laser durch eine grüne transparente Kiste grün gemacht werden kann. Damit kann er den Knopf erreichen, welchen den Spiegel nach links dreht. Schaut er weiter, sieht er auch einen grünen Laser-Schalter, den er mit diesem Laser aktivieren kann. Wenn der Spieler dies geschafft hat. Öffnet sich eine Tür links oben und der linke rote Laser scheint durch. Dieser kann dann mit einem Spiegel genutzt werden, um die Schmelzblock-Schichten zu schmelzen, um die Kiste an der Decke zu befreien.

Ist die Kiste nun frei, fällt diese auf den Boden und kann vom Spieler geschoben werden, um die Laser in der letzten Sektion zu blockieren. Sobald diese blockiert wurden, ist der Weg zum Ziel frei und der Spieler beendet das finale Level und somit das Spiel.

PROJEKTORGANISATION

Die ursprüngliche Planung des Projektes bestand in der Wahl der Mechaniken und dem Fokus auf eine Umsetzung des Projektes, die dem Spieler indirekt den Weg weist: Von der Gestaltung der Narrative bis hin zur Einführung der einzelnen Mechaniken und wie der Spieler das Spiel spielen soll.

Unsere initiale Projektorganisation ab der Vorstellung unserer Spieleidee bis zum Proof of Concept bot uns eine solide Basis, erlaubte dennoch weitere Änderungen und Flexibilität, in Form von Vertiefung der Mechaniken. Im Nachhinein stellte sich heraus, dass diese Flexibilität von großen Nöten war, da während der Entwicklung ein Gruppenmitglied unangekündigt das Projekt verlassen hatte. Aufgrund der Schadensminimierung mussten wir u.a. im Bereich der Narrative bestimmte Elemente streichen. Dennoch können wir behaupten, dass wir unserer Vision weitgehend treu geblieben sind und die Hauptpunkte unserer Spieleidee umsetzen konnten.

Tutorial

So wurde von Anfang an geplant, das Tutorial des Spiels nicht explizit, sondern implizit einzubinden, indem der Spieler nach und nach neue Mechaniken kennenlernt und diese miteinander kombiniert, um in den Leveln voranschreiten zu können. Dies hat sich bis zum Ende dieses Projektes durchgezogen und wir haben uns stets bemüht, ein solches verstecktes Tutorial in den Spielfluss mit einzubinden, um diesen nicht zu unterbrechen.

So haben wir in bestimmten Leveln Steuerungselemente getarnt, als Schrott herabfallen und schreddern lassen, um den Spieler zu zeigen, wie man sich beispielsweise bewegt oder das Level neu startet. Dies hat sich weitergezogen zu „Decals“ an Wänden, welche den Spieler Tipps geben und diesen den Weg weisen. Dies haben wir besonders an den Stellen angewandt, an denen es die meisten Probleme in den Playtests gab.

Mechaniken

Initial hatten wir uns auf 4 Mechaniken festgelegt. Darunter das Verschieben von Blöcken, das Interagieren mit Schaltern, die Spiegel-Laser-Reflektion und Batterien. Schon bei der Vorstellung der Projektidee stellte sich heraus, dass die ersten beiden Mechaniken nicht als solche wahrgenommen wurden, was uns zunächst verunsichert hatte, da sich unser Projekt um einen Puzzle-Platformer handeln sollte und das Verschieben von Blöcken ein integraler Bestandteil des Spiels sein sollte, damit der Spieler die Umgebung traversieren, Mechanismen beeinflussen oder Laser blockieren kann.

Das gleiche gilt für die Interaktion mit Schaltern in Form von Druck-Plattformen oder solche, die durch Laser aktiviert werden können, um bestimmte Mechanismen beeinflussen zu können, beispielsweise in Form von Öffnen von Türen oder Aktivieren von neuen Laserquellen.

Doch hat sich schon früh in der Entwicklung herausgestellt, dass das Verschieben von Blöcken und interagieren mit Schaltern als der Laser-Mechanik untergeordnet betrachtet werden konnten. So entschlossen wir uns, allein die Laser-Mechanik als Kernmechanik zu werten. Diese Mechanik wurde während des Entwicklungsprozesses immer weiter ausgearbeitet. Dies hat sich im Verlauf der Entwicklung auch enorm ausgezahlt. Anfangs wurde lediglich geplant, dass Laser den Spieler schaden, somit als Hindernis fungieren, aber auch Hindernisse zerstören können. In Kombination mit Spiegeln sollte der Spieler diese Laser als Werkzeug nutzen können, um seinen Weg zu bereiten und mit Mechanismen und Schaltern interagieren. Somit sollten Laser einen wesentlichen Beitrag dazu leisten, Rätsel zu lösen.

Während des Entwicklungsprozesses haben wir uns immer mehr mit dieser Laser-Mechanik auseinandergesetzt und diese weitgehend ausgebaut. So haben wir verschiedene Lasertypen eingeführt; grüne Laser, die dem Spieler keinen Schaden zufügen und rote Laser, welche den Spieler sofort töten. Dual dazu kamen Laser-spezifische Schalter, welche nur einen bestimmten Lasertyp

annehmen, um sich zu aktivieren. Um diese Lasertypen herum bauten wir weitere Interaktionsmöglichkeiten, darunter transparente Gläser, welche den Typen des Lasers ändern können, transparente Boxen, welche neben der gleichen Funktion außerdem zum Traversieren der Umgebung benutzt werden können oder den Typ eines Lasers unter Spieler-Einfluss verändern. Zu einem weiteren, der Laser-Mechanik untergeordneten Element wurde der sog. Schmelzblock, welcher nur durch rote, tödliche Laser geschmolzen werden kann. Zusammen mit Spiegeln hat sich eine tiefgreifende Komplexität entwickelt, welche sich in unseren Leveln widerspiegelt.

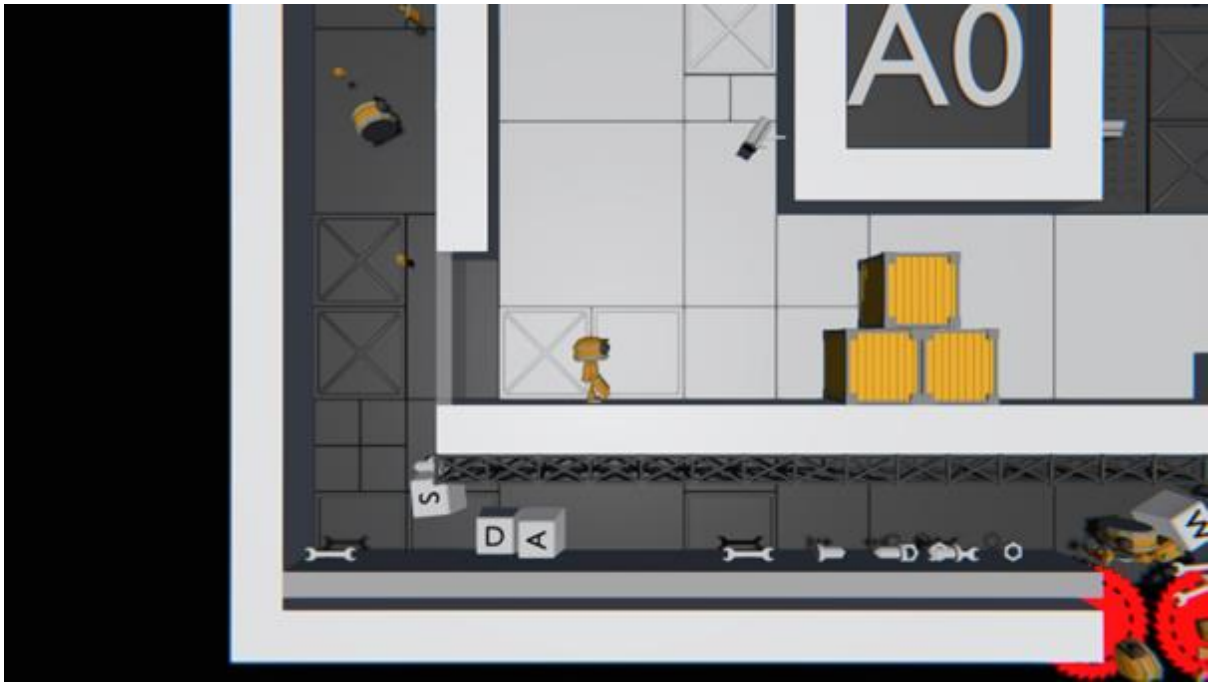
Eine weitere Mechanik war die Nutzung von Batterien, welche aufgehoben und genutzt werden sollten, um Mechanismen zu aktivieren, wenn diese in einem dafür vorgesehenen Sockel abgelegt wurden. Diese Idee nahmen wir bis in die Entwicklung des MVPs mit, ließen nach der MVP-Vorstellung aber davon ab. Obwohl diese Mechanik für weitere Komplexität und Tiefe gesorgt hätte, stellte sich heraus, dass die der Laser-Mechanik untergeordneten Sub-Mechaniken bereits für hohe Komplexität sorgen würden. Ebenso erfüllte eine Kiste, welche auf einen Knopf geschoben wurde, weitgehend die gleiche Aufgabe einer Batterie, welche diese redundant gemacht hätte.

Eine letzte Idee war die von gegnerischen Wachtürmen, welche auf den Spieler schießen, um seinen Weg zu blockieren. Wir haben früh mit der Idee gespielt und experimentiert, haben diese doch nicht offiziell in das Projekt eingebunden, sondern als experimentell betrachtet. Diese Wachtürme haben wir ebenfalls bis zur Vorstellung des MVPs mitgenommen und uns danach von dieser Idee verabschiedet, da wir uns auf bestehende Mechaniken fokussieren wollten. Wachtürme haben somit nur in unserer Sandbox eine Weile ihren Nutzen gehabt.

Narrative

Anfangs haben wir uns für eine nebenherlaufende Narrative entschieden, welche zwar nicht für den Spieler essentiell sein sollte, aber dem Ganzen mehr Leben und Humor geben sollte. Dies sollte sich hauptsächlich im Level Design wiederfinden, und in Form einer den Spieler überwachenden KI, welche zynische Randbemerkungen machen sollte, wenn der Spieler bestimmte Aktionen machen sollte. Früh wurden wir gewarnt, dass es schwer sei, der KI eine Stimme zu geben. Die Narrative bestand also aus Level Design und der sprechenden KI.

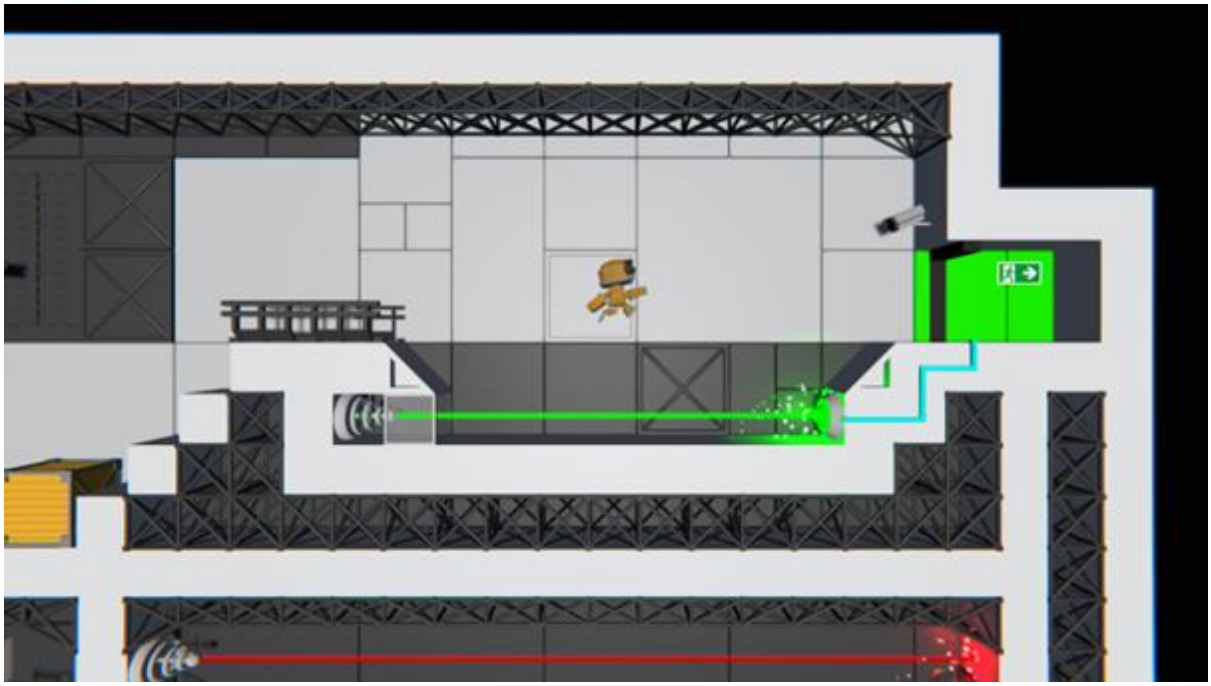
Leider hat sich herausgestellt, dass wir die Narrative nicht so umsetzen konnten, wie wir es ursprünglich geplant haben, aufgrund des unangekündigten Austritts eines Gruppenmitglieds, welcher für den Sound und somit für die sprechende KI verantwortlich sein sollte. Wegen dieser Ungewissheit haben wir uns mehr auf das „World Building“ in Form von Level Design fokussiert. So entstanden Überwachungskameras, welche den Spieler immer beobachteten, wenn eine direkte Sichtlinie bestand, Roboter Props, welche anstelle des Spielers exekutiert wurden, sog. „Scrap Shoots“, wo Roboterteile herabfallen und von Kreissägen geschreddert werden, besonders im Einführungslevel, in dem der Spieler initial fällt, aber seiner Zerstörung entgeht. All dies und mehr sollte eine feindliche Umgebung suggerieren. Es gab aber auch kleinere Elemente, wie Schilder, welche den Ausgang weisen oder „Decals“ an Wänden, die dem Spieler Hinweise geben sollten.



Affordanz

Noch vor der eigentlichen Entwicklung des Projektes standen Affordanz und Erwartungskonformität im Vordergrund. Der Spieler sollte immer wissen, wie sich seine Aktionen auf die Umgebung auswirken und welche Komponenten eines Mechanismus zusammenhängen. Somit sollte sich der Spieler nur um das Lösen von Rätseln Gedanken machen und nicht seine eigenen Aktionen hinterfragen.

Während des gesamten Entwicklungsprozesses haben wir uns stets darum bemüht, dem Spieler viele Informationen zu vermitteln, die er benötigt, um zu verstehen, was er tun soll, was er tun kann und was seine Aktionen bewirken. Beispiele hierfür sind Kabel, welche Schalter und Türen, Spiegel oder Laserquellen verbinden und aufleuchten, wenn diese aktiv sind. Aber auch subtilere Elemente wie Animationen von Knöpfen, Framing von Spielelementen und das Level-Design, welche indirekt Mechaniken vermitteln sollen. Beispielsweise das Einführungslevel, in dem der Spieler einen grünen Laser blockieren muss. So wird ihm direkt am Anfang vermittelt, dass grüne Laser sicher sind, der Schalter vom Laser gesteuert wird und mit der Tür durch Kabel verbunden ist.



Gameplay

Unsere Spieleidee sollte von Anfang an ein Puzzle Platformer sein. In diesem Sinne sollten Platforming Fähigkeiten und Problemlösungsfähigkeiten gefordert werden. Für gutes Gameplay allein reichen aber nicht nur gut durchdachte Mechaniken aus. Diese müssen sich auch entfalten können. Dies war von Anfang an klar und lässt sich durch qualitativ hochwertige Level und Rätsel erreichen. Im Verlauf der Entwicklung haben wir weitere Level eingeführt, welche aufeinander aufbauen, die sowohl Platforming Sektionen, als auch herausfordernde Rätsel bieten. Gerade nach der MVP-Vorstellung haben wir uns mehr auf die Platforming Levels fokussiert, da die Laser-Mechanik und ihre Submechaniken sich bereits bewiesen haben. Bis zu den Prototypen haben wir 8 einzigartige Levels gestaltet, welche neue Mechaniken aufbauend einführen, diese kombinieren und eine Schwierigkeitskurve bieten, welche den Spieler herausfordert.

ARBEITSTEILUNG

Themen Florian Kern

KURZÜBERSICHT

Zeit Gesamt: 127 Stunden

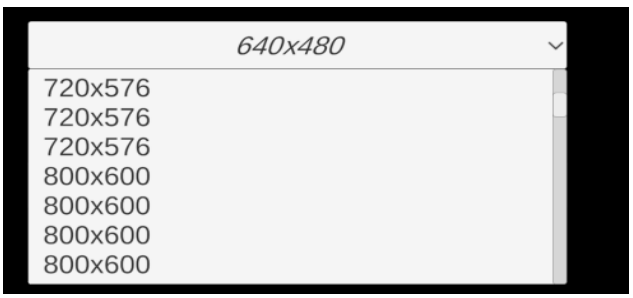
1. Hauptmenü: 20 Stunden
2. Spiel-Sound: 24 Stunden
3. Tür-Animation und Laden des nächsten Levels: 5 Stunden
4. Tod des Spielers durch Laser mit Prince: 4 Stunden
5. Animation beim Übergang in ein neues Level: 2 Stunden
6. Knopf-Animation: 3 Stunden
7. Schmelz-Mechanik mit Prince: 3 Stunden
8. Dev-Skript zum manuellen Wechseln zwischen Leveln: 1 Stunde
9. Character Movement mit Prince: 6 Stunden
10. Einarbeitung in Unity zu Projektstart: 5 Stunden
11. Pausemenü: 3 Stunden
12. Auswertung der Playtests vom MVP: 9 Stunden
13. Level-Design von Level 02 mit Prince: 7 Stunden
14. Projektorganisation: 15 Stunden
15. Schreiben der Dokumentation und Dokumentation des Codes: 20 Stunden

ERKLÄRUNG

1. Das Hauptmenü ist ein bedeutender Bestandteil jedes Spiels. Es ist das Erste, das vom Spieler gesehen wird, sobald das Spiel gestartet wird und muss dementsprechend einladend wirken und einen direkten Eindruck darüber vermitteln, was das grundlegende Thema des Spiels ist. Zum PoC war noch kein Hauptmenü erstellt worden, da zu diesem Zeitpunkt nur gezeigt werden sollte, dass auf der Spielidee aufgebaut werden kann und die vorgestellten Mechaniken grundlegend funktionieren. Zum MVP wurde mir die Aufgabe übertragen, ein Hauptmenü zu designen und die damit zusammenhängenden Funktionalitäten (z. B. Grafikeinstellungen) zu implementieren. Ich entschied mich für eine Trennung zwischen Hauptmenü und einem Optionsmenü, das über einen Button erreicht werden sollte. Dafür verwendete ich einen Canvas als Container, der das Haupt- und Optionsmenü halten sollte.

Für beide Menüs verwendete ich jeweils ein schwarzes Image, das sich über den Canvas erstreckte. Im Hauptmenü platzierte ich vier Buttons. Einen, um das Spiel zu starten, einen, der das Optionsmenü aktiviert und das Hauptmenü deaktiviert, einen Button, der das Sandbox-Level öffnet und einen letzten, womit das Spiel beendet werden kann. Für die Funktionalität hinter den Knöpfen legte ich die Klasse "menu" an. Die Implementierung war sehr simpel, jeder Button verfügt über eine On-Click-Funktion, der man im Inspector eine Funktion zuweisen kann und ausgeführt wird, sobald der Button geklickt wird. Ich legte ein Empty-Objekt in der Szene an, verlieh diesem das menu-Skript und konnte, nachdem ich im Inspector das Objekt hinterlegt hatte, nun auf die Funktionen zugreifen. Sowohl der Knopf zum Starten des Spiels als auch der Knopf für das Sandbox-Level hatten eine sehr ähnliche Implementierung, die Unity-Engine bietet dem Entwickler mit der Klasse *SceneManager* die Möglichkeit, eine Szene mithilfe eines Namens oder eines Indexes zu laden. Zum Starten des Spiels holte ich mir dafür den Build-Index des aktuellen Levels und addierte eins drauf, um das erste Level zu laden. Für das Laden der Sandbox-Szene verwendete ich den Namen der Szene. Zum Beenden des Spiels bietet die Klasse *Application* von Unity die Methode "Quit", die das Spiel beendet. Zum Wechseln zwischen dem Optionsmenü und dem Hauptmenü wurde es ein wenig komplizierter, da man nicht ohne Weiteres auf die beiden Objekte im Code zugreifen kann. Hierzu werden zwei weitere Variablen benötigt, die das Options- und Hauptmenü repräsentieren und im Editor dem Skript zugewiesen werden können. Man könnte diese Variablen public, also öffentlich zugänglich, deklarieren, jedoch gibt es hierbei eine elegantere Lösung: *SerializeField*. Hiermit bietet die Unity-Engine die Möglichkeit, dass private deklarierte Variablen trotzdem im Editor zugänglich sind und ihnen ein passendes Objekt zugewiesen werden kann. Nachdem dieser Schritt getan wurde, brauchte bei einem Klick auf den Button nur das eine GameObject als aktiv gesetzt werden (in dem Fall das Optionsmenü) und das andere als inaktiv gesetzt werden (hier das Hauptmenü) und es wurde das Fenster für das Optionsmenü geöffnet. Somit war die Funktionalität des Hauptmenüs erledigt und ich konnte das Optionsmenü bearbeiten.

Für ein Optionsmenü sind viele Einstellungen denkbar: V-Sync, Grafikeinstellungen, Einstellungen für die Bildschirmauflösungen, Audio-Einstellungen, Fenster-Modus und vieles mehr. Wir entschieden uns dazu, die wohl gängigsten Einstellungen zur Verfügung zu stellen: Grafikeinstellungen, Einstellungen für die Bildschirmauflösung, das Wechseln zwischen Fenster- und Vollbildmodus und Audio-Einstellungen. Für Grafikeinstellungen und Bildschirmauflösung entschied ich mich dazu, dass Dropdown-Menüs verwendet werden.



Was die Einstellungen für die Bildschirmauflösungen betrifft, lernte ich in der Veranstaltung Game-Engines, dass man die Bildschirmauflösungen am besten dynamisch zur Laufzeit lädt und abhängig vom System des Benutzers einstellbar machen sollte. Die Veranstaltung nutzte zwar die Unreal-Engine, jedoch war dies zu meiner Freude mit der Klasse *Screen* der Unity-Engine genauso realisierbar. Ich schrieb eine Methode, die mithilfe einer Liste und einem Array des Typs *Resolution* arbeitet, das von der *Screen*-Klasse zurückgegeben wird, wenn man auf die Auflösungen zugreift, die der genutzte Monitor unterstützt. Ich musste also nur über die Liste der Auflösungen iterieren und die Bildschirm-

Breite und -Höhe zu einem String konkatenieren und dem Dropdown-Menü hinzufügen. Jedoch entstand dadurch ein Problem, das ich mir zunächst nicht erklären konnte. Im Dropdown-Menü wurden dieselben Auflösungen mehrfach im Menü abgelegt.

Wie sich herausstellte, stammt dieses Problem daher, dass Unity zwischen Auflösungen in unterschiedlicher Hz-Rate unterscheidet. Da ich keine genauere Filterung der gewünschten Auflösungen vorgenommen habe, erhielt ich also sämtliche Auflösungen für verschiedene Hz-Raten meines Monitors. Es wurde also in einem vorherigen Schritt nach den Auflösungen, die zu der aktuell eingestellten Hz-Rate des Monitors passen, gefiltert und damit konnte das Problem gelöst werden. Damit war die Methode fertig geschrieben und ich konnte sie in der Start-Methode des Skripts verwenden, damit sie beim Laden des Skripts ebenfalls ausgeführt wird. Möchte der Benutzer nun die Auflösung ändern, muss eine weitere Methode geschrieben werden, die aufgerufen wird, sobald sich der eingestellte Wert des Dropdown-Menüs ändert. Dafür schrieb ich eine weitere Methode, die genau darauf ausgelegt ist. Sie nimmt den Index des ausgewählten Eintrags des Menüs an, spaltet den Inhalt des Eintrags in Höhe und Breite und parsed dessen String-Werte in Int-Werte. Mithilfe der Klasse *Screen* können diese Werte verwendet werden, um die Auflösung anzuwenden.

Für die Grafikeinstellungen musste nicht viel beachtet werden, da in den Projekteinstellungen bereits eingestellt werden kann, welche Einstellungen unterstützt werden. Wir haben uns dazu entschieden, die Einstellungen “low”, “medium”, “high” und “ultra” zu unterstützen. Mit diesem Wissen habe ich dem Dropdown-Menü diese vier Einträge vergeben und eine Methode geschrieben, die den Index des ausgewählten Eintrags entgegennimmt und diesen verwendet, um mit der Klasse *QualitySettings* das Qualitäts-Level einzustellen, da der Index mit den Einträgen in den Projekteinstellungen übereinstimmt, brauchte hier nichts weiter getan zu werden.

Die nächste Einstellung ist das Wechseln zwischen Fenstermodus und Vollbildmodus. Hierbei handelt es sich um einen einfachen Toggle, der dieses Wechseln umsetzen sollte. Hierbei wird einfach nur der Status des Toggles an eine Methode übergeben, die anschließend erneut mit der *Screen*-Klasse den Vollbildmodus aktiviert oder deaktiviert.

Neben der Kritik der Tests zum MVP (Näheres dazu im 12. punkt), die die noch nicht 100% fehlerfreie Umsetzung der Einstellungen beschrieb, wurde häufig erwähnt, dass die herabfallenden Tasten mit den Buchstaben im ersten nicht unbedingt ausreichen, um die Steuerung zu erklären. Also beschloss ich, ein weiteres Fenster für die Steuerung anzulegen, dass aus dem Optionsmenü und im Pausemenü (siehe Punkt 11) aufgerufen werden konnte. Hierbei handelte es sich ähnlich zum Options- und Hauptmenü einfach nur um ein Fenster, das durch einen Button aktiviert werden müsste, während ein anderes deaktiviert wird.

Abschließend brauchten wir noch einen Button, der den Spieler wieder in das Fenster des Hauptmenüs führt. Dieser aktivierte das Fenster des Hauptmenüs und deaktivierte das des Optionsmenüs.

Zum Einstellen von Audio bzw. Musik und Spezialeffekte haben wir Slider verwendet, die wir in der Farbe und Größe so angepasst haben, dass es uns gefiel. Wir entschieden uns dafür, die Farbe des Sliders rot darzustellen, da es mit dem Thema des Spiels bzw. den Lasern gut harmoniert. Dieselbe Überlegung fand auch beim Hovern über den Knöpfen statt.

Die Umsetzung der Funktionalität der Slider wird näher im nächsten Kapitel erläutert.

In der Version vom MVP verwendeten wir zum Anwenden und Verwerfen von neuen Einstellungen zwei Buttons. Diese haben wir für den Prototypen verworfen, weil sie vieles komplizierter gemacht haben als nötig. Es musste mithilfe von Variablen gespeichert werden, ob sich ein Eintrag verändert hat, welcher genaue Eintrag von welchem Menü sich verändert hat, und welcher Wert vor der Veränderung eingestellt war, um ihn bei einem Verwerfen der Änderung wieder richtig darzustellen. Die Klasse wurde durch all diese Fälle deutlich unübersichtlicher, weshalb wir uns dafür entschieden haben, dass Änderungen direkt angewendet werden.

Was mir sehr wichtig war, ist, dass die vom Benutzer festgelegten Einstellungen auch auf irgendeine Art gespeichert werden und nicht immer aufs Neue eingestellt werden müssen. Hier bietet die Unity-Engine die Klasse *PlayerPrefs* an. Es handelt sich hierbei um eine Klasse, die dafür da ist, um eben genau diese Einstellungen zwischen einzelnen Sitzungen zu speichern. Hier können Strings, Floats und Integer gespeichert werden, um sie dann an gezielten Stellen verwenden zu können. Die PlayerPrefs bieten die Methoden SetFloat und GetFloat, beim Verändern einer Einstellung, z. B. der der aktuellen Auflösung, wird der Index des Eintrags in Kombination mit einem Schlüssel gespeichert, der jederzeit mit diesem Schlüssel abgerufen werden kann. Beim Starten des Spiels bzw. wenn die Szene des Hauptmenüs lädt, werden mit der Start-Methode des Menu-Skripts die richtigen Einträge der Dropdown-Menüs gesetzt, indem die gespeicherten Werte mit ihren Schlüsseln verwendet werden.

Abschließend brauchte das Hauptmenü noch einen visuellen Schliff. Für den MVP blieb nicht genug Zeit, um eine wirklich ansprechende Szene zu gestalten, weshalb ich mich dazu entschied, ein Logo und den Namen des Spiels im Hauptmenü zu verwenden, welche ich generieren ließ (siehe Kapitel “Externe Assets”).

Nach dem MVP besprachen Jonathan und ich uns darüber, dass das Menü eine Umgestaltung brauchte, da es nicht wirklich einladend aussah und nichts über das Spiel ausgesagt hat. Es sollte eine Live-Szene geschaffen werden, die gleichzeitig als Hauptmenü dienen sollte. Ich versuchte mich daran, scheiterte aber darin, ein zufriedenstellendes Design hervorzubringen, weshalb Jonathan das übernahm, da dieser einen ausgeprägteren Sinn für Gestaltung hat als ich.

2. Musik und Spezialeffekte sind wahre Bühnenträger und verhelfen dem Spiel dazu, erst richtig lebendig zu werden. Ursprünglich besetzte unser viertes Gruppenmitglied Dinh Nguyen die Rolle des Sound Design Lead. Mit dessen Ausscheiden musste diese Rolle neu vergeben werden. Da ich mit dem Menü beschäftigt war, das Hauptmenü direkt mit Musik in Verbindung steht und ich Interesse an der Umsetzung von Musik im Spiel hatte, entschied ich mich dazu, diese Rolle anzunehmen.

Ein großer Teil dieser Rolle ist dabei das Recherchieren nach Musik, die zu dem Spiel passen, womit ich insgesamt eine Menge Zeit verbrachte. Zum Zeitpunkt der Tests vom MVP hatten wir noch keine Spezialeffekte im Spiel eingebunden, was uns auch als Kritik nahegelegt wurde. Es kam

dementsprechend vieles Neues hinzu: AudioMixer, AudioSources und AudioClips, mit dessen Funktionen ich mich erstmal auseinandersetzen musste, bevor ich das Audio z. B. mit den Slidern aus dem Optionsmenü oder dem Pausemenü verbinden konnte. Es stellte sich heraus, dass jede Audioquelle einen Output in Unity besitzt. Ohne weiteres Zutun vom Entwickler ist das ein einziger Audiomixer, der sämtliches Audio steuert.

Möchte man also den Sound in seiner Lautstärke manipulieren, kann man dies tun, indem man den entsprechenden Audiomixer anspricht und einen eigenen Wert für dessen Lautstärke festlegt. Ich legte also zunächst ein Empty-Objekt in der Szene an und habe diesem eine AudioSource sowie einen AudioClip gegeben. Wie bereits in Punkt 1 angesprochen, lassen sich Objekte in Unity mithilfe von SerializeField im Code bearbeiten, also legte ich ein weiteres Skript an, das sich nur mit der Kontrolle von Audio beschäftigen sollte und hängte es dem Objekt mit der AudioSource an. Ich verband den Slider des Optionsmenüs und den Audiomixer mit dem Skript und schrieb eine Methode, die einen Float entgegennimmt und aufgerufen wird, sobald man den Wert des Sliders verändert. Ich speicherte den Wert in den PlayerPrefs, um den Wert des Sliders wieder an die Stelle bringen zu können, wo der Benutzer ihn in seiner letzten Sitzung gelassen hat. Damit der Wert des AudioMixers zugreifbar ist, muss bei diesem erst ein Parameter offengelegt (exposed) werden. Nun konnte ich mit der SetFloat-Methode des AudioMixers auf diesen Parameter zugreifen und den Wert des Audio-Sliders verwenden, um die Musik lauter oder leiser zu stellen. Dachte ich zumindest. Die Veränderung des Sliders zeigte keinerlei Auswirkungen auf die Musik, es sei denn, ich habe sie komplett ausgestellt. Auf der Suche nach einer Lösung für dieses Problem, stieß ich darauf, dass die Werte des AudioMixers logarithmisch sind, während die des Sliders linear verlaufen. Es war also eine logarithmische Konversion nötig (siehe Kapitel “Externe Assets und Anleitungen - Logarithmische Konversion von Audio-Slidern”) und die Werte des Sliders mussten angepasst werden. Danach verhielt sich alles wie erwartet und die Musik des Hauptmenüs konnte nun nach Belieben eingestellt werden. Das Vorgehen für die Musik im Spiel war sehr ähnlich, mit dem Zusatz, dass nun auch Spezialeffekte benötigt wurden, deren Lautstärke separat einstellbar sein musste. Damit der Spieler also bereits im Hauptmenü diese steuern konnte, legte ich einen weiteren Slider für die Spezialeffekte (SFX) an und verband ihn mit dem Skript. Damit aber Musik und SFX separat einstellbar sein können, musste ich einen neuen AudioMixer erstellen und ebenfalls dem Skript zugänglich machen. Somit konnten nun sowohl Musik als auch SFX separat eingestellt werden.

Der wohl schwerste Teil war jedoch herauszufinden, wie man mehrere verschiedene Sounds im Spiel unterbringt und diese genau dann abspielt, wenn man sie braucht. Es muss sich die Frage gestellt werden, wo und wie man die Sounds alle zusammenträgt, um auf sie nach Bedarf zugreifen zu können. Die Idee, dass möglicherweise mehrere Sounds in einer Liste zusammengefasst werden, um dann mit einem Index auf sie zugreifen zu können und diese abspielen zu können war eine Idee, die ich hatte, aber die Umsetzung fiel mir schwer, weshalb ich im Internet nach Leuten suchte, die möglicherweise dasselbe Problem wie ich hatten. Bei meiner Recherche stieß ich auf den Content-Creator “Brackeys”, welcher eine Menge Inhalte zu der Entwicklung in Unity machte, darunter auch Videos im Umgang mit Audio. Bei einem seiner Videos unterteilt dieser in zwei Klassen: Einer Sound-Klasse und einem Sound-Manager. Bei der Sound-Klasse handelt es sich um eine Klasse, die einen Sound repräsentiert, er besteht in diesem Fall aus einem Audio-Clip, Volume, einem Namen, Pitch und einem Bool, der bestimmt, ob der Sound loopen soll oder nicht. Für die Zwecke unseres Projektes habe ich diese Klasse

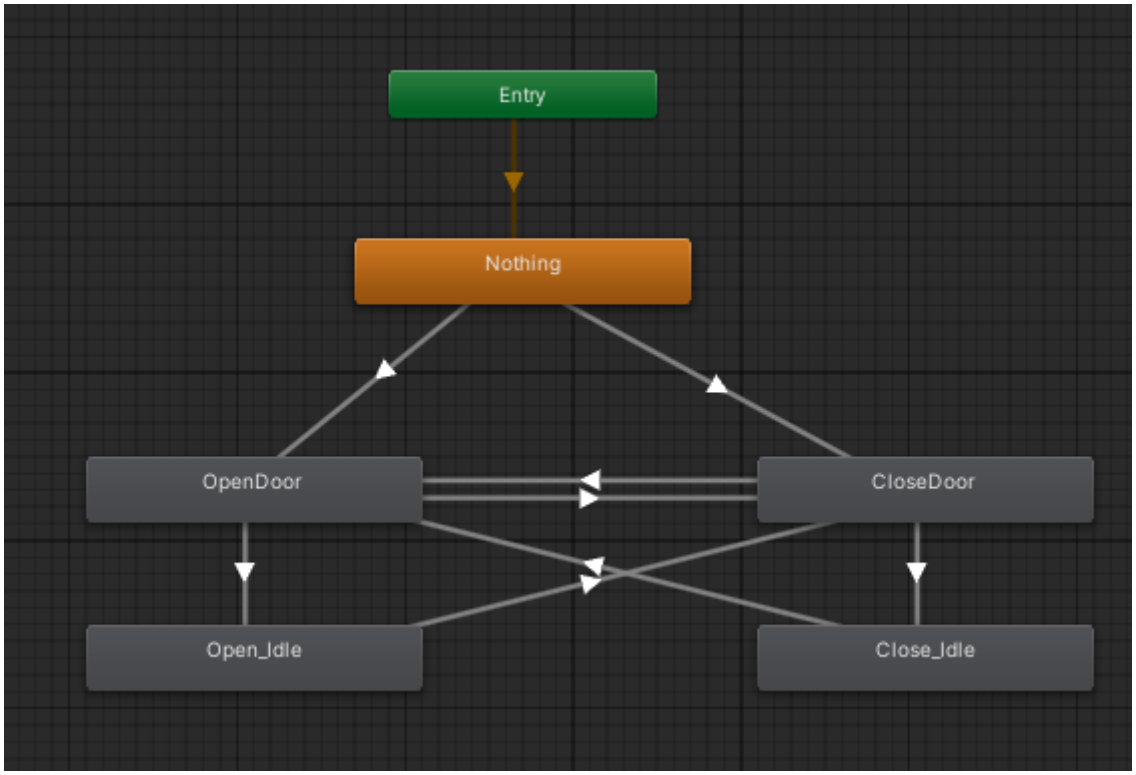
noch um einen AudioMixer erweitert, um die SFX mit einem AudioMixer gleichzeitig in ihrer Lautstärke mithilfe der Slider steuern zu können. Des Weiteren habe ich die Eigenschaft des Pitches entfernt, da wir sie nicht benötigen bzw. für uns keinen Zweck in dessen Verwendung gefunden haben. Ursprünglich wollte ich die Volume-Eigenschaft ebenfalls entfernen, da wir ja den Sound mit den AudioMixern steuern, jedoch ist der Sound für den Laser sehr laut, wodurch sich das leiser stellen mit Hilfe des Volume-Sliders als durchaus hilfreich herausgestellt hat. Im Sound-Manager wird dann das Abspielen des Sounds anhand seines Namens durch eine Public-Methode ermöglicht. So können Sounds für das Drücken eines Knopfes oder das Öffnen einer Tür ausgelöst werden, wodurch das Spielerlebnis verbessert werden kann.

Ein Sonderfall stellt der Sound des Lasers dar. Die Laser sind zum Großteil des Spiels permanent aktiv und loopen dementsprechend. In manchen Levels sind sie aber nicht zwingend immer aktiv, also müssen sie auch pausiert werden können. Ich fügte dem Skript also noch eine Methode zum Pausieren eines Sounds hinzu. Damit konnten Sounds nun gezielt pausiert, abgespielt und hinzugefügt werden.

Ein paar Probleme gab es dennoch, verteilt man nun das Objekt, das den Sound-Manager hält auf alle erstellten Level, startet jeder Sound von Beginn an, da das Skript auch neu geladen wird, was dazu führt, dass der Sound kurz “abgehackt” wird, bevor er wieder von neu beginnt. Dieses Problem habe ich mit *DontDestroyOnLoad()* gelöst. Diese Methode führt dazu, dass das Objekt mit Skript nicht gelöscht wird, wenn eine neue Szene geladen wird, wodurch der Sound weiterläuft und es uns ermöglicht, den Soundmanager einmalig im ersten Level zu platzieren. Aber auch hier kann dies zu Problemen führen, die mit dem Neustarten des Levels oder dem Sound im Hauptmenü zusammenhängen. Startet man die Szene mit dem Sound-Manager neu, wird eine neue Instanz von diesem erstellt und die vorherige Instanz wird nicht zerstört, was dazu führt, dass sich Sounds überlappen. Ich entschied mich also dazu, den Sound-Manager als Singleton zu implementieren. Damit würde nur eine Instanz vom Sound-Manager erstellt werden, wenn noch keine existiert.

Das letzte Problem bestand darin, wenn man zum Hauptmenü wechseln würde. Dadurch, dass die Instanz beim Laden nicht zerstört wird, werden Level-Sound und Menü-Sound gleichzeitig abgespielt. Hier war es nötig, dass das Objekt mit dem Sound-Manager-Skript explizit zerstört werden musste, wenn das Skript des Hauptmenüs geladen wurde.

3. Die Tür stellt das Hindernis dar, das vom Spieler überwindet werden muss, um in das nächste Level zu gelangen, also brauchte es irgendwas, um den Spieler zu signalisieren, dass die Tür geöffnet wurde und das sollte möglichst elegant passieren. Wir entschieden uns für eine kleine Animation, die die Tür nach oben schiebt, sobald der Laser den mit der Tür verbundenen Schalter berührt. Ich hatte keinerlei Erfahrung mit Animationen in Unity, weshalb hier auch eine kurze Recherche nötig war, um herauszufinden, wie ich vorgehen muss. Animationen bestehen aus Clips, die die Animationen darstellen und Animatoren, die die Animation ausführen. Für die Animation der Tür brauchte ich zunächst vier Clips: Einen, der die Tür öffnet, einen, der sie schließt und zwei Idle-Clips, die die tür geschlossen oder offen halten. Die Erstellung der Clips war simpel, man wählt das Objekt aus, das animiert werden soll, startet den Clip, wählt einen Zeitpunkt, wie lange die Animation gehen soll und verschiebt das Objekt anschließend auf einer Achse. Anschließend ging es darum, den Animations-Graphen zu bearbeiten und zu bestimmen, wann genau welche Animation abgespielt werden soll.



Damit nicht direkt eine Animation abgespielt wird, steigt der Graph mit einem Empty ein, welcher an zwei Animationen gekoppelt ist, die mit Hilfe eines *Triggers* ausgelöst werden können. Diese Trigger werden jeweils mit einem Namen ausgestattet, damit sie im Code angesprochen werden können, um die Animation gezielt auszulösen. Ich schrieb ein Skript, dessen einzige Funktion es ist, die entsprechenden Trigger zu setzen und die Tür zu öffnen oder zu schließen und hing es dem Tür-Objekt an. Allerdings funktionierte es zunächst nur für eine bestimmte Tür, da ich mich an dieser orientiert habe. Andere Türen, die in anderen Leveln an anderen Positionen positioniert waren, sprangen an die gleiche Position wie die Tür, an der ich mich orientiert hatte. Die Lösung für dieses Problem war, der Tür ein Empty als Parent-Objekt zuzuteilen, damit es an seiner Position bleibt.

In einem der Tests für den MVP wurde es geschafft, schnell hintereinander auf einen Knopf zu treten, wodurch die Tür so häufig animiert wurde, dass die Tür offen blieb, obwohl sie geschlossen sein sollte, wenn der Spieler sich nicht auf dem Knopf befand. Das war ein sehr wichtiger Fund für uns und ein Bug, der dringend entfernt werden musste. Das Problem war, dass der Wechsel zwischen den Idle-Animationen und den anderen Animationen nicht schnell genug passierte. Es lag daran, dass die Wechsel von den Idle-Animationen zu der öffnenden/schließenden Animation eine "Exit Time" hatten, die ausgestellt werden musste, damit die Animation schneller verläuft.

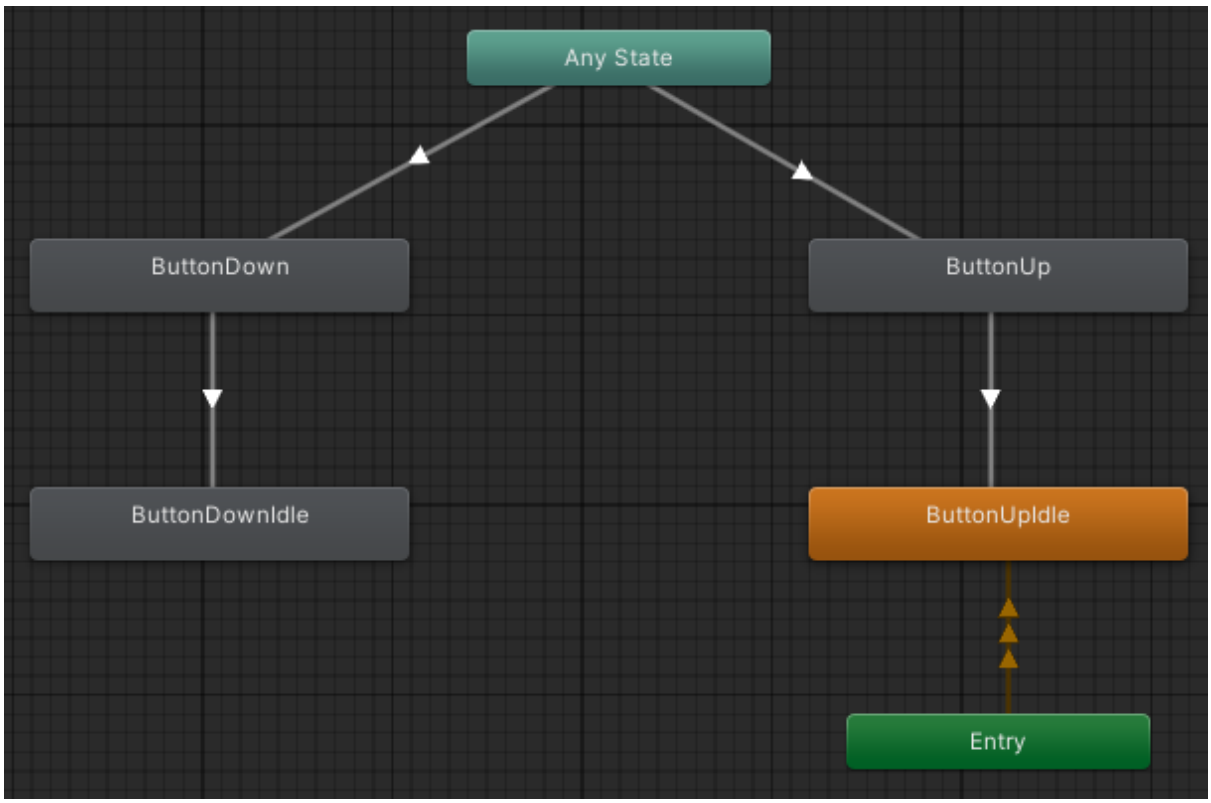
4. Siehe Kapitel "Arbeitspakete, die zusammen bearbeitet wurden" Abschnitt "Tod durch Laser von Prince und Florian"

5. Der Übergang in ein neues Level war eine Idee, die mir beim Spielen des PoC kam, nachdem wir den Wechsel in das nächste Level implementiert hatten. Dadurch, dass die nächste Szene geladen wird, kommt es zu einem kurzen Stocken des Spiels, was nicht natürlich gewirkt hat. Ich dachte an eine

kurze Animation, die den Bildschirm abdunkeln sollte, um einen nahtlosen Übergang in das nächste Level zu simulieren.

Durch die Animationen mit den Türen konnte ich nun auf grundlegendem Level mit Animationen umgehen und nahm mir ein großes schwarzes Panel, das ich mit einem Clip so animierte, dass es über einen kurzen Zeitraum über den Bildschirm fahren sollte. Das Einzige, das verblieb, war ein Skript zu schreiben, das diese Animation ausführt, sobald der Spieler das Ende des Levels erreicht. Ich erstelle ein Empty-Objekt und platzierte es in der Nähe des Ausgangs und hing ihm ein Skript an. In dessen *OnTriggerEnter()*-Methode sollte dann die Animation abgespielt werden. Allerdings war hier die Frage, wie man es timen konnte, dass mithilfe des *SceneManagers* das nächste Level geladen wird, sobald die Animation abgespielt wurde. An dieser Stelle kommen *Coroutinen* ins Spiel. Hierbei handelt es sich um Funktionen, die es erlauben, die Ausführung über einen selbst definierten Zeitraum zu stoppen und anschließend die Kontrolle wieder an Unity zu übergeben und auf dem nächsten Frame weiterzuarbeiten. Also konnte ich die Animation abspielen und nach einer kurzen Zeit das nächste Level laden. Durch die Animation wirkt der Übergang zwischen den Levels flüssiger und natürlicher, was das Erlebnis beim Spielen des Spiels erfreulicher macht.

6. Die Animation der Knöpfe sollte zum Polishing für den Prototypen von mir umgesetzt werden, damit es einerseits mit dem Spezialeffekt abgerundet wirkt und auch ein gewisses Feedback beim Drücken für den Spieler gibt. Die Animation ist nahezu identisch mit der von der Tür, deshalb dachte ich, dass dies nicht lange dauern würde. Ich legte wieder vier Clips an: Einen zum Hochfahren des Knopfes, wenn der Spieler den Knopf verlässt, einen zum Herunterfahren des Knopfes, sobald der Spieler oder eine Kiste den Knopf betritt und zwei Idle-Animationen, die den Knopf oben oder unten halten sollten. Anschließend entwarf ich den Animationsgraphen und setzte entsprechende Trigger.



Ein Skript für die Knöpfe wurde von Jonathan bereits geschrieben, da diese mit den verlegten Kabeln und den Spiegeln zusammenhängen, daher konnte ich den benötigten Code einfach dort ergänzen. Ich fügte den Animator an und spielte in den entsprechenden *OnTrigger*-Methoden die Animationen ab. Beim Testen verhielt sich jedoch nichts so, wie ich es zunächst vermutet hatte. Beim ersten Drücken des Knopfes verhielt sich alles normal, der Knopf fuhr runter und beim Verlassen wieder hoch. Alle anschließenden Interaktionen ließen den Knopf entweder einfach unten verbleiben oder erst wieder hochfahren, sobald man den Knopf erneut betrat. Ich probierte im Code mit allem herum, dass das mit dem Wort "Trigger" zu tun hatte und fand die Methode *ResetTrigger().* Mit dieser Methode wird der Trigger-Parameter zurückgesetzt und damit kann sichergestellt werden, dass dieser nicht mehr aktiv ist, sobald er verwendet wurde. Damit konnte ich das Problem lösen und wir hatten anschließend Knöpfe, die sich realistisch verhalten haben.

7. Siehe Kapitel "Arbeitspakete, die zusammen bearbeitet wurden" Abschnitt "Schmelzen der Blöcke von Prince und Florian"

8. Das Dev-Skript sollte es den Testern erleichtern, die Level einzeln zu testen und jederzeit neu starten zu können, ohne das komplette Spiel neu starten zu müssen, nur um dann wieder den gesamten Weg zum gewollten Level spielen zu müssen.

Hier gab es keine großen Probleme, das schnelle Wechseln sollte leicht für die Tester umsetzbar sein, also entschlossen wir uns dazu, dass das über die Tasten von 1-8 machbar sein sollte. Über die Klasse *SceneManager* von Unity konnten wir über mehrere if-Anweisungen unterscheiden, welche Taste gerade gedrückt wurde und dann mit dem entsprechenden Build-Index das gewollte Level laden. Das Einzige, worauf geachtet werden musste, war, dass der TimeScale des Spiels wieder auf 1 gesetzt werden musste, sollte das Spiel pausiert worden sein, mehr dazu in den Punkten "Pausemenü" und "Auswertung Playtests vom MVP".

9. Siehe Kapitel "Arbeitspakete, die zusammen bearbeitet wurden" Abschnitt "Character Movement von Prince und Florian"

10. Wie bereits im letzten Punkt erwähnt, hatte ich vor diesem Projekt bisher keinerlei Erfahrung mit Unity, meine bisherige Erfahrung mit Engines beschränkte sich auf die Unreal Engine in der Veranstaltung Game-Engines und ein kleines Projekt in der Godot-Engine. Bevor ich mit dem Projekt also anfangen konnte, musste ich mich erstmal darüber erkundigen, mit welchen Werkzeugen in der Unity-Engine gearbeitet werden konnte, wie man auf Objekte im Code über Skripte zugreifen konnte und welche Klassen der Unity-Engine uns im Laufe der Zeit nützlich sein könnten. Ich las die Dokumentation auf der Unity-Engine und probierte auch einfach viel aus, bevor ich tatsächlich damit begonnen habe, mich mit Dingen wie dem Bewegen der Spieler-Figur zu beschäftigen. All dies benötigte Zeit, aber es war eine gut investierte Zeit, da ich deshalb zum Teil schon ahnen konnte, dass Klassen wie der *SceneManager* uns im Laufe des Projekts häufiger begegnen würden und vor allem eine große Hilfe sein würden.

11. Der Spieler braucht irgendeine Möglichkeit, um innerhalb des Spiels wieder ins Hauptmenü zurückkehren zu können, bei Bedarf die Steuerung einsehen zu können oder Musik und Spezialeffekte einstellen zu können. Hierfür wurde das Pausemenü erstellt. Es hat einen sehr ähnlichen Aufbau zum

Hauptmenü, ein Fenster, von dem alles erreicht werden kann, ein weiteres zum Einsehen der Steuerung und zwei Slider zum Einstellen des Audios.

Die darin enthaltenen Slider sind an das Skript vom Sound-Manager gekoppelt, wodurch der Sound auch zuverlässig gesteuert werden kann. Es befinden sich fünf Buttons in diesem Pausemenü: einer, der das Spiel weiterlaufen lässt und das Pausemenü schließt, einer, der das Level neu startet, ein Button, der das Fenster für die Steuerung öffnet, einer, mit dem der Spieler zurück ins Hauptmenü gelangen kann und ein Button, der das Spiel komplett beendet.

Die einzige Problematik, die mit dem Pausemenü aufkam, war das damit verbundene Skript, bzw. ein kleiner, aber wichtiger Teil davon, der erst zum MVP auffiel, da ich selber nicht ausreichend getestet hatte. Im Skript des Pausemenüs wird, sobald es aufgerufen wird, mit der Klasse *Time* dessen Eigenschaft *TimeScale* von 1 auf 0 gesetzt. Dies führt dazu, dass die Szene einfriert, beziehungsweise, im Sinne des Pausemenüs, das Spiel pausiert. Würde das Level nun ohne weiteres Zutun im Code neu gestartet werden, würde die Szene zwar neu geladen werden, aber sie wäre nach wie vor eingefroren. Dieser Fehler unterlief mir zu den Tests im MVP. Es musste also bei allem, was dazu führte, dass eine neue Szene geladen würde, streng aufgepasst werden, dass bei all diesen Funktionen der *TimeScale* zurückgesetzt wird.

12. Ich war für die Auswertung der Tests vom MVP zuständig. Ich sichtete das Material und wertete es aus. Hierzu war es wichtig, herauszufinden, was den Testern gefiel,an welchen Stellen Kritik geübt wurde und was man verbessern muss. Ein Fehler, der schnell deutlich wurde, war der Fehler des Pausemenüs, der die Szene pausierte. Wechselte ein Tester vom Pausemenü in das Hauptmenü und startete das Spiel erneut, startete das Spiel nicht so wie gewollt, da der TimeScale nicht zurückgesetzt wurde. Was weiterhin kritisiert wurde, war, dass Spieler an Wänden und Kisten hängen bleiben konnten. Dieses Problem versuchte ich für mehrere Stunden auszumerzen, schaffte es aber nicht. Eine wichtige Anmerkung war das Fehlen von Spezialeffekten. Zwar hatten wir Musik, die über einen Slider einstellbar war, allerdings war die Implementation davon zu diesem Zeitpunkt noch mit Fehlern behaftet, wodurch zwar die Musik im Hauptmenü auf neue Einstellungen reagierte, die im Level jedoch nicht, außerdem waren die Slider im Level und Hauptmenü nicht synchron. Für Spezialeffekte

fehlte leider die Zeit, da wir leider auch sehr spät erst bereit zum Testen waren und ich dafür mich noch nicht genug mit Sounds in Unity beschäftigt hatte.

13. Siehe Kapitel “Spielbeschreibung” Abschnitt “Levelbeschreibung” Punkt “Level 02 – Simple Parkour”

14. Über die Laufzeit des Projektes haben Jonathan, Prince und ich uns in regelmäßigen Abständen darüber unterhalten, wie der Stand des Projektes zu diesem Zeitpunkt war. Wir besprachen Probleme, besprachen Lösungen und diskutierten über Spielinhalte, die eventuell einen Mehrwert für unser Projekt bieten würden. So besprachen wir unter anderem, ob oder wie wir den schießenden Roboter in unsere Level einbinden könnten. Am Ende kamen wir zu dem Schluss, diesen zu streichen, da uns kein sinnvolles Szenario einfiel, in dem er sich positiv in Leveln einfügen würde.

15. Das Schreiben der Dokumentation ist äußerst wichtig. Es führt den Entwicklern sowohl Erfolge als auch Fehlschläge vor Augen und lässt sie aus ihren Fehlern lernen. Des Weiteren werden Personen, die die Dokumentation lesen, mit den Gedankengängen der Entwickler vertraut und können diese nachvollziehen. Das gesamte Team nahm sich daher viel Zeit, um diese möglichst sorgfältig zu gestalten.

Ich übernahm die Dokumentation des Codes unseres Projektes. Da wir eine recht hohe Anzahl an Klassen erstellt haben und auch nicht jede von mir geschrieben war, musste ich natürlich zunächst auch diesen Code verstehen, bevor ich ihn kommentieren konnte, was Zeit in Anspruch nahm. Zwar kommentierte ich das Projekt von Anfang bis Ende, dennoch war es den Autoren der Klassen stets freigestellt, diese zu ändern, sollten sie etwas zu ergänzen haben.

Themen Jonathan El Jusup

KURZÜBERSICHT

Zeit Gesamt: 186

1.	Asset Modellierung	11	Stunden
2.	Character Controller	20	Stunden
3.	Laser Implementierung	26	Stunden
4.	Spiegel Implementierung	7	Stunden
5.	Überwachungskamera	2	Stunden
6.	Animation & Partikeleffekte	4	Stunden
7.	Trigger (Knopf & Laser-Schalter)	4	Stunden
8.	Level Design	51	Stunden
9.	Kamera & Post Processing	3	Stunden
10.	Bewegliche Plattformen	2	Stunden
11.	Menü Redesign	9	Stunden
12.	Projektorganisation	15	Stunden
13.	Dokumentation	20	Stunden
14.	Sonstiges	12	Stunden

ERKLÄRUNG

(1) Asset Modellierung

Ich habe es mir zur Aufgabe gemacht, jegliche Assets für das Projekt zu modellieren. Dies galt ursprünglich auch für den Roboter Charakter. Durch Umstrukturierung aufgrund eines fehlenden Gruppenmitglieds konnte ich mir diesen Luxus leider nicht mehr leisten. Ausgenommen des Spieler Charakters wurde jedoch alles selbst in Blender modelliert. Diese modellierten Assets wollte ich vom Stil her möglichst konsistent halten und unserer Vision aus der Spieleidee treu bleiben. Angefangen beim Farbschema habe ich mich für monochrome Farben für die Umgebung, der Hintergrund-Panels und der Konstruktionsblöcke entschieden. Um dies aufzubrechen sollten Farben ins Spiel gebracht werden u.a. durch die grünen und roten Laser, aber auch durch farbige Kisten und Gläser. Dies soll eine gewisse Relevanz für den Spieler andeuten, damit dieser merkt, dass sie in irgendeiner Form wichtig für den Spielverlauf und für das Lösen von Rätseln sind.

Die Ästhetik der Modelle ist minimalistisch und sauber. Ich habe für den Gesamteindruck bewusst auf komplexe Formen und Mesh-Topologien verzichtet. Es gibt wenige Kurven und viele rechte Winkel.

Dies spiegelt sich ebenfalls im Level Design wider. Zusammen mit dem Farbschema soll eine futuristische und saubere Atmosphäre geschaffen werden, eine Umgebung, wie ein Labor oder eine Einrichtung, in der alles unter perfekter Kontrolle und Ordnung ist.

(2) Character Controller

Der Character Controller wurde zum Proof of Concept von Florian und Prince bereits rudimentär implementiert, doch stellte sich früh heraus, dass bestimmte Funktionen wie das Springen des Spielers und die Reibung an Objekten noch optimiert werden konnten. Ich habe u.a. an diesen einzelnen Komponenten gearbeitet, um die Spielerbewegung zu verbessern.

Angefangen mit dem Sprung des Spielers gab es früh Probleme mit der Erkennung des Bodens, worin der Spieler auf dem Boden stehen muss, um springen zu dürfen. Dies habe ich mit einem zweiten kreisförmigen Collider erzielt, welcher die Trigger Eigenschaft besaß. Solange ein Objekt diesen Collider geschnitten hatte, wurde der Spieler als auf-dem-Boden betrachtet und konnte springen. Zum Ende hin wurde dieser Collider von Prince durch einen vertikalen Raycast nach unten ausgetauscht. Diesen Code habe ich vereinfacht, optimiert und angepasst, um mit Steigungen von bis zu 45° umgehen zu können.

Der Sprung des Spielers selbst funktionierte soweit, fühlte sich jedoch unnatürlich an und es fehlte an Kontrolle. Diesbezüglich habe ich mich weiter informiert und stellte fest, dass der Sprung zwar physikalisch korrekt funktioniert, uns Spielern jedoch unnatürlich vorkommt, da wir an Videospiele gewöhnt sind. Am Beispiel „Super Mario“ kann man sehen, dass der Sprung nicht physikalisch korrekt ist, denn der Spieler fällt nach dem Apex des höchsten Punktes schneller nach unten. Oder anders: Die Gravitation wird beim Fall erhöht. Dies wollte ich ebenfalls so implementieren, bin jedoch noch einen Schritt weiter gegangen, indem der Spieler nun über die Sprunghöhe selbst entscheiden kann, abhängig davon, wie lange dieser die Sprungtaste gedrückt hält. Die Gravitation wird somit basierend auf der Länge des Tastendrucks geringer gewertet und wirkt dennoch im Fall viel stärker als am Anfang. Damit wollte ich erzielen, dass der Spieler mehr Kontrolle über seinen Sprung erhalten soll, insbesondere an Stellen im Spiel, an denen präzise Sprünge erforderlich sind, darunter auch sehr lange oder ganz kurze Sprünge.

Während der Playtesting Phase des Prototyps ergab sich. Dass die Spieler diese Funktionalität weder positiv noch negativ empfunden haben und den Sprung als „floaty“ beschrieben. Doch hat es den Spielern nicht wirklich gestört. Ich persönlich schließe zusammen mit meiner vorherigen Recherche zum Thema, dass die Kontrolle der Sprunghöhe eher unterbewusst wahrgenommen wird und demnach auch so genutzt wird, wenn die richtige Situation kommt. Deswegen habe ich mich im Level Design speziell darauf fokussiert, verschiedene Sprung-Distanzen vom Spieler zu fordern. Durch diesen variablen Sprung wollte ich dem Spieler einen gewissen Mehrwert durch Kontrolle seiner Bewegung bieten.

Der zweite große Punkt des Character Controllers war die Reibung an Objekten. Es hat sich herausgestellt, dass die Einstellung des Reibungswertes des Spielers schwieriger als Gedacht war. Denn der Spieler sollte einerseits eine gewisse Reibung am Boden besitzen, jedoch keine an Wänden. Er sollte Reibung an beweglichen Plattformen erleben, damit dieser nicht runterrutscht, aber wenig

bis keine an Kisten, wenn er diese verschiebt. Dieses Problem, welches sich leider erst später als trivial entpuppte, bereitete uns bis zum MVP-Meilenstein große Probleme, denn es beeinflusste den Spielfluss trächtig. Ein Lösungsansatz war, dem Spieler verschiedene Physik-Materialien bzw. verschiedene Reibungswerte zu geben, welches nur durch separate Collider lösbar war. Dafür habe ich auf die Form des Spielers geschaut und darauf geschlossen, dass die Seiten des Spielers nur mit Wänden und Kisten in Berührung kommen können; Und gerade an den Seiten sollte es keine Reibung geben. Aus diesem Grund habe ich mich für einen weiteren Collider entschieden, welcher etwas breiter, aber in der Höhe kleiner ist neben dem ursprünglichen Kapsel-Collider, sodass die Seiten des Spielers dadurch abgedeckt werden. Diesem Collider habe ich einen Reibungswert von 0 gegeben und dies funktionierte auch soweit für die Seiten, während der ursprüngliche Collider für den Boden verantwortlich war, der Reibung erfahren sollte.

Dies sorgte jedoch für weitere Probleme. Zum einen sorgte es für Unstetigkeiten am Schnittpunkt des Seiten-Colliders mit dem Basis-Collider. Dies führt dazu, dass der Spieler seitlich an den Füßen am Kanten der Umgebung stecken bleibt. Dies konnte ich kompensieren, indem ich die Breite des Seiten-Colliders schmaler machte, sodass diese Unstetigkeiten weniger auffielen. Dies war jedoch auch nicht perfekt und sorgte wieder für Probleme mit der Reibung, da die Collider so nah beieinander waren, dass die Kollisionserkennung diese schlechter auseinanderhalten konnte. Ein weiteres Problem zweier Collider war die Interaktion mit Kisten, wenn der Spieler diese verschieben wollte. Zwar lief alles gut an den Seiten. Sobald der Spieler aber auf den Kisten stand, war die Reibung so hoch, dass allein die Bewegung des Spielers auf den Kisten eine Rotationskraft auf diese wirkte. Wieder einmal gab es einen guten Reibungswert für den Boden; der war aber zu hoch für Kisten. Noch einen weiteren Collider allein dafür zu erstellen, wäre naiv und zum Scheitern verurteilt.

Nach einiger Zeit und bei weiterer Recherche über das Reibungsmaterial ergab sich jedoch, dass man verschiedene Reibungen auf bestimmte Weise miteinander verrechnen konnte. Standardmäßig geschah dies durch Bilden eines Durchschnitts, doch gab es auch die Option der Multiplikation, nicht nur bei dynamischen, sondern auch statischen Reibungen. Dieses, doch triviale Problem konnte somit gelöst werden und ich konnte auf weitere unnötige Collider am Spieler verzichten, auch wenn es einige Zeit in Anspruch genommen hatte. Durch weitere Feineinstellungen konnte ich das Reibungsproblem für alle Kollisionsfälle beheben.

Es gab weitere, kleinere Komponenten des Character Controller, an denen ich arbeitete, dass die Geschwindigkeit bei Sprüngen zurückgesetzt wird, damit der Spieler diese nicht akkumulieren kann. Außerdem habe ich mich darum gekümmert, dass der Roboter beim Tod des Spielers in seine Einzelteile zerspringt. Dies waren jedoch die signifikantesten Teile, an denen ich gearbeitet habe.

(3) Laser Implementierung

Die Laser Implementierung ist einer der Kernkomponenten des Projektes. Dementsprechend muss diese Mechanik auch gut funktionieren, da die ganze Vision davon abhängt. Das Prinzip, Strahlen zu verschießen, welche an bestimmten Objekten reflektiert werden oder diese zu durchdringen, ist nicht allzu schwer zu implementieren. Hierfür habe ich Raycasting verwendet, welches den Strahl berechnet, bevor dieser gezeichnet werden kann. Anhand des Ursprungs und des potenziellen Kollisionspunktes kann ein Line Renderer verwendet werden, welcher eine Linie zwischen beiden

Punkten zeichnet. Die Eigenschaften und Parameter können individuell im Code eingestellt werden, von der Farbe, der Liniendicke, bis hin zum Material des Lasers. Dies war ein guter Anfang, ich musste aber schnell feststellen, dass die Reflektion und Transmission nicht so einfach waren, wie ich dachte. Die isoliert betrachtete Implementierung davon mag nicht so schwer gewesen sein. Es musste einfach nur rekursiv ein weiterer Strahl verschossen und geschaut werden, wo dieser als nächstes aufprallt. Bei Reflektionen wird einfach der Richtungsvektor des vorherigen Strahls reflektiert und bei Transmissionen wird die Richtung beibehalten. Ein kleines Problem tauchte jedoch auf, was weiter folgende Laserstrahlen anging. Wenn ein Folge-Laser an der letzten Kollisionsposition, also dem Endpunkt des vorherigen Strahls anfängt, wird sofort eine Kollision wahrgenommen. Dies kann durch einen einfachen Offset entlang des neuen Richtungsvektors kompensiert werden. Dieser Offset musste aber bei steileren Winkeln stärker sein. Diesen Offset zusätzlich vom Winkel zur Oberflächennormalen abhängig zu machen, wäre möglich gewesen. Doch je steiler der Winkel ist, desto größer müsste der Offset sein. Demnach würde der gezeichnete Strahl sichtbar unterbrochen werden. Deshalb habe ich mich für die einfachere und visuell ansprechende Variante entschieden, einen weiteren Offset entlang der Oberflächennormale des Objektes zu nutzen. Dies hat das Problem gelöst und die Laserstrahlen werden visuell kontinuierlich gezeichnet.

Das nächste Problem war die Änderung des Lasertypen, wenn es transparente Objekte wie Glas oder transparente Kisten durchdringt, welche eine andere Farbe als der Laser haben. Da es sich beim Zeichnen des Lasers um einen Line Renderer handelt, kann ich schlecht die Eigenschaften pro Liniensegment festlegen. Die Farbe kann zwar als Gradient festgelegt werden. Dies wäre auch im Code möglich. Es wäre jedoch sehr umständlich, einen Laser an bestimmten Stellen umzufärben. Auch gäbe es ein Problem mit dem Tagging System in Unity, da der Laserstrahl als ein Objekt betrachtet wird und nur einen Tag besitzen kann. Doch brauche ich mindestens zwei Tags, um zu unterscheiden, ob ein Laser sicher oder tödlich ist. Nach einigem Überlegen habe ich mich dazu entschieden, die Laserstrahlen in einzelne Objekte zu segmentieren, die ihren eigenen Line Renderer und Tag besitzen. Diese Laserstrahlen werden in einer Hierarchie eingegliedert, bei dem alles von einem initialen Laserstrahl ausgeht und sich weiter nach unten kaskadiert. So konnte ich sicherstellen, dass jedes Laser-Segment eigene Eigenschaften besitzt.

Bis hierhin war ich in der Lage einen Laser zu verschießen, welche an Objekten reflektiert oder durchgelassen werden kann. Es entsteht eine kontinuierliche Linie, bestehend aus einzelnen untergeordneten Segmenten mit eigenen Eigenschaften. Ab hier trat aber das größte Problem auf, das Aktualisieren von Laserstrahlen, welches u.a. Richtungsänderungen, Blockierungen und (De-)Aktivieren von Laserstrahlen beinhaltet. Sobald ein reflektierender Spiegel seine Rotation ändert, soll sich auch der reflektierte Laser mit allen seinen folgenden Segmenten aktualisieren. Das Problem ist jedoch, dass diese bereits initialisiert wurden und diese neu verschossen werden müssen. Dies könnte so implementiert werden, dass nur betroffene Segmente neu berechnet werden. Doch benötigt dies ein weiteres System, welches solche Ereignisse wie sich ändernde Spiegel-Rotationen, Laser Blockierungen, Ein- und Ausschalten von Laserquellen berücksichtigt und betroffene Segmente ermittelt und diese über Änderungen benachrichtigt. Dies wäre möglich und eventuell auch der performantere Weg gewesen, doch wäre die Komplexität um ein Vielfaches angehoben worden. Aus diesem Grund entschied ich mich für die Variante, jedes Laser-Segment, jedes Update, also jeden Frame zu zerstören und neu zu verschießen. Dies geschieht so schnell, dass der Spieler den Laser nicht

verschwinden sieht, sondern Laserstrahlen kontinuierlich zu sehen sind, wie eine Lampe in Realität, welche mit einer hohen Frequenz mehr oder weniger „blinkt“. Der Vorteil hierbei ist, dass sich die Laserstrahlen und alle Segmente an jegliche Änderungen sofort anpassen. Da die Laserstrahlen in der Hierarchie geordnet und verschachtelt sind, wird diese auch nicht weiter verschmutzt. Ein weiterer Vorteil der Verschachtelung der Laser Segmente ist der, dass nur das oberste Objekt zerstört werden muss, und alle untergeordneten Laser Segmente automatisch zerstört werden.

Zusammen mit der Aktualisierung der Laserstrahlen jedes Frame, entstehen dynamische, anpassbare und kontinuierliche Laserstrahlen. Die Interaktion der Laserstrahlen mit Objekten in der Szene wird über ein einfaches Tagging-System gelöst, bei dem Tags von Objekten und Laser Segmenten abgefragt werden, um zu entscheiden, ob ein Laser beispielsweise ein Objekt durchdringt, zerstört, schmelzt oder daran reflektiert wird. Bis hierhin wurde eine solide Basis entwickelt, die noch weiter ausgebaut werden konnte. Ich musste aber viel Zeit investieren, um an diesen Punkt zu gelangen.

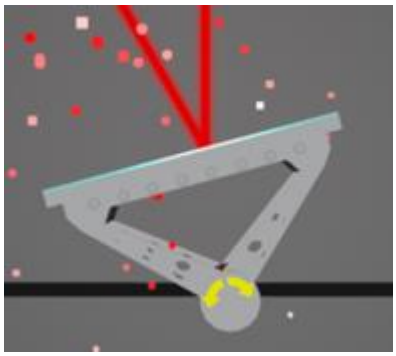
Nun war es Zeit für weitere Optimierungen. Mir vielen 2 Optimierungsmöglichkeiten ein, welche jedoch ein tieferes Eingreifen in die Laser Implementierung erforderten. Zum einen das Benutzen von sog. „Object Pools“. Hierbei handelt es sich um eine Vorab-Initialisierung von Laser-Segmenten in einem „Pool“, welche zum Spielstart deaktiviert werden. Zur Laufzeit wird zu jedem Update ein verfügbares Laser-Segment aus dem Pool angefordert, geliefert, konfiguriert und wieder aktiviert, bis es zum nächsten Update wieder deaktiviert und zurück in den Pool verschoben wird. Von dieser Optimierung habe ich mir viel versprochen, da ich mich mit dieser Technik schon einigermaßen auskannte und diese in Praxis oft benutzt wird, musste jedoch feststellen, dass diese Optimierung die Performance des Spiels nicht weiter verbessert und in einigen Fällen sogar verschlechtert hatte. Außerdem musste die Poolgröße aufgrund von Inkonsistenzen deutlich Größer eingestellt werden als eigentlich nötig wäre. Aus diesem Grund habe ich diese Optimierung wieder verworfen.

Eine weitere Möglichkeit, die Performance zu optimieren, wäre die Nutzung von Laser Prefabs, welches bereits mit allen nötigen Komponenten und Skripten versehen wird, welches nur noch instanziiert werden muss. Nach einigem Testen habe ich aber festgestellt, dass die Performance trotz vieler Laserstrahlen relativ respektabel war und die Framerate dennoch stabil weit über 60 FPS bis hin zu 90 FPS lief. Ich vermute, dass die Garbage Collection und der Cache relativ gut für solche Fälle optimiert sind. Aus diesem Grund habe ich entschieden, auf weitere Optimierungen des Lasers zu verzichten, da die Performance sich in der Praxis vollkommen in Ordnung verhält. In keiner Weise möchte ich damit aber andeuten, dass meine Implementierung die Optimalste wäre. Für unsere Zwecke reicht diese Implementierung aber komplett aus. Nichtsdestotrotz war die Implementierung des Lasers eine der Zeit-intensiveren Arbeitspakete.

Zuletzt wollte ich mich noch um die Dekoration des Lasers kümmern. Hierzu haben sich Partikeleffekte und Punktlichter sehr gut angeboten. Dafür habe ich den Endpunkt des Lasers am letzten Segment berechnet und dort Funken sprühen lassen. Außerdem wird an der Stelle eine Punktlichtquelle in der passenden Farbe gesetzt. Da diese Art von Lichtquelle in Echtzeit funktioniert, ergeben sich schöne Licht Interaktionen mit der Umgebung und den Hintergrund Panels. Zusammen stellt dies einen visuellen Mehrwert für den Spieler dar, sodass dieser den Laser als dynamisches Objekt wahrnimmt.

(4) Spiegel Implementierung

Die Implementierung der Spiegel war im Vergleich zu der Implementierung der Laser weitaus entspannter, da ich mich auf weniger Aspekte außer Rotation und Translation der Spiegel Komponenten fokussieren musste. Dennoch hatte die Implementierung seine eigenen kleinen Tücken. Um ein Objekt zunächst reflektierend zu machen, muss nur der richtige Tag gesetzt werden und Laserstrahlen berücksichtigen dies bei einer Kollision mit solchen Objekten. Demnach konnte ich relativ einfach eine reflektierende Fläche an das selbst modellierte Spiegel Modell anbringen und mit einem reflektierenden Tag versehen. Die Reflektion funktioniert dabei automatisch.



Der Spiegel soll aber vom Spieler gesteuert und rotiert werden können. Der Spiegel soll in 2 Richtungen gedreht werden können: Einmal gegen den Uhrzeigersinn bzw. um positive Winkel und einmal im Uhrzeigersinn, also negative Winkel, so wie es in Unity funktioniert. Der Spiegel wird mit 2 optionalen Trigger Objekten versehen, welche im Editor individuell versehen werden können. Als Trigger zählen beispielsweise Druckplattformen bzw. Knöpfe oder Laser Schalter. Jeweils ein Trigger ist für eine Rotationsrichtung zuständig. Es kann kein Trigger, ein Trigger oder beide Trigger zugewiesen werden. Demnach kann der Spiegel auch als statisch betrachtet werden, wenn er nicht rotiert werden soll. Im Code kann individuell eingestellt werden, wie schnell sich der Spiegel drehen soll und was die unteren und oberen Rotationsgrenzen sind. Durch die Einschränkung der Rotation des Spiegels soll dem Spieler nicht unnötig viel Freiheit gegeben werden, Spiegel so zu drehen, wo es keinen Sinn macht. Bis hierhin steht eine gute Basis eines Spiegels.

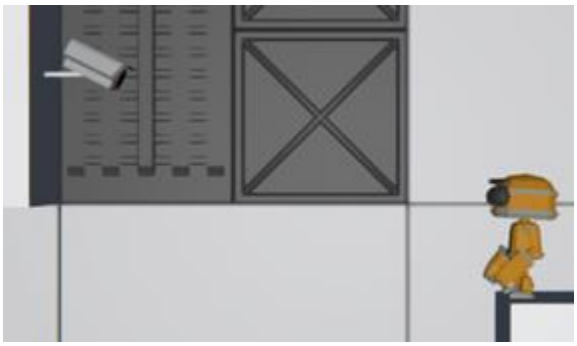
Zusätzlich habe ich den Fall behandelt, wenn beide Trigger gleichzeitig aktiviert sind. Demnach ist der Spiegel in seiner Rotation blockiert, da er gleichzeitig nach links und rechts gedreht werden soll. Diese Blockierung wird durch Animation des Spiegels und Partikeleffekte visualisiert.

Doch ergibt sich aufgrund des Modells des Spiegels ein Problem in der Rotation, da der Drehpunkt des Spiegels weiter unten liegt als die eigentliche Spiegelfläche. Dies hat zur Folge, dass sich der Kollisionspunkt eines Laserstrahls auf der Spiegeloberfläche bei Rotationen des Spiegels ändert. So liegt der Kollisionspunkt mit der Spiegeloberfläche exakt in der Mitte, wenn der Strahl exakt gegen die Oberflächennormale schießt, aber weiter links, wenn sich der Spiegel im Uhrzeigersinn dreht und analog dazu weiter rechts bei Drehungen gegen den Uhrzeigersinn. Dies schränkt die Fläche ein, wo der Laserstrahl reflektiert werden kann und macht insbesondere bei 45° Reflektionen Probleme, da der Laser genau auf die Spiegelkante trifft. Um dies auszugleichen, habe ich ein leeres Objekt über dem eigentlichen Spiegel eingeführt, welches als Translations-Mittelpunkt fungiert und den Spiegel horizontal oder vertikal verschiebt, wenn dieser gedreht wird. Somit soll sichergestellt werden, dass der Kollisionspunkt des Laserstrahls mit der Spiegeloberfläche bei jeder Drehung identisch bleibt. Hierfür habe ich im Skript die Einstellung angeboten, den Spiegel entweder vertikal oder horizontal zu verschieben mit einem unteren und oberen Grenzwert für die Verschiebung. Die Grenzen werden

zur Laufzeit in globale Positionen umgerechnet und passen sich demnach an. Die Verschiebung des Spiegels wird basierend auf seiner aktuellen Drehung zwischen beiden Rotationsgrenzen interpoliert.

Mit allen Komponenten zusammen entsteht ein hoch benutzerdefinierter Spiegel, welcher feingranular eingestellt werden kann. Dies geht jedoch auch auf Kosten der Komplexität, diesen Spiegel im Level Design zu nutzen. Es erfordert dementsprechend ein gewisses Grundverständnis der Implementierung des Spiegels, um diesen passend einstellen zu können.

(5) Überwachungskamera



Überwachungskameras galten initial als narratives Element, um den Spieler anzudeuten, dass er zu jeder Zeit von der Entität dieser Einrichtung beobachtet wird. Nach initialen Playtests hat sich auch bestätigt, dass es den Spielern gefiel. Auch wenn die Überwachungskameras keinen funktionalen Zweck erfüllen, sind sie dennoch ein schönes dekoratives Element, erzeugen eine gewisse Atmosphäre und tragen zum "World Building" bei. Auch weitere Elemente wie der sog. „Scrap Shoot", die „Prop Execution" oder aufleuchtende Kabel, welche ich entwickelt habe, sorgen für mehr Abwechslung und sind ein wichtiger Bestandteil für das Level Design und die Narrative.

Die Implementierung der Überwachungskamera ist recht trivial. Die Überwachungskamera besteht aus einem selbst modellierten Körper, welcher sich drehen kann und einem statischen Stab, der den Körper mit der Wand verbindet. Die Überwachungskamera führt jedes Update einen Raycast zum Spieler aus und schaut, ob der Strahl den Spieler getroffen hat oder dieser zuvor von einem Objekt blockiert wurde. Solange direkter Sichtkontakt zum Spieler besteht, dreht sich der Kamera Körper stets zum Spieler und schaut diesen an.

(6) Animation & Partikeleffekte

Ursprünglich war ich nicht für die Animationen der Spielelemente zuständig. Dies wurde von meinen Gruppenmitgliedern umgesetzt, doch fand ich immer wieder passende Stellen, wo ich schnell eigene Animationen gestalten konnte. Beispiele hierfür wären die Kreissägen- Animation oder das Ruckeln des Spiegels, wenn dieser blockiert wurde. Langsam gewann ich mehr Erfahrung in diesem Gebiet und schaute mir bereits umgesetzte Animationen von meinen Gruppenmitgliedern an. Es gab ein paar, welche unter bestimmten Umständen nicht wie gewollt funktionierten oder optimiert werden konnten. Ein Beispiel hierfür wäre die Tür, dessen Animation zum Öffnen und Schließen ich angepasst habe, indem ich den Übergang zwischen beiden Stati über eine benutzerdefinierte Kurve geregelt habe. Damit konnte ich eine schönere Animation erzielen, bei der die Tür beim Öffnen kurzzeitig zu weit geht und dann wieder zurückspringt oder beim Schließen hart auf den Boden aufprallt und kurz wieder aufspringt. Ich konnte hier und da kleinere Animationen reparieren, indem ich bestimmte Transitionen neu eingeführt habe. Zuvor gab es nämlich das Problem, dass die Tür von ihrer Animation beim Schließen diese erst abschließen musste, um in ihren Idle-Close-State zu kommen.

Erst danach konnte sie wieder geöffnet werden. Dies lief auch im Fall des Öffnens der Tür so. Dies sorgte für das Problem, dass beim rapiden Öffnen und Schließen der Tür die Animation und der interne Status der Tür vertauscht wurden und die Tür zu war, wenn sie offen sein sollte und offen war, wenn sie zu sein sollte. Weitere Animationen, an denen ich beteiligt war, war die der Knöpfe, welche Probleme mit ihrer Exit Transition-Zeit hatten.

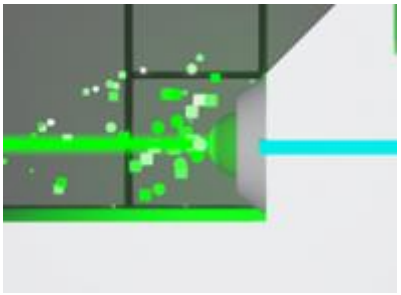
Neben den Animationen habe ich viel mit Partikel-Effekten experimentiert. Hierfür habe ich am Ende eines Laserstrahls Partikeleffekte in Form von Funken eingeführt, welche die gleiche Farbe haben, wie der Laser. Dies führt mehr visuellen Zucker in die Szene ein und lässt den Laser und die gesamte Szene lebendiger wirken. Ursprünglich hatte ich das Problem, mich zu entscheiden, ob ich am Ende jedes Laser Segmentes Partikeleffekte anbringen sollte. Dies war insbesondere problematisch, da sich die Laser Segmente jedes Update zerstören und neu generieren. Nach einigem Überlegen fiel mir jedoch auf, dass Reflektionen und Transmissionen keine Funken erzeugen und Funken nur bei Termination des Laserstrahls, sprich am Ende eines Strahls, auftreten. Dies vereinfachte alles sehr. Demnach brauchte ich nur ein Partikeleffekt pro Laserquelle, was viel performanter wäre, als mehrere. Es blieb nur noch das Problem, dass sich Laser Segmente rapide selbst zerstören. Ich habe das Problem so gelöst, dass die Partikeleffekte Teil eines Objektes sind, dessen Position explizit vom letzten Laser Segment aktualisiert wird.

Des Weiteren habe ich mich um Beleuchtungen gekümmert, wie das Platzieren einer Punktlichtquelle am Ende eines Laserstrahls oder an einem Laser-Schalter, wenn dieser von einem Laser aktiviert wurde, um dem Spieler extra zu zeigen, dass der Schalter aktiv ist. Dies waren die größten Gebiete zum Thema Animationen und Partikel-Effekten, an denen ich gearbeitet habe, die nennenswert waren.

(7) Trigger (Knopf & Laser-Schalter)

Unter Trigger zählen Knöpfe bzw. Druckplattformen und Laser-Schalter. Diese teilen sich die gleichen Komponenten, erben deswegen von der Trigger Oberklasse. Trigger können mit Effektoren wie Laserquellen, Spiegeln oder Türen verbunden werden, um diese zu steuern. Trigger spielen eine wichtige Rolle, denn sie dienen als Auslöser und Manipulator von Mechanismen, die der Spieler nutzen kann. Die Implementierung dieser Trigger war relativ trivial und bedarf keiner allzu genaueren Erklärung als diese, welche folgt.

Ein Knopf ist eine einfache Druckplatte, welche vom Spieler oder Kisten gedrückt werden kann. Im gedrückten Zustand ist der Trigger aktiv und beeinflusst den zugewiesenen Effektor. Knöpfe werden oft benutzt, um Spiegel zu drehen. So kann der Spieler auf einen Knopf treten, um den Spiegel in die eine Richtung zu drehen oder auf einen anderen Knopf, um den Spiegel in die andere Richtung zu drehen.

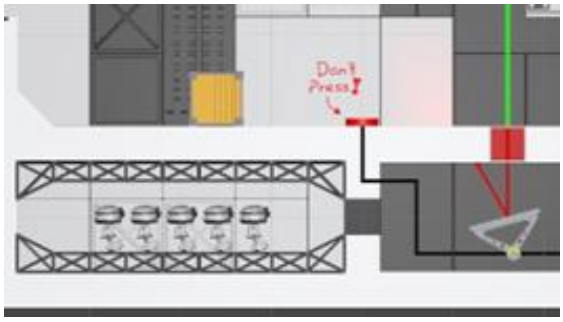


Laser-Schalter hingegen können nur von Laserstrahlen aktiviert werden. Diese sind jeweils mit einem Laser Typen kompatibel, was man an seiner Farbe sehen kann. Ist der Laser-Schalter rot, wird dieser nur von roten Laserstrahlen aktiviert. Analog dazu gibt es auch einen grünen Laser-Schalter. Der Laser-Schalter besteht aus mehreren Komponenten, wobei eine Kugel benutzt wird, die vom Laser bestrahlt werden muss, damit der Schalter sich aktiviert. Ein Laserstrahl, der woanders am Körper des Schalters strahlt, aktiviert diesen nicht. Um zu zeigen, dass der Schalter aktiviert wurde, leuchtet dieser auf in der entsprechenden Farbe. Ein Laser-Schalter unterscheidet sich außerdem vom Knopf insofern, dass dieser für eine bestimmte Zeit aktiv bleibt, auch wenn der Laserstrahl nicht mehr auf diesen strahlt. Hierfür wird ein Timer benutzt, der sicherstellt, dass der Schalter eine Weile länger aktiv bleibt und nicht sofort auf Änderungen reagiert. Dies ist besonders wichtig, wenn der Spieler den Schalter rapide aktiviert und deaktiviert.

(8) Level Design

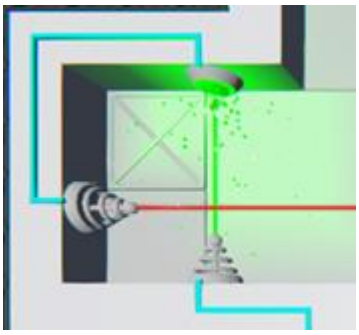
Im Vergleich zu allen anderen Arbeitspaketen habe ich mit Abstand die meiste Zeit im Level Design verbracht. Auch wenn ich darauf bestand, dass jeder für eine Hand voll Level verantwortlich sein sollte, habe ich die meisten Level entwickelt. In diesem Sinne habe ich folgende Level von Anfang bis Ende entwickelt (Level 00, Level 01, Level 03, Level 04, Level 05). Im Fokus stand insbesondere das versteckte Tutorial, Mechaniken nach und nach einzuführen und dem Spieler möglichst indirekt und durch gutes Level Design das Spiel beizubringen. Diese Levels sollten möglichst aufeinander aufbauen und eine gewisse Schwierigkeitskurve haben. Es sollten keine Spielelemente in Levels vorkommen, welche mechanisch noch nicht eingeführt wurden. In diesem Sinne habe ich mir besonders Mühe gegeben, Mechaniken aufbauend aufeinander nach und nach einzuführen, welche vom Spieler zunächst einfach erlernt werden können und im weiteren Spielverlauf gemischt kombiniert werden müssen. Alles soll zusammen konsistent wirken und durch gutes Level Design und Dekorationen den Spieler ansprechen. Außerdem war uns Affordanz von Anfang an wichtig. Dies muss sich insbesondere im Level Design wiederfinden. Der Spieler muss zu jeder Zeit Bescheid wissen, mit welchen Spielelementen er interagieren kann, was er gerade tut und welchen Status bestimmte Mechaniken gerade haben. Außerdem sollte das “World Building” im Level Design entsprechend gut vertreten sein, um die Narrative und die Welt darzustellen, auch wenn die Narrative nicht im Vordergrund steht. All diese Aspekte müssen im Level Design beachtet werden. Dementsprechend hat das Level Design viel Zeit in Anspruch genommen. Darunter gilt auch das Playtesting und Balancing der Rätsel und Levels.Da alle Level bereits vorher im Detail beschrieben worden sind, möchte ich hier auf spezielle Stellen eingehen, welche für mich besonders wichtig waren und das Level Design ausmachten. Angefangen mit dem ersten Level habe ich für die Narrative den Spieler durch einen Schacht fallen lassen, welcher mit lauter grünen Lasern gefüllt ist. Der Spieler landet dann direkt neben einem sog. „Scrap Shoot“, in welchem Roboterteile

herabfallen und von Kreissägen geschreddert werden. Dies ist unmittelbar mit der Narrativen verbunden, die schon im Hauptmenü angedeutet wurde, wo auch Roboterteile herab fielen. Dem Spieler soll klar werden, dass es ihm auch so ergangen wäre, er aber auf wundersamer Art und Weise überlebt hat. Sofort wird er von einer Kamera beobachtet und sein Abenteuer fängt an. Dieser „Scrap Shoot“ erfüllt aber noch eine zweite Rolle, und zwar die des versteckten Tutorials, indem dort indirekt durch herabfallende Tasten die Steuerung des Spiels erklärt wird, ohne den Spielfluss durch Text zu brechen.



Im nächsten Level angelangt findet er direkt seinesgleichen und der Spieler wird aus Neugier den großen roten Knopf drücken. Zu sehen, wie die Roboter Props durch den roten Laser zerstört werden, hat somit Bedeutung für den Spieler in Form der Narrativen, aber auch um zu verstehen, dass rote Laser gefährlich sind. Trotzdem befindet sich der Spieler in einer relativ sicheren Umgebung. Alles weitere baut hierauf auf. Sobald der Spieler von dem roten Laser Bescheid weiß, und dass er Spiegel rotieren kann, wird dies sofort im nächsten Parkour-Level wichtig. Level 03 ist die gleiche Geschichte. Aus vorherigen Leveln kennt der Spieler den Schmelzblock. Hier wird dieser genutzt, um ein Zeitfenster zu schaffen. Auch dort habe ich einen „Scrap Shoot“ genutzt, um dem Spieler indirekt zu zeigen, wie das Level neu gestartet werden kann. Das Level wurde so entworfen, dass der Spieler Fehler machen kann und auch soll, um zu lernen. Alle Elemente sind aus einem Grund hier und haben ihre eigene Aufgabe.

Da die Rätsel zunehmend schwerer wurden, wurden seit der MVP-Vorstellung Tipps vorgeschlagen, welche den Spieler Hinweise zum Lösen von Rätseln und Parkour Segmenten geben sollen. Dies habe ich in Form von Decals an Hintergrund Panels umgesetzt. Diese Hinweise sind darauf ausgelegt, dem Spieler einen Gedanken-Schub zu geben. Der Spieler kann sich dennoch dafür entscheiden, diese zu ignorieren.



Da ich anfangs auf Affordanz zu sprechen kam, möchte ich auch auf die Kabel eingehen, welche Mechanismen verbinden. Dem Spieler soll klar werden, wenn dieser einen Knopf drückt, was er bewirkt und in welchem Status sich der Mechanismus gerade befindet. Hierfür habe ich Kabel implementiert, welche Trigger und Effektoren miteinander verbinden und im Aktiv-Zustand leuchten, um dies dem Spieler klarzumachen. Der Spieler sollte sich keine unnötigen Gedanken über Zusammenhänge in Levels machen, wo es nicht benötigt wird.

Ich möchte zuletzt einmal auf die Ästhetik der Levels eingehen. Ich war zwar vollkommen verantwortlich über die oben genannten Level, habe aber auch die restlichen Level von Florian und Prince vollendet und weiter angepasst, sodass alle Level relativ konsistent miteinander sind. Gerade das Dekorieren der Levels hat große Zeit in Anspruch genommen, von der Platzierung der Panels und der einzelnen Blöcke und gerade das sicherstellen, dass alle Blöcke in einem Grid perfekt

nebeneinander liegen. Ich habe mich um Abwechslung bemüht, dass die Level nicht blank und weiß aussehen. Dafür habe ich schwarze sog. Konstruktionsblöcke genutzt, welche einen Kontrast bilden und das Level strukturell stützen sollen. Hinzu kommen weitere Elemente wie die oben genannten Kabel, Metallzäune, Überwachungskameras, wie auch dekorative Laserquellen und verschiedene Hintergrund Panels.

(9) Kamera & Post Processing

Ich habe mich in der Vergangenheit mit einem Unity Addon „Cinemachine“ auseinandergesetzt und habe gute Erfahrungen damit sammeln können. Dieses Addon erlaubt es u.a. die Kamera-Pfade zu manipulieren. Damit konnte ich einen Raum definieren, in welchem sich die Kamera bewegen darf. Außerdem konnte ich die Bewegung der Kamera interpolieren, damit sie den Spieler leicht verzögert folgt. Hierzu wird eine virtuelle Kamera konfiguriert und steuert die tatsächliche Kamera in der Szene. So konnte ich entscheiden, welche Bereiche im jeweiligen Level im Fokus stehen, und welche eher abgeschnitten werden sollten, damit der Bildschirm optimal mit Inhalten gefüllt wird, die für den Spieler bedeutungsvoll sind.

Außerdem habe ich mich mit Post Processing Effekten auseinandergesetzt und diese in jeder Szene einheitlich mit eingebracht, um subtile Effekte abbilden zu können. Hierfür habe ich einen leichten Tiefenschärfe-Effekt hinzugefügt, damit der Hintergrund und dessen Panels besonders an ihren Übergängen ein wenig glatter werden. Dazu kommt die chromatische Aberration, welche insbesondere bei weißer Farbe für schöne Effekte am Rand sorgt und ein Color Grading Effekt, indem ich die Farben weniger gesättigt habe. Zusammen mit der Richtungslichtquelle wollte ich erzielen, dass die Farben nicht zu intensiv sind, gerade da viele weiße Elemente in den Szenen vorkommen. Doch wollte ich auch für eine angenehme Farbsättigung sorgen, sodass farbige Elemente wie Laserstrahlen oder auch Kisten dem Spieler sofort ins Auge fallen und ihn ansprechen. Zuletzt habe für sanftes Anti-Aliasing und Soft-Shadows gesorgt, welche die Qualität erheblich verbessert haben. Ich habe lange mit diesen Optionen und Effekten experimentiert, um die idealen Werte für ein visuell ansprechendes Bild zu finden.

(10) Bewegliche Plattformen

Die beweglichen Plattformen wurden bereits von Prince implementiert, mir fehlte es jedoch an Kontrolle und der Code konnte optimiert werden. Deswegen habe ich mich schnell an ein neues Skript gesetzt und die beweglichen Plattformen neu implementiert. Diese neue Implementierung bietet neben der Einstellung der Bewegungsgeschwindigkeit die Option, den Zielort der Plattform mit anzugeben. Somit muss nicht mehr zwischen vertikalen und horizontalen Bewegungen und Distanzen unterschieden werden. Außerdem habe ich 2 einfache Prefabs erstellt, welche den Stahlträger und einen einfachen Block zu beweglichen Plattformen machen.

Doch einer der wichtigsten Änderungen im Vergleich zur alten Implementierung war das „Parenting“ des Spielers unter der beweglichen Plattform. Dies stellt sicher, dass sich der Spieler auf der Plattform immer relativ zur Position der Plattform bewegt. Somit rutscht der Spieler nicht mehr von der Plattform, sondern bewegt sich mit der Plattform. Hierfür war ein zusätzlicher Trigger Collider

notwendig, der überprüft, ob der Spieler auf der Plattform steht. Wichtig ist hier, dass der Collider an den Seiten schmaler sein muss, sonst wird der Spieler auch an den Seiten mit der Plattform gezogen.

(11) Menü Redesign

Uns war bis zum MVP wichtig, dass wir ein grundlegendes Menü anbieten wollten. Darum hat sich Florian gekümmert und alle technischen Aspekte implementiert. Doch war uns klar, dass wir an dem Design des Menüs noch zu arbeiten hatten. Da ich mich generell auf Design Aspekte des Projektes spezialisierte, vom Level Design bis hin zur Modellierung von Assets, habe ich die Aufgabe übernommen, das Menü neu zu designen und für das Hauptmenü eine ansprechende Hintergrund-Szene zu entwerfen. Ich habe mich um das Redesign des Hauptmenüs, des Optionsmenüs, des Steuerungsmenüs und des Pause-Menüs gekümmert.

Bereits am Anfang wurde die Farbe rot in den Hover Effekten benutzt. Ich habe mich dazu entschieden, diese Farbe durchgängig zu benutzen, sowohl als Hover Effekt Farbe, als auch als Füllfarbe von Slidern. Generell war es mir besonders wichtig, den Stil konsistent zu halten. Ich habe außerdem eine passende Schriftart gefunden, welche ich durchgängig benutzt habe. Des Weiteren habe ich Button-Anordnungen geändert, Größenverhältnisse angepasst und Elemente verschoben und neu verankert. Auch habe ich den Hintergrund im Hauptmenü und im Spiel verdunkelt, sodass die Menüs und die UI-Elemente im Vordergrund stehen und besser lesbar sind.

Für das Hauptmenü habe ich eine neue Hintergrund-Szene entworfen. Ich wollte das Kernelement des Spiels, die Laser und den Spieler mit einbringen, um in dem Spieler die Erwartung zu wecken, was ihm bevorsteht. Dafür habe ich einen Raum entworfen, in dem Laserquellen ihre Laser verschießen, Partikeleffekte abspielen und Roboterteile von oben nach unten fallen. Diese Teile interagieren mit den Lasern und blockieren diese kurzzeitig, um die Dynamik der Szene zu betonen. Die Designentscheidung, Roboterteile im Hintergrund fallen zu lassen, steht im direkten Zusammenhang mit dem ersten Level, in dem der Spieler initial auch runterfällt und neben ihm diese Teile sieht, die an seiner Stelle geschreddert werden. Außerdem steht der Spielercharakter auf einem der schwarzen Panels, auf denen der Titel des Spiels abgebildet ist. Dieser steht in seiner Idle-Animation und bewegt sich leicht. Die Panels habe ich leicht angewinkelt wie ein geöffnetes Buch platziert und den Titel des Spiels an beiden Seiten angebracht. Diese Panels sorgen für mehr Tiefe in einem anderweitig 2,5-dimensionalen Spiel. Dies wird stärker betont, indem diese schwarzen Panels über den Rahmen der dahinterliegenden Umgebung schauen. Sie brechen quasi den Rahmen auf.

Eine Kleinigkeit, die mir im Nachhinein aufgefallen ist, ist das Optionsmenü. Denn in der Hintergrund-Szene gibt es Punktlichtquellen, die die Szene an den Enden der Laserstrahlen beleuchten. In diesem Menü kann man die Qualität einstellen und diese wird auch direkt übernommen. Der Spieler kann dies sofort an der Beleuchtungsqualität im Hintergrund sehen, welche sich ändert. Diese Szene bietet somit noch einen unerwarteten Mehrwert für den Spieler, außer nur schön auszusehen und das erste Level zu antizipieren.

(12) Projektorganisation

Wir haben uns stets über Aufgabenverteilungen und Fortschritte in dazu vorgesehene Kanäle ausgetauscht. Hierfür nutzten wir u.a. Discord und MS Teams. Hin und wieder haben wir zusammen an bestimmten Funktionalitäten gearbeitet und Aufgaben so verteilt, dass diese gut parallelisiert werden konnten. Dies hat sich als sinnvoll erwiesen, denn wir hatten in unserer Entwicklungsphase nur wenige Merge-Konflikte. Zudem habe ich meine Zeit in einer Excel-Tabelle protokolliert.

Zum Thema des fehlenden Gruppenmitgliedes sollte noch genannt werden, dass wir uns in unserer Organisation und unseren geplanten Inhalten deutlich umstrukturieren mussten, um den Schaden und die fehlende Kapazität zu kompensieren.

(13) Dokumentation

Diese Dokumentation nimmt für mich eine wichtige Rolle ein, da wir so unseren Gedankengang sehr detailliert darlegen und Designentscheidungen hervorheben können, die eventuell nicht aufgefallen wären. Da sich unser Projekt um ein Puzzle-Platformer handelt, war es uns sehr wichtig, dass im Level Design so gut wie jedes Element seinen Sinn hat. Dieser Bericht nimmt dementsprechend aber auch eine signifikante Zeit in Anspruch.

(14) Sonstiges

In diesem Abschnitt möchte ich auf Arbeitspakete eingehen, die nicht wirklich eingeordnet werden konnten, oder als separater Punkt überflüssig wären. Während der Entwicklungsphase bin ich oft umher gesprungen und habe an Teilen von Gruppenmitgliedern gearbeitet, um diese zu optimieren oder auch zu vereinfachen. Es handelt sich auch um einige Kleinigkeiten und Funktionalitäten. Als Visions-Träger des Projektes war es meine Aufgabe, insbesondere alle Komponenten reibungslos miteinander zu verbinden, Anpassungen vorzunehmen und Level richtig anzuordnen, auch wenn sie von verschiedenen Mitgliedern entwickelt worden sind. In ihrer Einzelheit hätten sie nicht viel Zeit gekostet, in ihrer Gesamtheit hat sich jedoch signifikant viel Zeit angesammelt.

Themen Prince Lare-Lantone

KURZÜBERSICHT

Zeit Gesamt: 135 Stunden

1. Einarbeiten in Unity: 2 Stunden
2. Tod des Spielers durch Laser mit Flo: 4 Stunden
3. Schmelz Mechanik mit Flo: 3 Stunden
4. Geschütz Gegnertyp (aus Endprodukt gestrichen): 6 Stunden
5. Character Controller: 6 Stunden
6. Animation : 12 Stunden
7. Level: (Gesamt 68 Stunden)
8. Bewegliche Plattformen (1. Iteration): 1 Stunde
9. Meeting und Organisation: 15 Stunden
10. Playtest Auswertung (Prototype): 6 Stunden
11. Dokumentation: 12 Stunden

ERKLÄRUNG

1. Zu Beginn des Projektes hatte ich vorher noch nicht mit Unity gearbeitet. Demnach musste ich mir zunächst ein paar Grundlagen aneignen. Hierzu habe ich ein paar Videos geschaut und generell ein wenig in der Unity Engine rumprobiert. Flo und Jonathan waren hierbei auch eine große Hilfe und wir haben uns im Verlauf des gesamten Projektes gegenseitig mit Tipps und Techniken in Unity unterstützt.

2. Siehe Kapitel “Tod durch Laser von Prince und Florian”

3. Siehe Kapitel “Schmelzen der Blöcke von Prince und Florian”

4.Bei einem unserer ersten Meetings und „Brainstormings“ zu den Mechaniken und Tools, die wir im späteren Spiel verwenden wollen, kam uns die Idee eines Geschützturmes. Dieser Geschützturm sollte ein einfacher, aber wirkungsvoller Gegnertyp sein, der das Spielerlebnis durch seine Fähigkeit, bei Sichtkontakt mit dem Spieler ein moderat schnelles Projektil abzufeuern, intensiviert. Ein Treffer dieses Geschosses bedeutet das sofortige Ableben des Spielers – eine Herausforderung, die das Gameplay auf eine neue Ebene hebt.

Um diese Idee zu konkretisieren, begann ich mit der Suche nach einem passenden Modell für den Geschützturm. Da unser Spiel ein Fabrik-Setting hat, war es wichtig, dass das Modell sowohl in seinem Design als auch im Stil zum Rest des Spiels passte. Nach gründlicher Recherche entschieden wir uns für ein Open-Source-Modell, das unseren Anforderungen entsprach. Es wurde jedoch offen gelassen, ob wir möglicherweise später ein maßgeschneidertes Geschütz Modell erstellen würden.

Nachdem das Modell feststand, war die nächste Herausforderung die Implementation des Geschützturms in das Spiel. Hierfür wurde ein umfassendes Verhaltensskript mit dem Namen "Robo1Behaviour" verfasst. Dieses Skript umfasste alle erforderlichen Funktionen des Geschützturms, darunter:

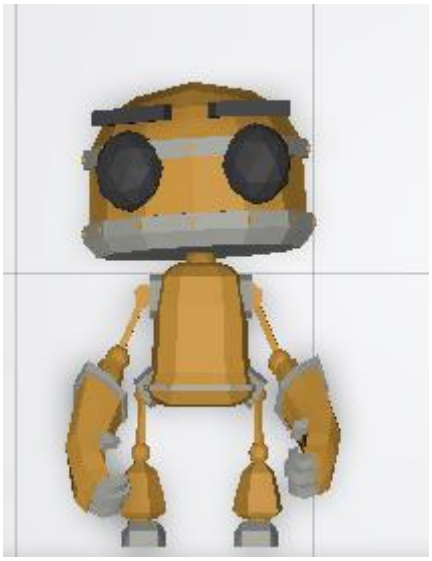
- Ein Raycast, um zu überprüfen, ob der Spieler im Sichtbereich des Geschützturms ist.
- Variablen zur Kontrolle der Schussfrequenz, Geschwindigkeit des Projektils und Sichtweite des Geschützturms.
- Logik, um bei Sichtkontakt mit dem Spieler ein Bullet-Prefab zu erzeugen, das sich mit einer festgelegten Geschwindigkeit auf den Spieler zubewegt. Das Projektil wird automatisch zerstört, wenn es keine Kollisionen hat, oder nach Ablauf einer bestimmten Zeitspanne.
- Die Kollision des Spielers mit dem Projektil löst den Tod des Spielers aus.
- springt der Spieler auf das Turret stirbt stattdessen dieses.

Nachdem die grundlegende Logik des Geschützturms implementiert war, folgten ausgiebige Testläufe, um die optimalen Werte für die verschiedenen Variablen zu ermitteln. Trotz unserer Bemühungen stellten wir im Verlauf des Projekts fest, dass die bereits vorhandenen Grundmechaniken des Spiels – insbesondere die Plattform-Elemente und Laser – bereits eine Vielzahl von Herausforderungen und Gameplay-Varianten boten. Eine sinnvolle Integration des Geschützturms erwies sich als schwierig.

Letztendlich entschieden wir daher, den Geschützturm vorerst aus dem finalen Spiel zu streichen. Dies war sowohl auf die mangelnde Notwendigkeit innerhalb der bereits erstellten Level als auch auf die Unstimmigkeit seines Einsatzes mit dem übrigen Gameplay zurückzuführen. Außerdem hätte die Implementierung zusätzliche Arbeit für Animationen und Verhaltensweisen erfordert, was Ressourcen beansprucht hätte, die stattdessen zur Verbesserung und Erweiterung anderer, bereits etablierter Spielelemente genutzt werden konnten.

5. Siehe Kapitel “Character Movement von Prince und Florian”

6. Animation ist das Herzstück eines Charakters, da sie seine Reaktionen auf die Umgebung darstellt und möglichst natürlich wirken sollte. Als ich mich zu Beginn des Projektes dieser Aufgabe annahm, war ich ein absoluter Neuling auf diesem Gebiet. Ich hatte keine Vorstellung davon, wie viel Zeit und Aufwand ich für das Mikromanagement und die feinsten Anpassungen investieren würde.



Zunächst musste allerdings ein Charakter Modell her, bis zum Proof of Concept nutzten wir eine Capsule, Meine Suche nach einem Placeholder-Modell für unseren Charakter gestaltete sich langwierig, doch schließlich stieß ich auf ein Open-Source-Modell, das perfekt zu unserer Vision passte und uns als Gruppe begeisterte. Dieses Modell enthielt sogar eine Fülle von vordefinierten Animationen, die sich theoretisch direkt einsetzen ließen.

Der Umgang mit dem Animation Controller stellte mich jedoch vor Herausforderungen. Es erforderte einige Zeit, bis ich herausfand, wie ich auf die vorhandenen Animationen des Modells zugreifen konnte, da sie alle direkt in das Modell eingebettet waren und nicht in separaten Dateien vorlagen. Für einen Anfänger wie mich war dies eine Hürde.

Bevor ich mich jedoch mit der Logik und den verschiedenen Zuständen des Animation Controllers auseinandersetzen konnte, musste ich zunächst lernen, welche Funktionen und Optionen überhaupt zur Verfügung standen. Dabei waren offizielle Guides, Videos und eigenes Ausprobieren meine besten Verbündeten.

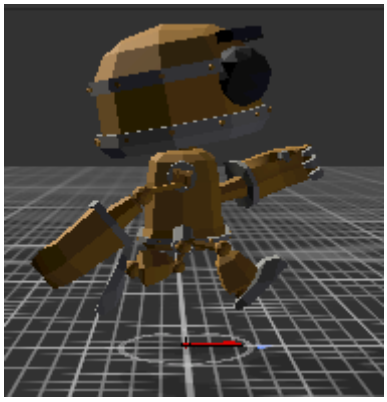
Beim Animieren traten verschiedene Probleme auf, die es zu beachten galt. Zunächst mussten wir entscheiden, welche Animationen benötigt wurden. Wir entschieden uns für "Idle", "Jump", “Walk” und "Death". Die “Walk” animation sollte ursprünglich aus zwei Animationen bestehen je nach Bewegungsgeschwindigkeit, dies wurde nach einigem Herumprobieren jedoch verworfen, da es zum einen nicht notwendig war und die Bewegungsgeschwindigkeit des Robos nicht zur “Run” animation gepasst hat, und zum anderen hätte es die Logik für State wechsel unnötig verkompliziert und fast doppelt so viele Zustandsänderungen hinzugefügt.

Insgesamt elementar wichtig für den Animation Controller ist es, die korrekten Trigger, Übergänge und Bedingungen zu definieren.

Hierzu bedarf es eines “Default” states, dies ist die Animation die direkt zum Start ausgelöst wird, von hier aus startet der Animation Tree und es ist der Ausgangspunkt für alles andere, was passieren kann. Bei uns ist das die Idle-Animation, sie wiederholt sich automatisch mit Exit-Time (um sich nicht selbst zu unterbrechen).

Der einfachste Trigger ist unsere Death Animation. Sie kann aus jedem anderen State aufgerufen werden, Interrupted diesen ohne Exit-Time und wird durch einen Deathtrigger ausgelöst.

Springt man gibt es die “Walkjump” Animation, diese habe ich nachträglich angepasst, sodass die Beine gespreizt bleiben, wenn die Fallzeit länger ist. In der ursprünglichen Variante hat der Charakter die Beine geschlossen und das “Fallen” sah nach dem Sprung sehr unnatürlich aus.



Diese Animation kann von Idle aus durch einen “Jumptrigger” und der IsGrounded Variable auf False ausgelöst werden. IsGrounded wurde für lange Zeit mithilfe eines Colliders bestimmt, dies führte jedoch zu Bugs, Einschränkungen und Problemen wie das nicht auslösen der “Jump” Animation, deshalb ist es nun stattdessen mithilfe eines Raycasts weitaus zuverlässiger. Ansonsten kann “WalkJump” auch von der “Walking” Animation aus gestartet werden durch dieselben Bedingungen wie aus Idle. Die “Walking” Animation hingegen kann von “Jump” und “Idle” jeweils gestartet werden, wenn die Bedingungen IsGrounded und IsMoving erfüllt sind, sie loopt in sich selbst. IsMoving testet hierbei auf eine horizontale Bewegung.

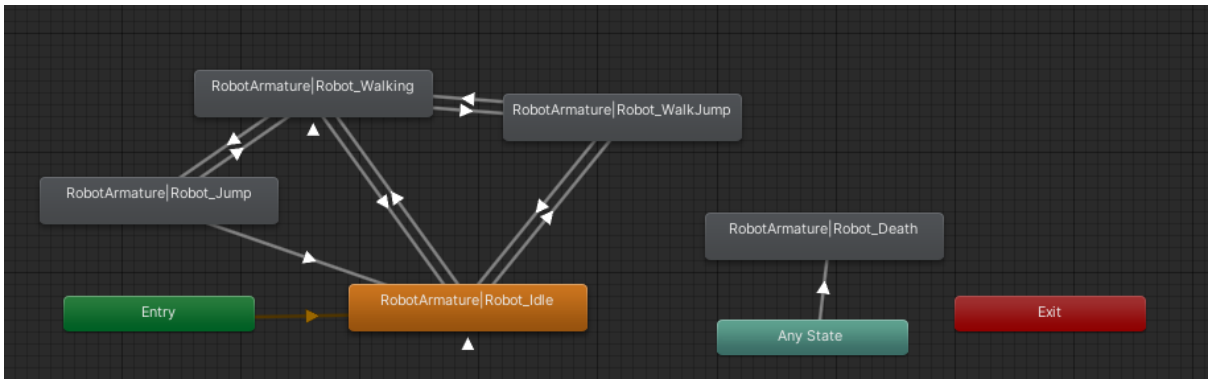
Für lange Zeit gab es einen Bug Innerhalb der Animation: sollte der Fall eintreten, dass man Springt, dabei mit dem charakter Modell mit einem Objekt so Kollidieren, dass der Groundcheck als positiv

ausgewertet wird, so kam es vor, dass der Charakter aus der Jump Animation direkt in eine “Walking” Animation Übergeht und demnach in der Luft läuft.

Um dieses Problem zu beheben, habe ich eine weitere Animation verwendet. Diese Animation steuert das Verhalten, wenn der Charakter ohne Jump Trigger plötzlich nicht mehr auf dem Boden ist, dazu zählt, irgendwo herunter zu laufen und natürlich der vorher erwähnte Bug.

Die Animation “RobotJump” beinhaltet einen kleinen Hopser, der sehr passend ist für den Übergang vom normalen Laufen zum in der Luft sein. Sie kann nur aus dem Walking aus aufgerufen werden wenn IsMoving erfüllt ist und man nicht grounded ist.

Von dieser Animation aus kann man entweder zurück ins “Walking” oder ins Idle übergehen.



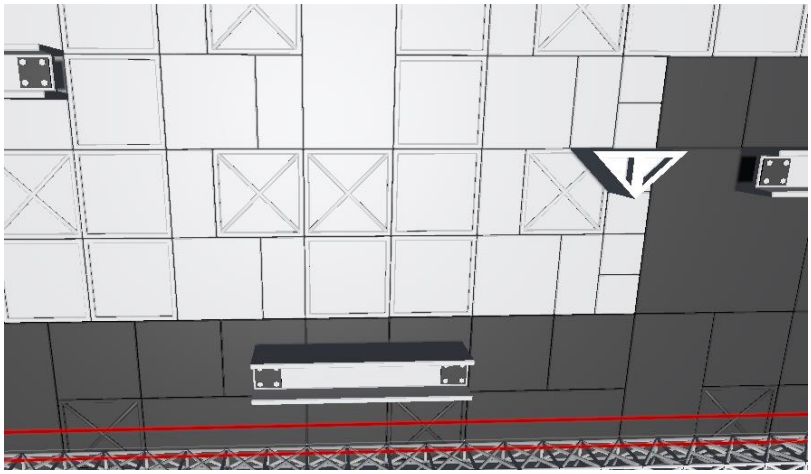
Ganz zu Anfang wurden die Animationen, Trigger und Variablen über ein separates Skript gesteuert, dies stellte sich jedoch als nicht Sinnvoll heraus, da vieles ohnehin Daten aus dem Charactercontroller benötigte, deshalb wurde dies alles dann in den Charactercontroller mit eingebettet.

Trotz all dieser Herausforderungen und Lernkurven war die Arbeit an den Animationen ein lohnender Prozess, der letztendlich dazu beitrug, dass unser Charakter lebendig und authentisch in der Spielwelt wirkte.

7. Der Prozess, ein Level zu entwerfen, zu gestalten und zu implementieren, hat mich ein wenig überrascht, und es wird schnell ersichtlich, dass dies für mich am meisten Zeit beansprucht hat. Ich habe hierbei versucht, relativ strukturiert vorzugehen und nicht einfach drauflos zu bauen, und würde behaupten, dass mir dies auch relativ gut gelungen ist. Generell hatte vorrangig Jonathan, zum Zeitpunkt, als ich angefangen habe, Level zu designen, schon ein paar Level fertiggestellt. Diese waren mit explizitem Fokus auf Rätsel mithilfe der Spiegel, Blöcke und Lasermechaniken. Ich hingegen wollte mehr Fokus auf den Parcour- und Plattformer-Teil legen, da dies bisher noch nicht sonderlich viel Verwendung fand und den Spielfluss ein wenig ankurbeln kann. Die Differenz zwischen Kopfsache (Rätsel) und Timing und Reflexen war mir wichtig, denn beides allein stehend im Übermaß kann zu Ermüdung führen. Deshalb sollte am besten eine gesunde Mischung erreicht werden, bei der natürlich immer noch klar erkennbar der Laser der Hauptaspekt unseres Spiels bleibt.

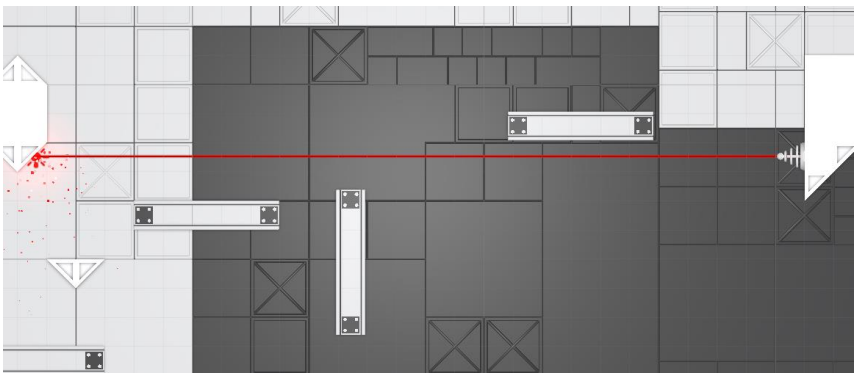
Zum Start des Level-Entwickelns stand bei mir eine Idee und ein Name. Hierbei waren die ersten beiden "Parcour" und "Three Rooms". Zunächst gehe ich auf das "Parkour"-Level weiter ein. "Parkour" sollte, wie der Name schon sagt, den Fokus auf Platforming und Parcour haben. Hierbei wollte ich die

Grenzen unserer Steuerung ein wenig ausreizen und ein paar Stellen des Levels wirklich knapp gestalten. Es sollte eines der späteren Level werden, bei denen der User sehr gefordert wird, ohne Rätsel lösen zu müssen. Für das Design dieses Levels habe ich mir zunächst ein wenig Inspiration gesucht und diverse Level-Layouts des



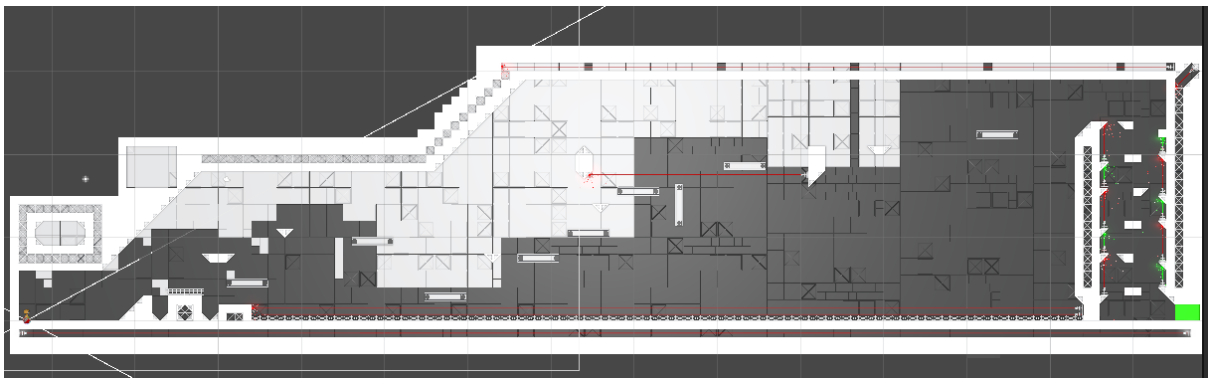
ersten Super Mario angeschaut und dabei analysiert, wie genau die Elemente platziert sind, miteinander interagieren und was dabei für Schwierigkeiten sorgt. Dabei ist mir aufgefallen, dass unser Spiel so grundlegend unterschiedliche Möglichkeiten hat, dass ich mich nicht unbedingt nach Mario richten kann. Nichtsdestotrotz hat es mir einen Einblick in die Gestaltung von Parkour-Mechaniken gegeben, der im weiteren Verlauf geholfen hat. Als nächstes habe ich dann angefangen, den ersten Entwurf des Levels zu skizzieren und mir weitere Gedanken zu Hindernissen gemacht. Hierbei kam mir dann eine absolut offensichtliche und dennoch noch nicht implementierte Idee für ein Element, das essentiell für unser Platforming werden würde - bewegliche Plattformen. Eine vergleichsweise unglaublich simple Mechanik, die aber durch Flexibilität in der Verwendung überzeugt, zusätzlich das Spiel durch Bewegung "lebendiger" wirken lässt und mit Variabilität in Geschwindigkeit und Bewegungsdistanz eine weitere Möglichkeit bietet, die Schwierigkeit zu steuern. Nachdem ich eine grobe Skizze für den Verlauf des Levels, den ich mir vorstellte, hatte, ging es nun darum, dies soweit zu implementieren, dass ich damit testen konnte. Block für Block habe ich also dann einen Grundriss für das Level erstellt und angefangen, die Distanzen zwischen den Hindernissen mit den Möglichkeiten unserer Steuerung abzugleichen. Hierbei habe ich sehr viel ausprobieren müssen, um zu wissen, was wie und unter welchen Umständen möglich ist und vor allem wie schwer es sein würde, diese Hindernisse zu überwinden. Das Herzstück und die schwierigsten Teile des Levels stellen explizit zwei Passagen dar. Zum einen gibt es relativ nah am Anfang eine Plattform, die sich unter den Todeslaser, der am Boden des Levels verläuft, bewegen kann. Hier muss der Spieler entweder in einem schnellen Moment, wenn die Plattform an ihrem höchsten Punkt ist, auf diese springen und dann zur nächsten hüpfen, oder wenn der Spieler das Timing verpasst und die Rettung erkennt, einmal springen, wenn die Plattform unter dem Laser verschwindet, und dann auf ihr landen, sobald sie wieder darüber ist.

Die zweite Stelle und mit Abstand der schwerste Teil des Levels ist ein Konstrukt aus einem horizontalen Laser, der so über einer Plattform verläuft, dass man zwar noch darunter stehen kann, aber nicht springen kann, einer

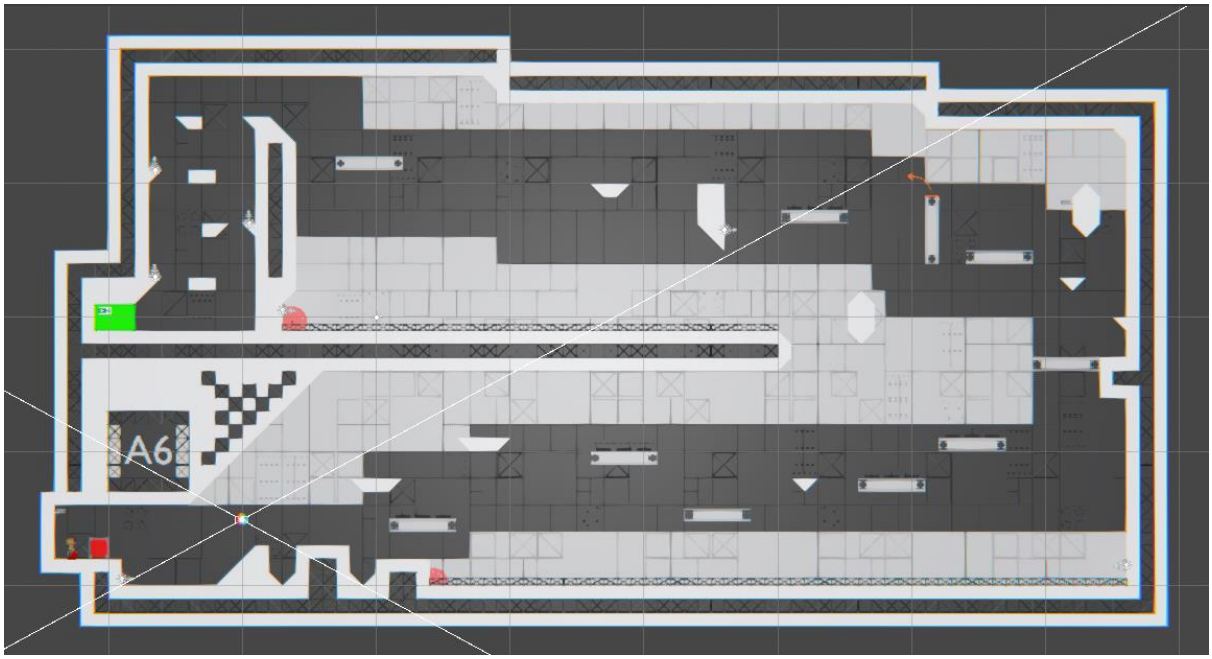


vertikalen Plattform, die sich durch diesen Laser bewegt und ihn zeitweise blockiert, und Distanzen, die durch Springen überwunden werden müssen.

Dieses Hindernis erforderte sehr viel Testing und wurde auch mehrfach neu angepasst. Es ist sehr leicht, es als zu schwer oder sogar unmöglich anzusehen, und selbst ich als Entwickler bin unzählige Male hier gestorben, während ich geprüft habe, ob es noch schaffbar ist. Das Problem war allerdings, dass die Mentalität "ist es möglich?" von meiner Seite hier leider ein wenig fehl am Platz war, denn es resultierte darin, dass ich es zu schwer gemacht hatte für unsere Prototyp-Playtests. 2 von 4 Testern haben dieses Hindernis nie überwunden und den Test hier beendet. Einer davon ging sogar davon aus, dass es physikalisch unmöglich sei. Glücklicherweise war dies ohnehin unser letztes Level und relativ am Ende des Levels. Dies war jedoch für mich auf jeden Fall ein Weckruf, das Hindernis weitaus einfacher zu machen, indem ich die Plattform langsamer mache und länger in Laserblock Position verweilen lasse. Den Wunsch nach einem Checkpoint vor einer schwierigen Stelle habe ich lange in Betracht gezogen, aber aufgrund der Schwierigkeitsgrad Senkung nicht implementiert. Dazu kam auch, dass aufgrund der Schwierigkeit einem Teil der Tester nicht bewusst war, dass sie tatsächlich auf die Laser blockierende Plattform springen müssen, und sie stattdessen andere Dinge ausprobiert haben, dies hat uns dazu veranlasst, hier auch noch einen kleinen Hinweis einzubauen, wo genau man hin springen muss. Letztes Problem, was in den Playtests zu diesem Level auch auffiel, war, dass unser Field of View für das Level zu gering war und sich gewünscht wurde, mehr zu sehen, um schnell zu wissen, wo man hin muss und was für Hindernisse einem voraus liegen. Dies wurde auch später angepasst. In der allerersten Version des Levels "Parkour" geht es sehr weit in die Horizontale.



Dies stand im Konflikt mit anderen Levels, hat sehr viel ungenutzten Platz eingebracht und sah im Allgemeinen nicht so gut aus. Aus diesem Grund musste ich das Level grundlegend (bei Beibehalt aller



Mechaniken) in der Form verändern. Dazu habe ich es in der Mitte geteilt und den zweiten Teil sehr stark vereinfacht ausgedrückt, gespiegelt und oben drauf gesetzt.

Dies wiederum hat zu sehr vielen Tweaks und Anpassungen geführt, da sich nicht alles einfach so verschieben ließ und immer noch klar funktioniert hat.

Der letzte Teil des Leveldesign-Prozesses war es dann, das Ganze "aufzuhübschen", das Look and Feel aufzuwerten. Viele Teile davon habe ich schon zwischendurch implementiert, wie den Hintergrund, damit beim Testen schon ein Gefühl gegeben wird, ob etwas funktioniert im Gesamtkonzept und wie sich das Level selbst entwickelt. Zudem wurde auch an der Form des Levels insgesamt mehrfach gearbeitet, um möglichst viel "empty space" zu entfernen. Im Verlauf dieses Prozesses musste man sich vor allem ständig die Frage stellen, ob etwas nun wirklich notwendig ist und welche Teile man zugunsten der Performance vielleicht nicht unbedingt benötigt. Da unsere Laser zum Beispiel weit mehr die Performance beeinflussen als Konstruktions Blöcke, habe ich teilweise im Verlauf der Bearbeitung einige rein ästhetische Laser und andere Elemente durch weniger aufwändige ersetzt, um zwar immer noch ein schönes Design zu erhalten, aber hierbei dennoch die Leistung zu maximieren.

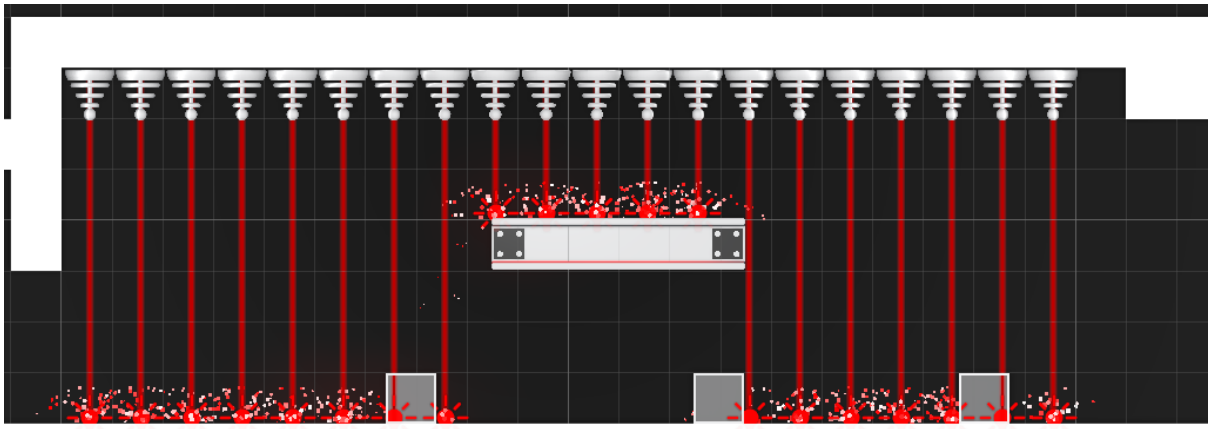
"Three Rooms" war durchgängig der Codename des Levels, das später unser Finale werden würde, obwohl schon relativ früh klar wurde, dass es keine drei Räume geben würde. Meine grundlegende Idee war es, ein Level zu gestalten, das aus drei Räumen besteht, die jeweils gelöst werden müssen, um am Schluss das Ziel freizuschalten. Ein Raum sollte ein Spiegelrätsel sein, ein Raum für Platforming und einer explizit auf die Geschütztürme, welche im späteren Verlauf ganz aus dem Spiel gestrichen wurden, ausgerichtet sein. In der Skizzierungsphase habe ich jeden Raum separat skizziert und mir überlegt, wie ich diese so verbinden kann, dass die Notwendigkeit, sie alle zu bewältigen, vom Spieler selbst erkannt wird und nicht einfach nur auf der Existenz der Räume basiert. Ich habe aber festgestellt, dass drei Räume entweder jeweils zu klein sind, um alleinstehend eine Herausforderung zu sein, oder insgesamt zu groß werden würden im Vergleich zum Rest. Dazu kam, dass das Geschütz ohnehin noch keine Verwendung hatte. Ein Raum mit dem Geschütz als Thema würde zudem großteils auf Projektilen basieren, denen man ausweichen muss, was nun in starkem Kontrast zum Rest des

Spiels stehen würde. Aus diesen Gründen habe ich Raum 3 gestrichen und mich auf die beiden anderen fokussiert.

Nach einiger Überlegung habe ich auch eine Möglichkeit gefunden, das Spiegelrätsel explizit mit dem Parcour teil zu verbinden und die Reihenfolge der Räume klar vorzugeben.

Hier sieht man eine der ersten Entwürfe für das Level, ohne sehr weit ins Detail zu gehen, es wird ersichtlich, dass im oberen Raum ein Spiegelrätsel existiert und ganz oben eine orange Kiste schwebt. Schaut man sich nun den unteren Raum an, so wird klar, dass ein Todeslaser über den gesamten Boden geht, auch bei dem Schacht, durch den man nach unten gelangt. Die Grundidee, die hier entsprungen ist, ist: Löse das Rätsel, bekomme die Kiste und schiebe sie nach unten, sodass man den Laser blockiert und Zugriff auf den zweiten Teil des Levels erhält. Hierbei gab es nun Unmengen an Verbesserungsbedarf, der durch Testen, Iterieren, Umbauen, weiteres Testen etc. erkannt wurde. Das Laser-Rätsel ist ein wenig zeitaufwändig, selbst wenn man es schon mal gelöst hat, demnach ist es unglaublich frustrierend, danach bei der Parcour-Passage zu sterben. Dazu ist es relativ leicht, in der Parcour-Passage zu sterben, da ich mir dazu ein völlig neues Hindernis überlegt habe, welches beim ersten Antreffen fatal sein kann. Hier ein kurzer Einschnitt zu besagtem Hindernis:

- Eine Passage gefüllt mit Lasern, eine Plattform, die als “Regenschirm” verwendet werden muss, unter der man laufen muss, um ans Ende zu gelangen. Auf dem Boden sind jedoch drei Blöcke, die den Spieler unter der Plattform hervorschieben können.



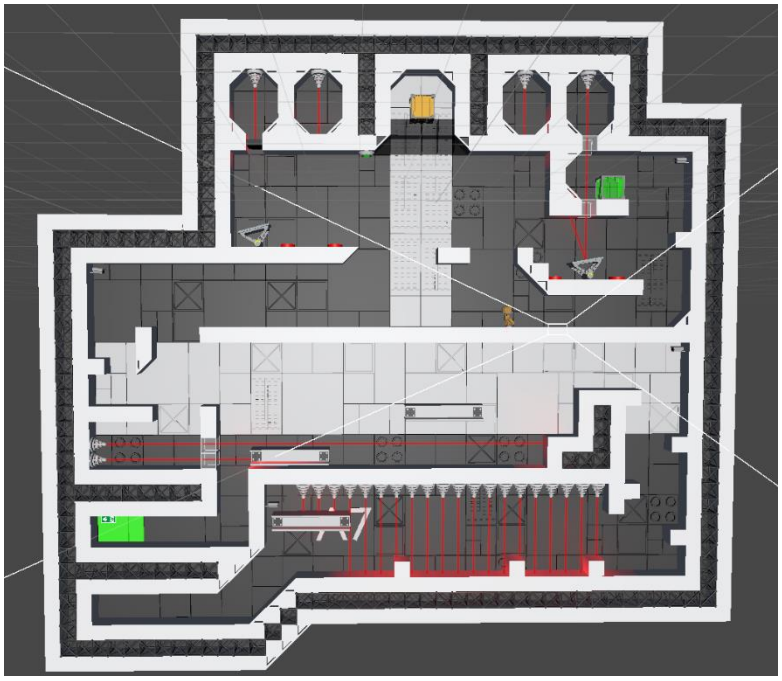
Das Level geht beim Spiegelrätsel zu weit in die Vertikale, der Spieler sieht nicht genug, um das Rätsel zu verstehen. Insgesamt auch wieder zu viel "empty space" im Level. Zu viele Spiegel machen das Schmelzen der Blöcke frustrierend, da mit jedem Spiegel die benötigte Präzision und das Micromanagement erhöht werden. Nicht genug Platz, um alle benötigten Buttons unterzubringen zur Steuerung der Spiegel. Dies sind so die wesentlichen Problematiken, die im Verlauf durch vollkommenes Umgestalten gefixt wurden.

Hierzu habe ich zunächst die Reihenfolge des Levels geändert. Der schwere Parcour-Teil, an dem man leicht durch Ungeduld, Überraschung oder Ähnliches sterben kann, an den Anfang. Das Rätsel an das Ende, um beruhigt das Level zu beenden. Dies hat außerdem den Nebeneffekt, dass die Passage mit den vielen Lasern direkt als Einleitung für das Finale dient, es sieht besonders gefährlich aus und ein bisschen "over the top". Beim Rätsel die Anzahl der zu schmelzenden Schichten sowie die Spiegel verringert. Das gesamte Level komprimiert, "empty space" entfernt und das Ziel in die Mitte eingebettet, hierfür wird nun die Kiste benötigt. Klingt nach leichten Änderungen, hat aber das Level vollkommen umgebaut, wie in dem Screenshot ersichtlich. Auch hier wurde ähnlich wie beim "Parcour"-Level ein Teil gespiegelt, andere Elemente hinzugefügt und andere geändert, sodass die Mechaniken weiterhin funktionieren. Bis zum Finale hat es mehrere Iterationen gegeben, und vor allem das Anpassen des Designs und Testen war sehr aufwändig, aber es hat sich gelohnt.

8. Nachdem ich feststellte, dass bewegliche Plattformen in unserem Spiel gut passen und flexibel als Hindernisse dienen könnten, entschied ich mich, ein Skript dafür in Unity zu schreiben. Zu Beginn hatte ich jeweils ein Skript für horizontale und ein Skript für vertikale Bewegung erstellt, um den unterschiedlichen Anforderungen gerecht zu werden. Diese Skripte ermöglichten es den Plattformen, sich entsprechend der Spielmechanik zu bewegen und mit den Spieleraktionen zu interagieren.

Später, nach weiterer Entwicklung, erkannte Jonathan die Möglichkeit, diese Skripte zu optimieren. Er ersetzte sie durch ein universelles Bewegungsskript, das sowohl horizontale als auch vertikale Bewegungen abdeckte. Dieses universelle Bewegungs Skript bot eine effizientere und vereinfachte Lösung für die Implementierung von beweglichen Plattformen im Spiel.

Insgesamt erwies sich das Skript für bewegliche Plattformen als wichtiger Bestandteil des Level-Designs und trug dazu bei, die Vielfalt und Herausforderung im Spiel zu erhöhen.



9.Über die Laufzeit des Projekts hinweg haben wir regelmäßige Meetings abgehalten, um den Fortschritt zu besprechen, große Änderungen zu erörtern, Feedback auszutauschen und die Arbeitsaufgaben aufzuteilen. Diese Treffen waren entscheidend für die Koordination und Zusammenarbeit im Team.

Während dieser Meetings haben wir nicht nur den aktuellen Stand des Projekts analysiert, sondern auch konstruktive Ratschläge und Tipps untereinander ausgetauscht.

Die Diskussionen während der Meetings waren oft von Offenheit, Kreativität und Teamgeist geprägt. Wir haben aktiv Ideen vorgeschlagen, Bedenken geäußert und gemeinsam nach optimalen Lösungen gesucht. Diese interaktiven Sessions dienten nicht nur dazu, den Projektfortschritt voranzutreiben, sondern auch das Team zu stärken und eine positive Arbeitsatmosphäre zu schaffen.

Insgesamt würde ich sagen, dass diese Meetings ein wesentlicher Bestandteil unseres Erfolgs waren und haben dazu beigetragen, dass wir effizienter arbeiten konnten, indem wir regelmäßig miteinander kommunizierten, uns unterstützten und voneinander lernten.

10.Ich war dafür zuständig, die Play Tests des Prototyps auszuwerten. Dabei habe ich mir die verschiedenen Videos angeschaut und verschiedene Aspekte bewertet, die in meine Schlussfolgerungen eingeflossen sind. Es wurde deutlich, dass es eine Diskrepanz in den spielerischen Fähigkeiten sowie in der Mentalität, der Aufmerksamkeit und dem Temperament der Tester gab. Zum Beispiel gab es einen Tester, der jedes einzelne Hindernis innerhalb der ersten beiden Versuche überwinden konnte, sogar die unveränderte Laser Stelle im Parkour-Level, bei der andere Tester sogar aufgaben. Natürlich ist es wichtig zu betonen, dass dieser Tester nicht als Norm angesehen werden kann. Die Vielfalt der Spieler hat jedoch sehr geholfen, bestimmte Stellen aufzuzeigen, die einer Veränderung bedurfen, um einer größeren Anzahl von Spielertypen gerecht zu werden.

Es gab jedoch auch Tester, die Kritik äußerten, die nicht besonders konstruktiv war und zugleich Begründungen enthielt, die schlichtweg unzutreffend waren. In solchen Fällen war es besonders

wichtig, zu erkennen, warum diese Kritik geäußert wurde, und zu bewerten, welche tiefer liegenden Aussagen man daraus ableiten kann, insbesondere im Hinblick auf das allgemeine Verhalten des Testers.

Überraschend war, dass die meisten Tester sehr schnell ungeduldig wurden und mehr Fehler machten, sobald sie an einer Stelle hängen blieben. Insgesamt waren die Play Tests von unschätzbarem Wert für die Weiterentwicklung unseres Spiels. Die Perspektive eines Spielers, der nicht an der Entwicklung und am Design beteiligt war, ermöglicht eine Sicht auf das Projekt, die zu vielen Verbesserungen führen kann.

12. Die Dokumentation dient dazu, gewisse Gedankengänge und Arbeitsvorgänge zu erklären, sie liefert einen Einblick dazu, was wir gemacht haben und vor allem warum. Das Verfassen der Texte innerhalb dieses Dokuments ist Hauptbestandteil meiner zur Dokumentation aufgewandten Zeit. Hierbei haben Jonathan, Florian und ich jeweils einen Teil des Dokuments verfasst.

ARBEITSPAKETE, DIE ZUSAMMEN ERLEDIGT WURDEN

Tod durch Laser von Prince und Florian

Der Tod durch Laser war für unseren PoC essentiell wichtig und demnach eine unserer Prioritäten zu Projektstart. Hierbei haben sich Florian und ich zusammen darangesetzt. Unsere Vision: Der Spieler stirbt sichtbar bei Zusammentreffen mit einem “tödlichen” Laser. Die roten Laser sollten zudem als “tödlich” gelten.

Da wir anfangs noch kein Character Modell und auch keine Animation hatten, wollten wir zumindest einen Vorgeschmack auf unsere Ideen hierfür liefern und beim Tod den Character in kleine Cubes “zerspringen” lassen. Dies sollte einen Ausblick auf unseren späteren Roboter, der selber bei Tod in Einzelteile zerspringt, bieten.

Die technische Umsetzung für den Tod des Spielers war nach einer kurzen Auseinandersetzung mit der Implementierung des Lasers durch Jonathan recht angenehm umzusetzen, der Laser schickt einen Raycast in die Richtung, in die er geschossen wird und hat entweder die Eigenschaft tödlich zu sein oder dem Spieler nicht zu schaden. Kollidiert ein tödlicher Laser mit dem Spieler, soll dieser den Spieler auslöschen. Da der Spieler mit einem Tag mit dem Namen “Player” ausgestattet war, konnten wir bei einer Kollision das Tag des getroffenen Objektes prüfen und feststellen, ob es sich dabei um die Spielerfigur handelt. Sollte dem so sein, wird eine Methode ausgeführt, die die Figur des Spielers ausschaltet und einzelne Teile des Roboters in zufällige Richtungen verschießt (in PoC wurden dafür mehrere kleine, weiße Blöcke verwendet, um das zu visualisieren). Im Laufe des Projektes wurde uns nahegelegt, dass ein automatisches Neustarten des Levels von Vorteil wäre, sobald der Spieler stirbt. Zur Umsetzung wird nach dem Sterben des Spielers eine Coroutine ausgeführt, die das Level nach drei Sekunden neu startet.

Schmelzen der Blöcke von Prince und Florian

Für unseren Proof of Concept wollten wir die Schmelz-Mechanik bereits implementiert haben, da diese explizit zu unseren Laser-Mechaniken gehört und essentiell für einige Level Designs werden sollte. Die Idee hierbei war, dass der Laser auf einen Schmelzblock trifft, dieser sich orange verfärbt (so als würde er glühen) und dabei schrumpft, um das Schmelzen zu simulieren. Hierzu haben wir zum einen das Melting Block-Prefab erstellt, welches als Standard für diese Blöcke genutzt wird. Es hat den Tag "meltable", welcher das Objekt als schmelzbar markiert. Daraufhin haben wir ein Melting-Controller-Skript geschrieben. Dieses Skript kontrolliert die Schmelz-Mechanik des Blocks. Wenn ein Objekt mit dem Tag "Meltable" kollidiert, wird der Block schrittweise kleiner, um das Schmelzen zu simulieren. Gleichzeitig wird seine Farbe auf Rot geändert, um anzuzeigen, dass er sich erhitzt. Wenn die Kollision endet, wird der Schmelzvorgang gestoppt und das Material wieder auf seinen normalen Zustand zurückgesetzt. Diese Implementierung ermöglicht es dem Spieler, die Auswirkungen des Lasers auf die Umgebung zu erleben und trägt somit zur Immersion des Spiels bei.

Ursprünglich gab es hierbei noch das Problem, dass wenn der Laser aus einem bestimmten Winkel auf den Block trifft, dieser gegebenenfalls gerade so weit schmelzen konnte, dass der Laser am kleineren Block vorbeiging. Da dies zu Problemen und Frust führen konnte, wurde später hinzugefügt, dass die Hitbox, die auf das Auftreffen des Lasers prüft, separat ist und ihre Größe behält. Dies vereinfacht den Umgang mit Schmelz Blöcken.

Character Movement von Prince und Florian

Das Character Movement war eines der ersten Dinge, die wir implementieren mussten. Dazu setzten sich Prince und ich zusammen. Zunächst platzierten für den PoC eine weiße Kapsel, die unseren Spielerfigur darstellen sollte und begannen damit, dass der Spieler nach links und rechts laufen können sollte. Jedes Skript kommt automatisch mit zwei Methoden: der *Update()*- und der *Start()*-Methode. Während die Start()-Methode einmal vor dem ersten Frame-Update aufgerufen wird und es uns erlaubte, eine Variable für dessen RigidBody-Komponente zu belegen, wird die Update()-Methode einmal pro Frame ausgeführt. In dieser würde jeden Frame geprüft werden, ob der Spieler die Spielfigur mit einer Taste gerade bewegen möchte. Für die horizontale Bewegung müssten wir also irgendwie Informationen über die Eingaben des Spielers bekommen. Hierbei liefert die *Input*-Klasse von Unity genau das, was wir brauchen. In den Projekteinstellungen lassen sich die Steuerungsmöglichkeiten mit einem Namen und Tasten belegen und anschließend über die Input-Klasse ansprechen. Da wir durch die Start()-Methode auch Zugriff auf die RigidBody-Komponente der Spielfigur hatten, konnten wir also aus dem gelesenen Input in der horizontalen Achse einen neuen Vektor für die *Velocity* des RigidBody setzen, die wir in der x-Achse mit einem festen Wert für die Geschwindigkeit des Spielers multiplizierten. Nachdem wir die horizontale Bewegung abgeschlossen hatten, wollten wir uns um die vertikale Bewegung kümmern, der Spieler sollte die Möglichkeit haben springen zu können. Hierbei war der Gedanke sehr ähnlich zu dem der horizontalen Bewegung. In der Update()-Methode wird eine Methode für das Springen aufgerufen, die prüft, ob der zum Springen passende Knopf auf der Tastatur gedrückt wurde und ob der Spieler sich aktuell auf dem Boden befindet, Sprünge in der Luft bzw. Doppelsprünge sollten nicht möglich sein. Mit der Referenz auf die RigidBody-Komponente der Spielerfigur konnten wir auf deren Velocity zugreifen und in vertikale Richtung eine Kraft auf diese einwirken lassen, die wir im Vorfeld festgelegt haben, in Kombination mit der horizontalen Bewegung ließe sich der Sprung sogar in eine beliebige Richtung steuern lassen.

Nachdem wir den Sprung implementiert hatten, fingen wir an, diesen zu testen und bemerkten sehr schnell ein Problem: Der Spieler konnte, wenn er von einer Kante einer Plattform runterlief, in der Luft springen. Das sollte in unserem Spiel nicht möglich sein, deshalb fingen wir an zu überlegen, was wir tun könnten, um dem entgegenzuwirken. Hier kam auch der Großteil meiner verbrachten Zeit zustande, es war viel Debugging notwendig. Jonathan legte uns die Idee nahe, dass wir überprüfen sollen, ob sich die Füße des Spielers auf dem Boden befinden oder nicht. Also verteilten wir an jedem Teil, dass den Boden darstellte das Tag “Ground” und an dem Spieler das Tag “Player”. Wir prüften mit hilfe von Methoden, ob der Collider des Spielers sich mit etwas überschnitt, das das Tag “Ground” zugewiesen hatte und versuchten somit das Problem zu lösen. Zu unserem Bedauern tat es das nicht, der Spieler konnte immer noch weiter in der Luft springen. Zu diesem Zeitpunkt haben wir uns dann

dazu entschieden, dass andere Dinge für den PoC wichtiger sind, wie zum Beispiel das Wechseln in ein anderes Level oder dass der Spieler stirbt, wenn er einen roten Laser berührt und wie das dargestellt werden könnte.

Im späteren Verlauf haben wir dann dieses Problem fixen können, indem wir statt einen Collider für den Groundcheck zu nutzen, stattdessen einen Raycast verwendeten. Dieser war weit zuverlässiger darin, unseren Ground State zu überprüfen und auch essentiell für korrekte Sprung-Animationen etc.

PLAYTESTING UND BALANCING

Anfangs erhielten wir zum PoC das Feedback, dass wir das Ziel des Levels weiter hervorheben sollten, damit der Spieler weiß, einen genaueren Pfad hat, um das Level zu beenden. Dies haben wir in der Hinsicht so gelöst, dass wir den Hintergrund des Ziels grün gefärbt haben und ein Notausgangs-Schild dort platziert haben.

In den Tests des MVP wurden überwiegend Bugs von den Spielern gefunden, wir haben beim Pausieren des Spiels vergessen den Time-Scale wieder auf 1 zurückzusetzen, wodurch ein Neustart des Levels, entweder durch das Drücken der R-Taste oder über den Button des Hauptmenüs, dazu führte, dass der Spieler das Spiel nicht weiterspielen konnte, ohne den Grund dafür zu kennen. Diesen Bug zu entfernen war recht simpel, jede Aktion, die damit zusammenhängt, die Szene neu starten zu lassen oder sie zu wechseln, musste den TimeScale zurücksetzen.

Ein weiterer Fehler, dessen Fund durchaus wichtig war, war, dass die Tür in Level_04 sich nicht richtig verhalten hat, wenn der damit verbundene Knopf schnell hintereinander gedrückt wurde. Das Debugging dafür benötigte eine etwas längere Zeit, da wir dafür den Animator der Tür etwas genauer untersuchen mussten. Wie sich herausstellte, lag es daran, dass die Idle-Animationen der Tür eine Exit-Time hatten, die dazu führten, dass die Animation nicht schnell genug erfolgte.

Was des Weiteren bemängelt wurde, war, dass der Spieler an Kisten und Wänden hängen bleiben konnte. Dieses Problem haben wir wahrgenommen und im Zuge des Projektes ebenfalls lösen können.

Viele Tester merkten zum MVP an, dass der Anteil an spielbaren Levels gering war. Wir haben bewusst die Entscheidung getroffen, dass wir, bevor wir weitere Level designen und zum Spielen bereitstellen, erstmal eine solide Grundlage dafür schaffen mussten. Dies beinhaltete zum Beispiel das Bereitstellen von dekorativen Panels, die zwar auch entweder weiß oder schwarz waren, aber dennoch oberflächlich anders aussahen.

Der letzte Kritikpunkt kam aus einem Test, der das Fehlen von jeglichem Sound bemängelte. Durch das Ausscheiden eines Gruppenmitglieds waren wir zwischen dem PoC und dem MVP nicht mehr dazu imstande, um bis zu den Tests Musik einzubauen, da uns auch die Kenntnisse über den Umgang mit Sound in Unity fehlten. Dennoch haben wir es zumindest geschafft, das Feedback teilweise zur Vorstellung des MVP umzusetzen und zumindest Musik für das Hauptmenü und innerhalb des Spiels einzubinden, auch wenn die Einstellungen über die im Haupt- und Pausemenü noch nicht vollständig funktionierten. Erst zwischen MVP und Prototypen gelang es uns, Musik und Spezialeffekte vollständig und komplett einstellbar im Projekt zu verwenden.

In den Prototype Tests waren vorwiegend Balancing und mechanische Anpassungen Hauptthema des Feedbacks. Level 6 insbesondere war zu schwer, die Spiegel im Spiegel Level drehten sich zu stark, wodurch man sehr präzise sein musste, es wurde teilweise nicht klar, was genau gefordert ist, um weiterzukommen. Zum Beispiel gab es Stellen, an denen man feststecken konnte und mit R neustarten sollte, diese Möglichkeit wurde aber innerhalb des Spiels nicht ganz aufgezeigt. Der Sprung von reinen Rätseln zu schwerem Parkour war den Testern teilweise auch zu abrupt. Zudem war das Field of View gerade im Parkour teilweise nicht groß genug, um einen Überblick zu bekommen, was um den Spieler herum passiert.

Aus dem Feedback der Tester entnahmen wir eine Vielzahl an Anpassungen und Änderungen um das Spielerlebnis zu verbessern:

1. Spiegel sollten weniger präzises drehen erfordern
2. A6 vereinfachen und Hinweis mit Pfeilen, um den Weg anzuzeigen
3. A6 kleinere Anpassungen für Schwierigkeit
4. ggf. "unnötige" Laser streichen oder Funktion geben
5. Hinweis auf das Neustarten, spätestens im Schmelz Level
6. FoV in A6 anpassen und an Stellen an denen es benötigt wird
7. Im ersten Parkour Level (das sehr einfache) Rätsel teil hinzu, um die Verknüpfung zwischen Parkour und Rätsel zu erleichtern

Diese hier als Ausschnitt aus den wesentlichen Key Points dazu.

Ansonsten wurden das Design, die Menüs, der Sound und das generelle Spielgefühl gelobt und das Feedback war im Großen und Ganzen sehr positiv.

Was das Balancing betrifft, so gab uns besonders der Prototype Playtest einen Ausblick auf die zu erwartenden Fähigkeiten verschiedener Spieler. Beim ursprünglichen Testen von Parkour Mechaniken wurde sehr viel mit dem “physikalisch” Möglichen des Charakters probiert, Gerade in A6 wurde eine Stelle sehr stark an gerade so im Bereich des Möglichen orientiert und stellte sich in den Tests als viel zu schwer heraus. Man muss dauerhaft bedenken, dass der normale Spieler weder zu perfektem Timing in der Lage ist noch Zugriff auf Dev-Respawn Optionen hat, um das Hindernis ständig zu wiederholen. Im Verlauf des Levels eine Stelle bei ca. 70% des gesamten Levels zu haben, die sehr schwer ist, ist kein gutes Balancing und frustriert die Spieler nur. Explizit, da diese ohne Checkpoint wieder am Anfang starten. Aus diesem Grund waren weitreichende Anpassungen notwendig, um das Bestehen des Levels zu vereinfachen. Ansonsten haben wir bei einigen Rätseln Hinweise mithilfe von „Decals“ an der Wand implementiert, diese sollen es den Spielern erleichtern weiter zu kommen und wurde explizit in den Playtests gewünscht.

Insgesamt haben wir darauf geachtet, eine gewisse Schwierigkeitskurve im Verlauf unserer Level zu beachten, so wird es progressiv komplexer und endet mit einem Finale, welches nochmal alle Mechaniken die wir haben kombiniert.

KOMPILIEREN

Zum Ausführen des Projektes wird mindestens Windows 10 benötigt. Zum Öffnen des Projektes in der Unity-Engine wird Unity in der Version 2022.3.9f1 benötigt.

EXTERNE ASSETS UND ANLEITUNGEN

- Logo und Hintergrund Hauptmenü MVP erstellt mit: https://looka.com/
- Sound-Effekt robo_death: https://mixkit.co/free-sound-effects/robot/?page=2 (Name: Robot System Break)
- Sound-Effekt jump: https://freesound.org/people/cabled_mess/sounds/350905/
- Sound-Effekt button-click: https://www.freesoundslibrary.com/button-click-sound-effect/#google_vignette
- Sound-Effekt melting-sound: https://soundbible.com/1090-Hot-Sizzling.html
- Sound-Effekt laser: https://www.101soundboards.com/sounds/25915-humming-lightsaber#google_vignette
- Sound-Effekt Tür: https://pixabay.com/sound-effects/search/spaceship-doors/ (name: Sci-Fi sliding door (height adjustable chair sounds) (Audio wurde auf die benötigte Länge geschnitten))
- Hintergrund-Musik und Level-Musik: https://assetstore.unity.com/packages/audio/ambient/sci-fi/free-sci-fi-and-cyberpunk-music-pack-264590
- Logarithmische Konversion von Audio-Slidern: https://johnleonardfrench.com/the-right-way-to-make-a-volume-slider-in-unity-using-logarithmic-conversion/
- Brackeys Audio-Manager: (54) Introduction to AUDIO in Unity - YouTube
- Roboter-Modell: https://quaternius.itch.io/lowpoly-robot
- Notausgang Schild: https://www.hilpress.com/en/signs/emergency-signs/standard-emergency-signs/hil-sign-escape-route-to-the-right-polyester-300-x-150-mm.html#&gid=1&pid=1
- Sprung Mechanik mit variabler Höhe:https://www.youtube.com/watch?v=7KiK0Aqtmzc
- Im PoC gezeigter Robo-Sentry (Im Laufe des Projekts verworfen) & Laser Textur: https://assetstore.unity.com/packages/3d/characters/robots/tiny-robots-pack-98930

BEWERTUNG

Florian Kern

Ich empfand das Projekt als äußerst positiv. Es war sehr schön, dass wir eine komplett eigene Idee ausgearbeitet haben und diese umsetzen konnten. Es war toll zu sehen, wie sich das Projekt stets weiterentwickelt hat und die Mechaniken funktionierten. Was sehr schade war, ist, dass uns ein Gruppenmitglied verloren ging und wir aufgrund dessen nochmal neu planen mussten, wie wir damit umgehen. Ich habe tatsächlich keine Kritik, die ich an dem Projekt äußern könnte, die Zusammenarbeit in meiner Gruppe empfand ich als hervorragend, da wir uns stets geholfen haben, wenn Probleme aufgekommen sind. Ich habe auch keine Kritik, die ich an die Leitung äußern könnte, wenn von dieser Hilfe benötigt worden war, hat man diese auch bekommen. Das Einzige, was man vielleicht machen könnte und wichtig für die Tests wäre, ist, dass die Teilnehmer dazu angehalten werden, die Kommunikationskanäle, die die FH-Wedel bietet, regelmäßig auf Nachrichten zu überprüfen, da manche Teilnehmer nicht einfach oder nur über andere Mitglieder aus deren Gruppe zu erreichen waren.

Jonathan El Jusup

Ich konnte in diesem Projekt viele wertvolle Erfahrungen sammeln und kann mit gutem Gewissen das Projekt abschließen mit der Sicherheit, die Idee umgesetzt zu haben, die wir uns vor einem halben Jahr vorgenommen haben. Es war schön zu sehen, wie ein komplettes Projekt von Grund auf geplant und entwickelt wurde. Ich persönlich war auch froh, Arbeit anderen überlassen zu können mit dem Vertrauen, dass etwas Schönes daraus wird. Doch muss ich anmerken, dass der Anfang leider nicht so lief, wie wir es uns erhofft hatten. Unsere initiale Spielidee wurde stark kritisiert und wir mussten uns demnach extrem viel Mühe geben, um unserer Idee gerecht zu werden. Dies konnten wir dann zum Proof of Concept aber auch richtig beweisen. Dies hat uns den nötigen Schub gegeben, weiter zu machen und unseren Spaß in dem Projekt wiederzufinden. Gerade sehr lange an bestimmten Funktionen zu arbeiten, die einen frustrieren, aber zu sehen, wie gut sie am Ende im Gesamtkontext funktionieren, war sehr belohnend. Insbesondere die Komplexität, die dabei entstand. Doch mussten wir leider auch damit klarkommen, dass wir ein Gruppenmitglied bis zum MVP verlieren würden. Dies hat unsere Pläne ins Rütteln gebracht und wir mussten uns auf die wesentlichen Inhalte fokussieren und leider bei der Narrative kürzen, was ich sehr schade fand. Letzten Endes kam dennoch ein Endprodukt raus, mit dem wir alle zufrieden waren, auch wenn wir nicht alles umsetzen konnten, was wir ursprünglich geplant hatten.

Prince Lare-Lantone

Die Arbeit an diesem Projekt hat mir insgesamt sehr viel Spaß gemacht. Vor allem durch die gute Zusammenarbeit mit Flo und Jonathan hatten wir ein sehr angenehmes Arbeitsklima und keinerlei Unstimmigkeiten. Neue Ideen wurden mit offenen Armen empfangen und konstruktiv diskutiert Jede einzelne Meinung hatte Gewicht und Entscheidungen wurden zusammengetroffen.

Die Entwicklung von einer Idee bis hin zu einem alleinstehenden Spiel war sehr interessant und den Prozess der einzelnen Phasen zu sehen hat viel Spaß gemacht. Ich war tatsächlich vom Zeitaufwand

bestimmter Teile des Projektes überrascht. Gerade in Bezug auf das Level Design, Modellieren und Testen. Ich erinnere mich an einen Tag an dem ich “nur noch kurz” ein paar Kleinigkeiten ändern wollte und dann 10 Stunden am Stück an dem Level gearbeitet habe, da mir auffiel, dass einiges doch nicht so funktionierte wie ursprünglich geplant.

Insgesamt bin ich persönlich sehr stolz auf das, was wir hervorgebracht haben, denn leider haben wir zum Anfang des Projektes ein Gruppenmitglied verloren. Hierdurch mussten wir die Arbeitspakete neu verteilen und einige Hürden mehr zu dritt überwinden. Trotzdem kann ich sagen, dass wir unsere Vision ohne große Einschnitte verwirklichen konnten.

Insgesamt war ich mit unserer Zusammenarbeit vollkommen zufrieden. Auch an der Leitung der Übung habe ich nichts auszusetzen und finde, dass dies durchaus ein sehr gelungenes Modul ist. Das Feedback zu den verschiedenen Phasen war überaus Hilfreich und konstruktiv und man hat immer die Hilfe bekommen, die man benötigte.