Hochschule der Medien Stuttgart

Studiengang Medieninformatik

Projektdokumentation
Projektbetreuer: Prof. Dr. Jens-Uwe Hahn
Sommersemester 2021

JEDI PLAYGROUND

Praktikum Virtual Reality

Mitwirkende:

Ben Häußermann
Lea Dannecker
Lukas Kleinau
Liane Genheimer
Max Herkenhoff
Maximilian Dolbaum
Martin Frantal
Sarah Mauff

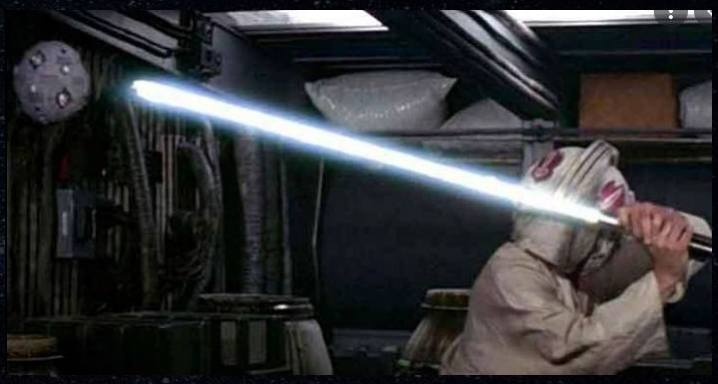
Inhaltsverzeichnis

Einführung

Unser Projekt, „Jedi Playground“, ist ein “Virtual Reality- Sandbox Spiel”, das im Rahmen der Vorlesung: „Praktikum Virtual Reality“ im Sommersemester 2021 entstanden ist. Unser Team bestand dabei aus 8 Medieninformatik-Studenten, alle derzeit in ihrem 3. Semester.
Ziel des Projektes war es, ein Spiel zu entwickeln, das in einem, der Star Wars Filmreihe nachgeahmten, Universum angesiedelt ist und dem Spieler die Fähigkeiten eines Jedi verleiht.

Wir entschieden uns für dieses Projekt da uns ein “Sandbox Spiel” freie Möglichkeiten gibt mit allen Features der VR-Headsets zu experimentieren. Auch wäre diese Art von Spiel nur schwer in einer klassischen Desktop Umgebung umsetzbar gewesen da große Teile des Spielspasses von der intuitiven Art sein Lichtschwert zu schwingen oder Objekte schweben zu lassen stammen. Sicherlich hatte auch die große Anzahl Star Wars Fans unter uns etwas damit zu tun.

Jedi benutzt die Macht um zu essen



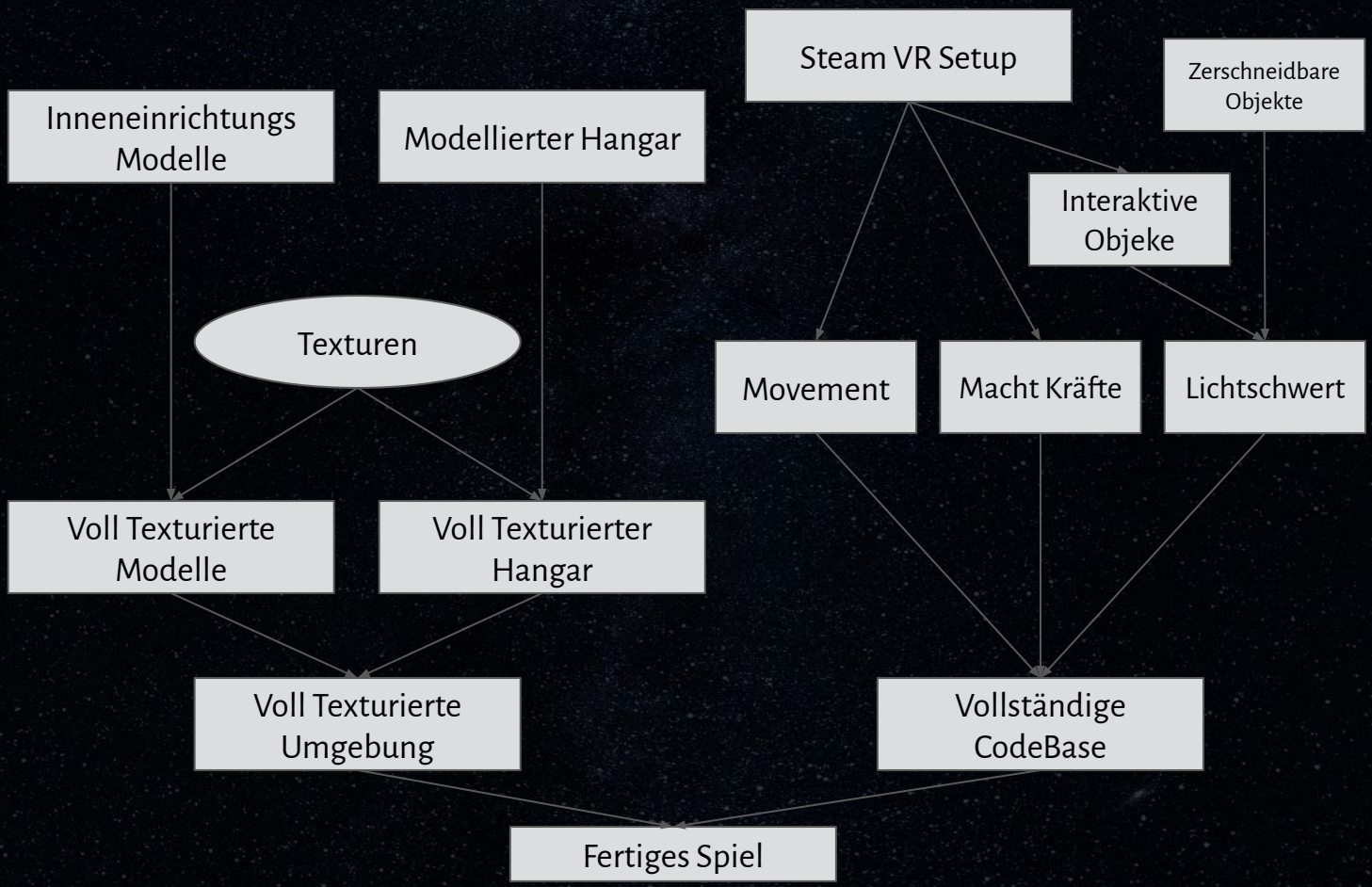
Jedi trainiert

Erstellen eines Minimum Viable Products

Nach der Ideenfindung folgte eine Planungsphase in der wir das Minimum Viable Product, kurz MVP, definieren mussten.

MVP ausdefiniert:
Freies Movement entweder über ein Teleport System oder Touchpad des Joysticks.
Ein voll begehbarer Raumschiff Hangar mit interaktivem Lichtschwert. Mithilfe der Macht sollte es möglich sein diverse Entfernte Objekte aufzuheben.

MVP als Produktflussdiagramm:



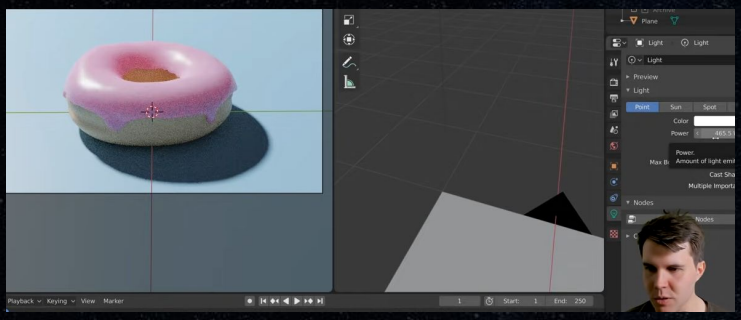
Verlauf des Projektes

Einarbeitung

Da nur die wenigsten Mitglieder unseres Teams bereits vor diesem Semester mit relevanten Programmen zu tun hatten planten wir mehrere Wochen ein, in denen jeder sich mit Blender, Unity und SteamVR vertraut machte. Da wir erst nach Erhalt der VR-Brillen mit der eigentlichen Arbeit am Projekt beginnen konnten war das auch die produktivste Art diese ersten Wochen zu nutzen.

Allgemeine Projektphase

Sobald wir in Besitz der VR-Brillen gelangten begann die eigentliche Arbeit. Nach dem Aufsetzen eines funktionierend Unity/SteamVR Projekts wurden Aufgaben aus dem MVP verteilt, Deadlines gesetzt und wir begannen und jeden zweiten Montag mit unserer Projektaufsicht und jeden Mittwoch untereinander zu treffen um zu überprüfen ob Deadlines eingehalten wurden und verschiedene Teilaufgaben miteinander zu mergen.



Einarbeitung in Blender mit dem bekannten Donut Tutorial

Aufgabenverteilung

Ein wichtiger Aspekt dieses Projekts war korrekte Verteilung der Aufgaben. Teilprodukte die auf dem kritischen Pfad lagen musste richtig zugewiesen werden und innerhalb der Deadline erfüllt werden sonst würde wir größere Zeit-Ziele verfehlen. Wir entschieden uns unser Team in zwei Hälften zu teilen von denen eine sich um den coding Teil und die andere um Visuelles und Auditorisches kümmerte. Innerhalb dieser beiden Teams wurde Aufgaben dann erneut genauer aufgeteilt. Hier die genaue Aufteilung in Unterbereiche.

Animation - Maximilian Dolbaum / Sarah Mauff
Coden - Max Herkenhoff / Maximilian Dolbaum / Lukas Kleinau / Martin Frantal
Texturieren - Ben Häußermann/ Lea Dannecker / Liane Genheimer
Modeln - Ben Häußermann / Lea Dannecker
Sound - Martin Frantal
User Interface - Lukas Kleinau
Gruppenleitung - Alle haben sich unter der moderativen Leitung von Max Herkenhoff mit eingebracht und so für eine angenehme Gruppenatmosphäre gesorgt. Alle waren gleichberechtigt Kritik und neue Ideen einzubringen.

Animation

Es wurde mit dem Unity -internen Animationssystem animiert, welches per se nicht kompliziert ist. Anfängliche Schwierigkeiten ergaben sich durch Komplikationen mit verschiedenen VR-Input Sets. Glücklicherweise gibt es eine große Menge an Videos. Durch die Vielzahl an Tutorials kam teilweise sogar Verwirrung auf welches die optimale Methode für unser Projekt war welche schlussendlich allerdings dem Verständnis und der Internalisierung ungemein geholfen hat.
Sobald man den Ablauf des Animierens verstanden hat, geht tatsächlich alles sehr schnell. Das zweite und auch letzte kleine Problem kam dann während des Animierens auf. Man muss sehr darauf achten mit welchen Werkzeugen man etwas ändert. Es passierte, dass bei Erstellung der Lauf-Animation von R2-A2 alles sehr gut aussah und keine erkennbaren Fehler unterliefen. Spielte man allerdings dann die Animation zum Testen ab, bewegten sich die Beine plötzlich durch den Körper des Roboters hindurch, was definitiv witzig aussah doch später ein wenig Verzweiflung aufkommen ließ, bis es zu einem Werkzeugwechsel kam.

Folgende Animationen waren dann binnen weniger Minuten erstellt. Es begann sogar, richtig Spaß zu machen, da Unitys Animationssystem prinzipiell sehr einfach bedienbar ist. Für zukünftige Projekte sehr zu empfehlen.

Durch das Arbeiten mit Transformationen via Script und Animationen mit Hilfe des Animation Editor von Unity, stellt sich die Frage, welche Lösungsstrategie die bessere ist. So haben wir ein Script erstellt, welches Objekte um eine Achse drehen lässt. Das war durchaus schneller an jedes Objekt zu binden, als mehrere Animator-Controller und Animationen zu erstellen.

R2 In Bewegung

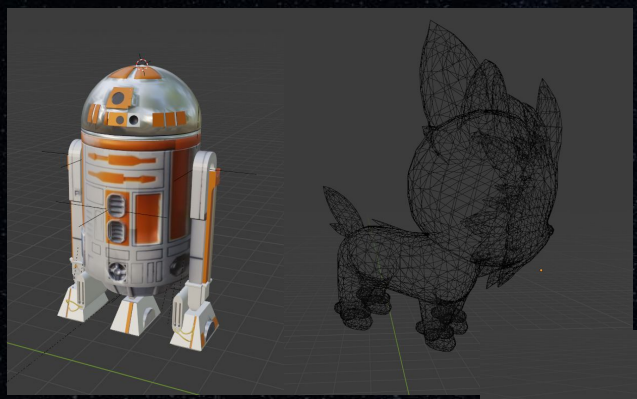
3D Modelling

Blender

Für die 3D-Modelle in unserem Game nutzen wir das Programm Blender, da es eine freie Software ist und sich auch schon ein Gruppenmitglied, welches für die Modelle zuständig war, mit dem Programm zuvor auseinandergesetzt hatte. Ein wichtiger Punkt, welcher vorab bei den Modellen geklärt werden musste, war der Stil der benutzt werden sollte. Der Stil auf den sich geeinigt wurde, ist Low- bis Mid-Poly mit realistischen/Foto Texturen, um eine realistische Umgebung für den Spieler zu schaffen und Rechenleistung zu sparen. Dadurch dass mehrere Leute an den Modellen saßen, war eine laufende Kommunikation wichtig, da trotz Richtlinien und Stil Entscheidungen jeder nochmal seinen eigenen Stil mit einbringt.

Unity ProBuilder

Auch zu erwähnen ist hier der Unity Probuilder. Ein Unity Addon mit dessen Hilfe man innerhalb von Unity selbst Models erstellen kann. Das ist vor allem hilfreich für größere oder modulare Assets, die man oft während des Design-Prozesses erneut anpassen muss. In unserem Projekt wurden die Wände des Hangars, sowie das begehbare Hangar-Kontrollzentrum mit dem Unity Probuilder erstellt um uns etwaige Arbeit bei häufigen Veränderungen zu ersparen.



Erste Modelle für das Spiel

Die Hauptattraktion des Hangars, der X-Wing

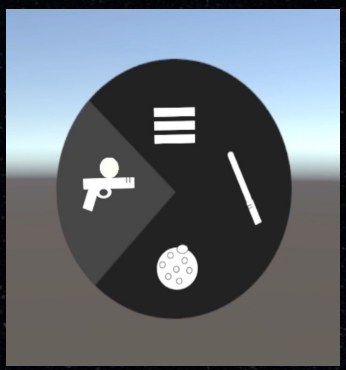
Sounddesign

Bei einer VR Anwendung steht das Nutzererlebnis im Vordergrund. Das finale Produkt muss dem Nutzer das Gefühl vermitteln, sich tatsächlich in der Umgebung zu befinden, die um ihn herum dargestellt wird. Dies führt zu einer tiefen Immersion des Spielers in die Spielwelt und vermittelt das Gefühl einer tatsächlichen realen Umgebung.
Um das soundtechnisch optimal zu gestalten, sind alle Soundeffekte in der Unity Engine als Audio Sources an das jeweilige Objekt gebunden zu dem sie gehören und werden von diesem aus abgespielt.
Die Unity3d Engine nimmt dem Entwickler hier glücklicherweise viel Arbeit ab, es ist sehr einfach einzustellen wie, wo und in welcher Intensität der Sound Effekt abgespielt werden soll. Was jedoch selbst entwickelt werden muss, ist der Zeitpunkt, an dem ein jeweiliger Effekt abgespielt werden soll.
Bei einfachen Gegenständen wie Kisten und Tonnen wurde dies durch eine einfache Kollisionsüberprüfung umgesetzt. Sobald der Gegenstand oder das Objekt mit einem Collider (wie beispielsweise dem Boden, anderen Objekten oder dem Spieler) kollidiert, wird ein entsprechendes Geräusch abgespielt. Zuträglich zu der Immersion des Spielers ist hier, dass nicht nur ein Soundeffekt zur Verfügung steht, sondern aus einer Reihe ähnlicher, aber sich doch unterscheidender Effekte zufällig einer ausgewählt und abgespielt wird.
Etwas komplizierter verhält es sich mit Effekten wie dem des Lichtschwertes, des Blasters, des Garagentors oder anderen interaktiven oder reaktiven objekten. Hier musste tief in die Skripte und die Funktionen des jeweiligen Spielobjektes eingetaucht werden, um an der richtigen Stelle den entsprechenden Befehl zum Abspielens des Soundeffekts einzufügen.
Es gibt auch interessante Lösungsansätze wie zum Beispiel, dass die Quelle des “Abfeuern” Sounds der Waffe nicht etwa die Waffe selbst, sondern der abgefeuerte Plasmastrahl ist. Zum einen erleichtert das die Wiederverwendbarkeit, denn der Strahl wurde auch für die Trainingsdrohnen verwendet, zum anderen kommt es dadurch auch zu einer viel stärkeren Tiefenwahrnehmung auf Seiten des Spielers, denn der Strahl fliegt mit seinem Sound durch den großen Hangar.
Alle Sounds sind lizenzfrei und können kostenlos und ohne namentliche Erwähnung des Erzeugers genutzt werden. Außerdem wurden Soundeffekte modifiziert oder hergestellt mit Hilfe der Software Audacity.

UI

Ein vollwertiges Videospiel benötigt selbstverständlich ein intuitives User Interface, über das der Spieler die Kernfunktionen der Software erreichen und steuern kann.
Doch bei der Implementierung des UI in einen VR-Titel muss man etwas umdenken.
Der Nutzer hat andere, umfangreichere Eingabegeräte als die klassische Computermaus zur Verfügung, die Bewegung und Darstellung erfolgt nicht mehr auf zweidimensionaler Ebene. Eine zusätzliche Dimension sowie haptisches Feedback, gekoppelt mit dem Mittendrin-Gefühl unter der VR-Brille heben das Spielmenü auf eine neue Ebene.
Statt der bloßen Navigation steht die erlebbare Interaktion im Vordergrund,
Menüs schweben im Raum, sind greifbar und stellen Eingabemöglichkeiten
mit Bezug zur echten Welt dar.

Zu Beginn der Entwicklung gab es die Bestrebung, ein klassisches Spielmenü mit Menüpunkten im Raum schweben zu lassen. Der Spieler sollte mit seinem Controller auf die gewünschten Einträge zeigen und diese so auswählen können. Schnell stellte sich heraus, dass diese Art der Navigation doch sehr umständlich sein würde und dem Spielerlebnis, insbesondere der Immersion, entgegenwirken würde.
Die technische Umsetzung der Unity UI-Module in Verbindung mit SteamVR stellte sich zudem als problematisch dar. So wurden ausgesendete Raycasts des Pointers nicht von den UI-Elementen erkannt und lösten demnach nicht die erwarteten Events aus.
Mit fortschreitender Entwicklung stellte sich heraus, dass für unser Sandbox-Game kein umfangreiches Menü vonnöten war, so wurde die Implementierung dieses Features nicht weiter verfolgt.



Verschiedene konzeptionelle Interface Entwürfe

UI (Fortsetzung)

Als weiteres grafisches Element war ein kleines Radialmenü angedacht. Ein kleiner Kreis, im Aussehen einem Glücksrad ähnlich, sollte bei Bedarf an der rechten Hand des Spielers erscheinen. Das Menü sollte eine Switch-Funktion für verschiedene Fähigkeiten des Spielers bereitstellen. Per Joystick konnten einzelne Segmente, für die jeweils eine Funktion hinterlegt war, angesteuert werden. Da im fertigen „Jedi Playground“ nur zwischen zwei verschiedenen Fähigkeiten gewählt wird, ist das Menü aus Gründen der Bedienerfreundlichkeit deaktiviert.

Zentrale Elemente des User Interface‘, die ihren Weg tatsächlich in die fertige Version des Spiels fanden, sind die beweglichen Buttons. Hier waren wir sofort von der Bedienbarkeit und dem Zugewinn an Immersion begeistert, so dass die Implementierung als obligatorisch galt. Wir steuern mit diesen Buttons z. B die Garagentore oder die Auswahl am Schießstand. Die Buttons selbst sind einfache Kuben, ausgestattet mit Collidern und einer Bewegungssperre in jeweils 2 Achsen. Drückt die Hand des Spielers auf einen dieser Buttons, reagiert dieser entsprechend bzw. kehrt danach in den Ursprungszustand zurück. Ein Button gilt als gedrückt, nachdem er um eine gewisse Distanz im Raum „verschoben“ wurde.
Neben den Buttons fügten wir einfache Schieberegler hinzu, um zum Beispiel die Farbe der Flüssigkeit im Nebenraum des Hangars bestimmen zu können. Der Griff des Reglers läuft auf einer unsichtbaren Spur, dessen lineare Werte beim Bewegen ausgegeben werden.
Zusätzlich zum Schieberegler haben wir ein bewegliches Ventilrad implementiert. Hierzu wurde es dem Spieler ermöglicht, das Steuerrad beim Greifen um die eigene Achse zu drehen. Mit Hilfe der dabei ausgegebenen Rotationswerte konnten wir so den Füllstand des Tanks beeinflussen.
Als abschließende Interaktionsmöglichkeit nutzen wir einen Hebel für den Lichtschwerautomaten. Hierbei handelt es sich um einen Griff, der mit Hilfe eines Hinge Joints um einen Ankerpunkt rotiert und bei Betätigung über einen Collider das zugehörige Event auslöst. Das eigentlich für die Nutzung an Türen gedachte Scharnier kann beliebige Öffnungswinkel einnehmen und dabei auch eingegrenzt werden, weshalb es hervorragend für unseren Anwendungsfall geeignet war.

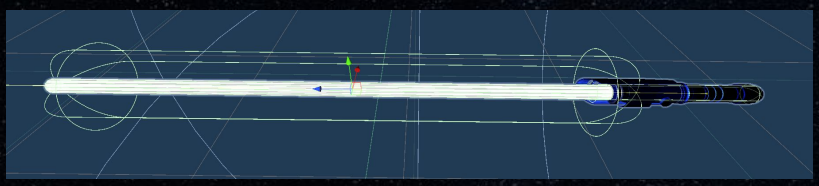
CODING

Scripting R2-A2

R2-A2's Movement ist tatsächlich relativ einfach gehalten. Hauptsächlich wird mit zufälligen Wartezeiten vor jeder Aktion und Collision-Detection gearbeitet, sodass seine nächste Aktion bei Kollision mit Objekten immer eine Rotation ist. Andernfalls könnte es passieren, dass er zwei oder drei mal versucht geradeaus zu laufen während er beispielsweise permanent mit einer Wand kollidiert. Das sähe ohne Frage sehr unnatürlich aus. Des Weiteren wurde darauf geachtet, dass auch die Rotation an sich natürlich aussieht. Es wird zunächst ein zufälliger Wert als Rotationsgrad gesetzt. Wichtig war hierbei nun, dass sich R2-A2 bei Werten, die größer als 180 Grad sind, auf der Achse entgegengesetzt dreht - und zwar um 360 - <zufälliger Wert> Grad, damit die Zielrichtung effektiv gleich bleibt. Sprich, er sich automatisch nach links oder rechts dreht und nicht nur nach rechts bis der Winkel erreicht ist.
Überraschenderweise ergaben sich kaum Fehler oder Probleme während der Implementation, was so auch sehr willkommen war. Zuletzt ist noch zu sagen, dass die Verknüpfung des Skriptes mit den jeweiligen Animationen der Zustände ebenfalls leicht fiel.

Scripting LightSaber - Ein/Ausfahren

Eine maximal Länge des Schwertes wird vom Developer festgelegt. Wird das Event zum ein/ausfahren triggered, so wird mit Hilfe des FixedUpdates und der Geschwindigkeit des ein/ausfahren pro FixedUpdate eine delta-Distanz berechnet, um welche das Schwert sich ein/ausfahren soll und so der Y-Wert (Höhe) des Schwertes angepasst.



Ausgefahrenes Lichtschwert in der Unity Prefab Ansicht

CODING

Scripting LightSaber - Mesh Slicer

Der Mesh slicer war eine große Herausforderung und hat immer mal wieder zu Problemen geführt. Letztendlich ist das System sehr simpel:
Nimm einen Punkt des Schwertes (Base) des Schwertes beim Kollidieren mit einem <Sliceable> Objekt.
Beim Verlassen des Colliders werden 2 weitere Punkte genommen, Base vom Schwert und Spitze vom Schwert. Mit Hilfe dieser 3 Punkte wird eine Schnittebene erstellt. Daraufhin wird das Mesh des <Sliceable> Objektes genommen und alle Mesh-Triangles in ein Array gesteckt für jedes Mesh-Triangle wird dann berechnet auf welcher Seite der Ebene es sich befindet und für diese Seite dann abgespeichert. Wenn ein Mesh-Triangle von der Ebene geschnitten wird, werden neue Triangles mit Hilfe der 2 Schnittpunkte erzeugt. und den jeweiligen Seiten zugeordnet.

Zuletzt bekommen alle Triangles die Meshdaten des <Sliceable> Objektes und die Schnittfläche wird mit Triangles zum Mittelpunkt gefüllt um eine gerade schnittfläche dar zu stellen.

Auf dem Lichtschwert kann man einstellen wie viel “Force” auf die geschnittenen Teile angewandt werden soll.
Auf den <Sliceable> Objekten können ein paar Spezifikationen wie z.B. ob das Objekt Solide ist, oder es durch Schwerkraft beeinflusst wird, gesetzt werden.

Scripting Blaster - Shooting

Hierbei wird das “Bullet” Prefab mit Hilfe der Ausrichtung der Waffe Instanziiert.

CODING

Scripting Blaster - Bullet

Die Kugel bekommt eine “Force” in Schussrichtung und lebt bis zu 3 Sekunden es sei denn sie kollidiert mit verschiedenen Objekten.
Da durch Bewegung die Hand und die Waffe mit der instanziierten Kugel kollidieren können, sind diese aus dem Collision-Layer der Kugel entnommen.
Kollidiert die Kugel mit einem normalen Objekt, so wird dort ein “Bullet-Hole” Prefab instanziiert.
Kollidiert sie mit einer Aktiven Trainings-Sphere so wird diese als getroffen markiert und handelt dementsprechend.
Bei Kollision mit <Reflecting> Objekten wird die Force der Kugel umgekehrt und so mit reflektiert.

Scripting Shooting-Range - “AimLab”

Basierend auf dem Programm “AimLab” werden beim Starten 3 Kugeln an randomPunkten im Schussbereich gespawnt. Kollidieren diese mit einer Kugel so wird eine dreh- und skalierungs- Animation ausgeführt, bei der, sobald das Objekt auf 0 skaliert ist ,die position ändert und dort mit gleicher Animation wieder auftaucht.

Scripting Shooting-Range - Reflecting

Hierbei fungiert eine einzelne Trainings-Sphere als Gegner und bewegt sich zu random Positionen um von dort den Spieler in Richtung einer random position auf dem Player-Collider abzuschießen.
Das ganze ist auf 20 Sekunden begrenzt, damit man nicht während den Schüssen den Ausschalt-Knopf drücken muss.

Scripting Space-stuff

Mit Hilfe eines simplen Transform.Rotate(rotationValue) Aufruf pro FixedUpdate können Objekte um das Schiff herumfliegen.

CODING

Scripting Mülleimer/Ofen

Der Mülleimer nutzt als einziges Objekt im Spiel das ParticleSystem. Er hat einen Collider auf dem Boden, wenn ein anderer Collider mit diesem Kollidiert wird der dazugehörige Rigidbody gesucht und das ganze Objekt dann bei Erzeugung einer Explosions-Animation zerstört.

Scripting Force

Die gesamte Funktionalität der Macht bzw. “Force” wird von dem GameObject “ForcepullManager” gehandhabt. Hierbei hat der Spieler 2 Modi zur Verfügung. Während der Forcegrab immer zur Verfügung steht, so kann der Spieler die Hauptfunktion der Hand zwischen ForcePush (Machtmodus) und ForcePull (Waffenmodus) wechseln.

Scripting Force - Forcepush

Die Grundlagen des Forcepushes sind so simpel wie effektiv: So wird beim Auslösen des Forcepushes eine transparente Kugel vor dem Spieler basierend auf einem Prefab instanziiert. Diese Kugel hat einen Rigidbody, sowie einen Collider, schiebt somit alle Objekte vor dem Spieler bzw. in Zielrichtung zur Seite. Um ungewollte Seiteneffekte zu vermeiden wird Kollision zwischen dem Forcepush-prefab und Wänden ignoriert. Bei der Instanziierung der Forcekugel wird eine Lebensdauer definiert, nachdem diese wieder despawnt.

CODING

Während dieser Ansatz sich als sehr effektiv erwies, so zeigte sich doch ein klares Problem: Das Zielen. Während der gewählte Ansatz anfangs war, basierend auf den aktuellen Handrotationswerten die Richtung der Kugel zu berechnen, so zeigte sich schnell, dass dieser Ansatz unnötig kompliziert, schwer zu debuggen und unintuitiv anpassbar war. Schließlich kamen wir auf eine einfachere Lösung und erweiterten die Hand um einen “Forcetracker”. Dieser besteht aus zwei Punktem, einem Start- und einem Endpunkt. Ersterer ist direkt in der Handfläche des Spielers platziert und letzterer schwebt ungefähr über dem Handballen. Die Orientierung der Spielerhand und die Richtung, in die der Ball fliegen soll, lassen sich so ganz einfach durch Subtraktion der globalen Koordinaten beider Punkte errechnen. Dieser Ansatz brachte zudem den Vorteil mit sich, dass es einfacher war Feinjustierungen in dem Winkel zu machen indem die Forcekugel relativ zur Hand instanziiert wird. Zudem war der Forcetracker dann auch beim Forcegrab einsetzbar und somit wiederverwendbar.

Scripting Force - Forcegrab

Der Forcegrab fundiert wie bereits erwähnt ebenfalls auf der Orientierung der Hand, die basierend auf dem Forcetracker erkannt wird. So wird von der Hand aus in die Richtung des Forcetrackers ein Ray gecastet, welcher bei Collision einen Punkt bzw. ein GameObject zurückgibt. Daraufhin wird geprüft, ob das Gegenüber einen Rigidbodycomponent hat. Sollte dies nicht der Fall sein, so wird der LineRenderer des ForcePullManagers aktiviert und von der Position der Hand zu dem Punkt gezogen an dem der Ray ein Object getroffen hat. Ziel dieser visuellen Repräsentation des Rays ist es, dem Spieler zu zeigen wo er “hinzielt” und feinjustierung so zu verinfachen.
Sollte das getroffene Objekt aber einen Rigidbody haben, so wird die Schwerkraft für das Objekt ausgeschaltet und das Objekt, sowie die aktuelle Distanz zwischen Hand und Objekt gespeichert. Von diesem Punkt an wird jeden Tick basierend auf der Forcetrackerorientierung und der ursprünglichen Distanz zur Hand ein neuer Punkt x berechnet. Daraufhin wird anhand der aktuellen Position des Objektes y die neue Geschwindigkeit des Objekts berechnet (Geschwindigkeit = x-y).
Sobald der Forcegrabknopf losgelassen wird, wird die Gravitation für das Objekt wieder angeschaltet und die Berechnung der Geschwindigkeit beendet. Zudem wird der Linerenderer wieder unsichtbar, sofern er aktiv war.

CODING

Scripting Force - Forcepull

Der Forcepull interagiert direkt mit dem WeaponManager des Lichtschwertes. So wird die letzte Waffe gespeichert, die der Spieler in der Hand gehalten hat. Sollte nun also der Forcepull aktiviert werden, so wird dem Lichtschwert eine Geschwindigkeit in Richtung der Hand des Spielers (mit der die Eingabe getätigt wurde) gegeben. Natürlich wird der Forcepull nicht aktiviert, wenn der Spieler ein anderes Objekt in der Hand hält. Sobald die Waffe nah genug an der Hand des Spielers ist, so wird die Steamvr AttachObject Methode aufgerufen. Wichtig hierfür ist, dass der Input für Forcepull und Greifen der Gleiche ist.

Scripting ForceField

Auch wenn der Name es vermuten lässt, so besteht kein Zusammenhang zwischen den Machtfähigkeiten und dem Forcefield.
Das Forcefield beschreibt die Barriere, die das Spielfeld für den Spieler und R2 (aus eigener Kraft) begrenzt, für andere Objekte hingegen durchlässig ist. Das Prefab für diese Begrenzung unterteilt sich in 3 Komponenten: eine visuelle Repräsentation ohne Collider und 2 Boxcollider (als Trigger). Verlässt ein Objekt mit einem Rigidbody einen Collider so wird die Gravitation für das Objekt basierend auf dem Collider an oder ausgeschaltet. Sollte allerdings der Spieler mit einem Forcefieltrigger kollidieren, so wird er davon abgehalten sich weiter in Richtung des Feldes zu bewegen. Dies ist zwar komplizierter als ein simples Collisionlayerprinzip, schien uns aber skalierbarer.

Scripting FluidShader Interaktion

Um die Tankfüllung, sowie Farbe zu ändern bedarf es einer Skriptinteraktion mit dem Liquidshader. Sollten die entsprechenden Werte (RGB Wert gegen von den Slinder oder Rotationswinkel des Pipewheels) sich geändert haben, so werden die “exposed Properties” des Shaders, basierend auf ihren Kennungen, angepasst.

CODING

Scripting Aufzug

Der Aufzug stellte sich als problematisch heraus, da er sowohl mit als auch ohne Rigidbody dazu führte, dass das Sichtfeld des Spielers, sowie dessen Hände anfingen zu zittern. Dieses Problem lösten wir, indem wir die Gravitation des Spielers gleich der Bewegungsgeschwindigkeit des Aufzugs setzten. Somit schwebt der Spieler nun nach oben bzw. unten und ist dem Aufzug minimal voraus. Während dieser Ansatz ursprünglich als Übergangslösung gedacht war, zeigte sich, dass er die Zitterprobleme der Spielerkamera (welche zu Übelkeit führten) derart effektiv löste, dass wir uns entschieden dies als permanente Lösung zu optimieren und implementieren.
Damit der Spieler aber nicht durch einen Zufall (wie ein Rigidbodyobjekt kommt auf den Schalter) plötzlich mit veränderter oder gar negativer Gravitation kämpfen muss, hat der Aufzug einen Triggercollider, anhand dessen bestimmt wird ob der Spieler im Aufzug ist oder nicht.
Das Aufzugskript in sich bewegt den Aufzug zwischen einem gesetzen Maximal- und Minimalpunkt in Höhe hin und her und stoppt, sobald einer der beiden erreicht ist.

Scirpting Lichtschwertautomat

Der Lichtschwertautomat setzt sich aus 3 separaten Collidern zusammen und wird im Zentralem vom LightSaberColorChanger-Skript gesteuert. Die Triggercollider für Lichtschwert und Farbbehälter funktionieren ähnlich. Sobald das gesuchte Objekt den OnTriggerEnter Event aufruft, wird das Objekt kinematisch und in die entsprechende Position innerhalb des Automaten gesetzt. Natürlich wird diese Transformation des Objekts verzögert, sollte der Spieler dieses noch in der Hand halten.
Wird der Hebel bewegt und tritt in den LeverCollider ein, so wird der Automat aktiviert. Dieser überprüft, ob sowohl Farbcontainer, als auch Lichtschwert im Automaten sind und beginnt dann mit der Farbänderung.

CODING

Im Prozess dieser Farbänderung wird erst der Farbwert des Containers abgefragt (ähnlich der im oberen Punkt beschriebenen Interaktion) und dieser RGB wert dann auf das Lichtschwert angewandt. Zu Guter Letzt wird die Intensität des Leuchtens neu berechnet, da ohne diesen Schritt das Lichtschwert nicht sein Leuchten erhält und die Intensität leider kein direkt änderbarer Wert von HDRP/Lit shadern ist.
Sobald der Farbänderungsprozess abgeschlossen ist wird das Lichtschwert aus dem Automaten geworfen und der Farbcontainer wird langsam geleert. Für den leeren Farbcollider wird daraufhin die Gravitation wieder aktiviert und somit fällt dieser einfach neben der Maschine zu Boden.

Texturen

Für die Texturen haben wir uns Anfangs für realistische entschieden, da wir diese sehr passend für unser Thema gefunden haben. Jedoch sind wir vorerst auf unsere Grenzen gestoßen, wie wir wirklichkeitsnahe Texturen erstellen konnten. Zwischendurch ist die Entscheidung von realistischen Texturen also eher auf Cartoon-artige bz. animierte Texturen gefallen. In diesem Fall wurde es, um einiges einfacher mit Photoshop verschiedene Kacheln für den Boden oder die Wand zu erstellen. Nachdem wir uns weiter über das Texturieren informiert haben und weitere Referenzen und Tutorials gefunden haben sind wir doch wieder in die lebensechte Richtung für das Aussehen unseres Spiels gegangen.

Da niemand wirkliche Erfahrungen mit dem Erstellen von realistischen Texturen hat haben wir die Grundlegenden Texturen von der Webseite textures.com bezogen. Wir brauchen die Albedo also die Standard Map wie auch die Normal, Roughness und die Metallic Map.

Die Albedo Map enthält die Farbe, jedoch ohne Schatten und Highlights. Die Normal Map speichert geringfügige Höhenunterschiede auf der Oberfläche des Materials die so deutlich performance optimierter dargestellt werden können als tatsächliche Höhenunterschiede im Modell zu haben.Die Roughness speichert welche stellen des Materials zu welchem Wert Licht reflektieren was besonders bei den vielen Metalloberflächen des Hangar und der Realtime Reflection oft genutzt wurde. Zu guter Letzt haben wir noch die Metallic Map die sehr ähnlich der Roughness Map bestimmt wo wie stark Licht reflektiert wird. Allerdings reflektiert Metall licht anders als völlig glatte Materialien und benötigt seine eigene Textur.

Wir implementierten einen eigens geschriebenen Standard Shader mit dessen Hilfe wir die meisten fotorealistischen Texturen implementieren konnten. Das half uns vor allem mit der variierenden Verfügbarkeit der Texture Maps verschiedener Materialien.

Durch das UV Mapping wurden von den in Unity gefertigten Wände und Decken die Flächen in kleinere Abschnitte unterteilt, dass es einfacher ist die Texturen richtig anzuordnen. Ich habe die gruppierten Flächen getrennt damit ich die Tiles bzw. Platten genau auf den Boden passen. Man konnte im UV Editor von Unity auch die Größe der Tiles einstellen und hier habe ich mich für ein Viertel der originalen Größe entschieden habe.

Da ich jedoch nicht für alles eine perfekte Textur gefunden habe, habe ich die gefundenen in Photoshop so verändert, dass aus der Textur für einen graden Weg auch für die Ecken eine Textur existiert.

Risikoanalyse

RISIKO	RISIKOMINDERUNG
STEUERUNG Es gibt mehrere gängige VR-Controller auf dem Markt. Es kann somit schnell passieren, dass die Steuerung auf manchen Controller unintuitiv ist.	Wir achteten schon beim Entwerfen der Kontrollschemata darauf, dass diese intuitiv und für alle gängigen VR Controller (Index, Vive, Oculus) anwendbar sind.
PERFORMANCE In einer VR Anwendung mit futuristischem Aussehen kann es aufgrund von Lichteffekten, sowie reflektiven Texturen zu Performanceproblemen kommen. Diese könnten gerade in einer VR Anwendung zu Motionsickness führen.	Nur bestimmte Lichter werden in realtime berechnet Bei den 3D-Modells wurde darauf geachtet keine High-Poly zu modeln sondern Low -bis Mid-Poly Modelle zu machen und das mit realistischen Texturen auszugleichen. Bei manchen Animationen haben wir es im Code so gelöst, dass sie sich nur bewegen wenn das Objekt im Sichtfeld vom Spieler ist.
AUFWAND UND IMPLEMENTIERUNG Das Erstellen eines VR Spiels ohne Vorerfahrung birgt stets ein Risiko, da es nicht einfach ist den Arbeitsaufwand einzuschätzen.	Da C# sehr ähnlich zu Java ist, wurde uns die Einarbeitung in Unity durch Vorerfahrungen etwas erleichtert. Außerdem gab es sehr viele Tutorials und eine gut geschriebene Doku, welche das Arbeiten mit den Programmen erleichterte. Ein paar Gruppenmitglieder hatten auch schon Erfahungen in Unity oder Blender, somit gab es gruppenintern auch Anlaufstellen.
TEAMKOORDINATION Innerhalb eines Teams können schlecht verteilte Aufgaben und schlechte Kommunikation dazu führen, dass Features doppelt bearbeitet werden oder schlecht ineinander greifen.	Wir organisierten uns über Discord und erstellten am Anfang die User Story um unser Ziel vor Augen zu halten. Ab dem Zeitpunkt trafen wir uns wöchentlich und verteilten nach erfolgreichem Abschluss von Aufgaben wieder neue.

Projektstand Ende des Semesters

Jetzt am Ende des Semesters und mit Blick auf unsere Ideen haben wir einiges umgesetzt, nichtsdestotrotz gab es auch Features, welche wir nicht einbauen, Jedoch durch andere Feature-Ideen, die im Prozess entstanden sind, ersetzen konnten. Da unser Spiel sich als Sandboxs, ohne eine vorbestimmte Liste an Spielmechaniken, kategorisieren lässt, schadete das agile Anpassen der angepeilten Spielinhalte dem finalen Produkt in keinster Weise.

Eindrücke

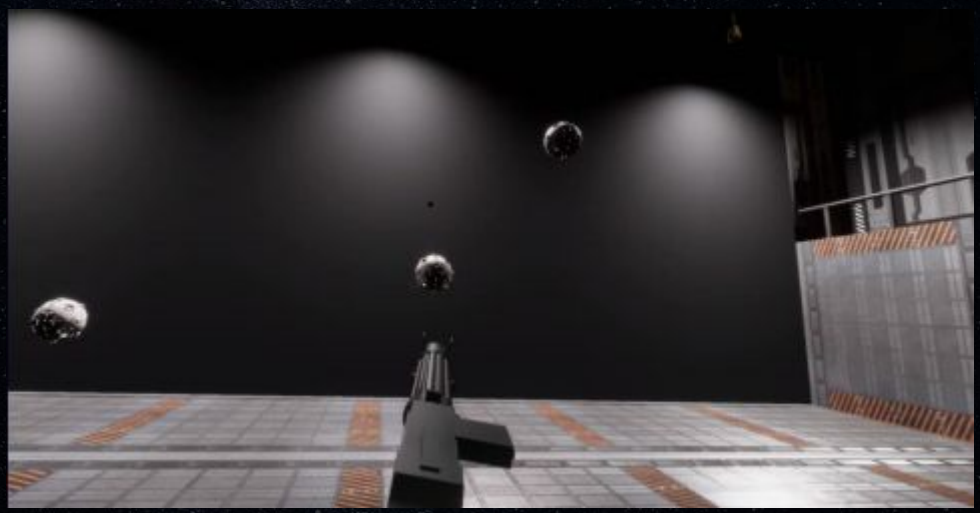
Um nun einen visuellen Eindruck vom Spiel zu bekommen haben wir ein paar Screenshots aufgenommen:

Lichtschwert

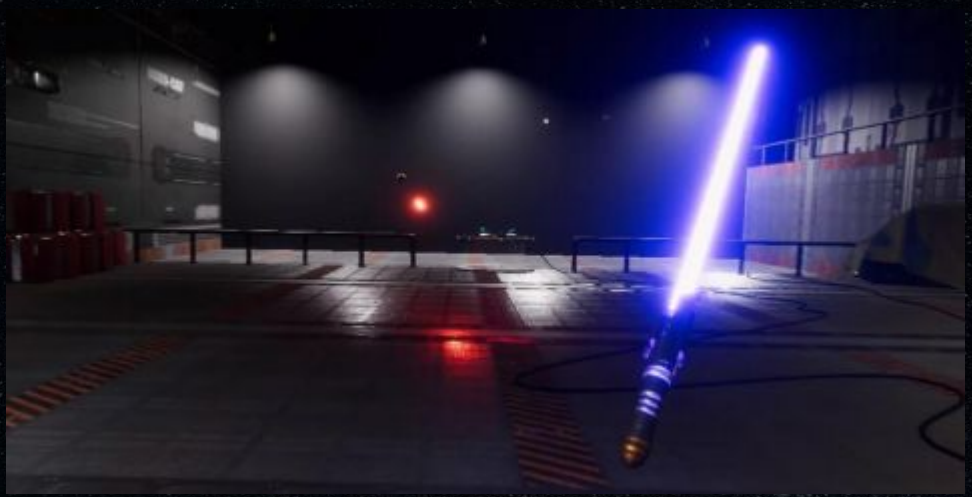
Vorbeifahrender R2

Projektstand Ende des Semesters

Force push



Blastertraining



Laserschwert Abwehrtraining

Projektstand Ende des Semesters

Vollständige Feature Liste

- Freie Bewegung mithilfe des Touchpads
- Vollständig gemodellter und texturierter Hangar
- Interaktive Objekte
- Vollständige Audio
- Funktionale Laserpistole
- Funktionales Lichtschwert
- Zerschneidbare Objekte
- Animierter R2-A2 Droide
- Gegnersphären
- Blockbare Kugeln
- Funktionaler Aufzug
- Animierter Außenbereich
- Funktionaler Verbrennungsofen
- Verschiedene Interaktive Objekte (Garagentor, Pipewheel,...)
- Lichtschwertautomat
- Funktionaler Flüssigkeitsshader
- Reset Hebel
- Machtbasiertes Bewegen von Objekten
- Machtbasiertes Heranziehen des Lichtschwerts

Was wir aus dem projekt gelernt haben

Da unser Projekt für uns alle das erste mit einer solchen Gruppengröße war, lernten wir viel über das Arbeiten in Gruppen und das sinnige Koordinieren ebendieser. Somit wurde es im Laufe des Projekts klar, dass reine Freiheit zwar inspirierend und motivierend war, aber nicht immer ein ausgeglichenes Arbeitsverhältnis kreierte. Nach einigen Wochen aktiver Entwicklung begannen wir also für Aufgaben Deadlines zu setzen und deren Einhaltung als Priorität anzusehen.

Während wir in diesem Punkt mit der Zeit etwas strenger wurden, so achteten wir darauf, dass man sich gegenseitig stets unterstützte und Wissen teilte. Interessant hierbei waren die unterschiedlichen Herangehensweisen an Probleme und Problemsituationen. Aufgrund dieser verschiedenen Ansätze jedes Gruppenmitglieds hatte sich schnell jeder in einem speziellen Bereich Fachwissen angelesen und konnte dies dann unterstützend mit den anderen Gruppenmitgliedern teilen.

Ein weiterer Aspekt in dem das Teilen von Informationen und Wissen innerhalb des Teams essentiell war, ist der SteamVR Teil. Im Laufe des Projektes stellten wir schnell fest, dass SteamVR nicht so umfangreich und anfängerfreundlich dokumentiert ist, wie viele von uns es kennen. Somit war es wichtig, dass jeder neue Erkenntnisse mit allen anderen teilte und als Ansprechpartner in diesem Thema bereitstehen konnte.

Für kommende Unity Projekte haben wir allerdings bereits beschlossen nicht mehr git mit dem Github Plugin für Unity zu benutzen. Diese Kombination führte zu unzähligen Mergekonflikten der Szene und auch wenn das Aufstellen von Regeln im Bezug auf "Wer darf die Szene bearbeiten?" die Probleme zwar reduzierten, so scheint uns Unitys Collaborate-feature doch sinniger für zukünftige Projekte.

Schlussendlich war es aber für uns alle eine spannende Erfahrung mit neuer und uns unbekannter Technik zu arbeiten. Wir haben sowohl individuell als auch als Gruppe viel dazugelernt und freuen uns auf das nächste Projekt.