Roskilde University

Virtual Reality som eksponeringsterapi

(https://mwi.usma.edu/mwi-podcast-four-combat-jump-veterans-tell-stories/)

Udarbejdet af gruppe S2026631-2:

Daniel Vestmar Norén: 65246

Frederik Røikjær Jensen: 66454

Kathrine Kauffeld Andreasen: 65204

Mikkel Kjær: 66370

Tobias Mikkel Schleiss-Andreassen: 66715

FAGMODULSPROJEKT I DATALOGI 2020

Vejleder: Martin Kirkegaard

3. juni 2020

Forord

Dette projekt er skrevet af studerende på 4. semester med fagmodulet datalogi på Roskilde Universitet under vejledning af Martin Kirkegaard og co-vejleder Ebbe Vang. Projektet henvender sig til studerende med kompetencer samt interesse indenfor ny og spændende teknologi.

Rapporten beskriver arbejdsmetodikken brugt for at besvare problemformuleringen, herunder de valgte afgrænsninger. Projektet er udarbejdet på baggrund af en interesse for samarbejde mellem kadetter fra Hærens Officersskole og RUC med teknologiske kompetencer. Den fremlægger konklusionen om, at der gennem en struktureret arbejdsproces kan udvikles en vellykkes VR-oplevelse ved hjælp af spilmotoren Unity. En VR-oplevelse som kan vække følelsen af at være højt oppe over jorden og på den måde være med til at behandle højdeskræk.

Der skal lyde en stor tak til vores vejleder, Martin Kirkegaard, som har været os behjælpelig i forbindelse med projektskrivningen. Derudover vil vi takke Ebbe Vang for god hjælp og sparring i forbindelse med udarbejdelse af selve programmeringen. Ligeledes vil vi takke Benedicte Fleron, som har været kontaktperson mellem os og Hærens Officersskole, som vi også vil udtrykke vores taknemmelighed for, da de har hjulpet os igennem designovervejelser med mere.

Abstract

This project revolves around how to structure as well as document the development process for a virtual reality application for Oculus Quest, and how it can be conceptualized and created to serve the purpose as an aid to treat fear of height. The project is structured mainly using a Kanban board and the application is developed using the Unity Game Engine and Visual Studio with written code in the C# programming language which is native to Unity.

Inspired by the game Richie's Plank Experience, the application uses elevators and buildings to simulate the users ascending travel from the ground and up, with the purpose of triggering the players fear of heights. By using Unity's functions to create an environment which contains objects of realism, sounds and physical movement.

Resume

Dette projekt ønsker at vise, hvorledes udviklingen og dokumentationen af en virtual reality applikation til en Oculus Quest, kan struktureres og konceptualiseres med det formål, at kunne behandle højdeskræk. Projektet er struktureret ved brug af kanbanboard og applikationen er udviklet ved brug af Unity Spilmotoren og Visual Studio, med skreven kode i C# programmeringssproget, hvilket er standard for Unity.

Inspireret af spillet Richie's Plank Experience benytter applikationen elevatorer samt bygninger til at simulere brugerens rejse fra bunden af den virtuelle verden til toppen, med det formål at udsætte brugeren for en højdeskræks udløsende situation. Ved at bruge Unitys funktioner til at skabe omgivelser som indeholder objekter af realisme, lyde og fysisk bevægelse.

Roskilde University

Indholdsfortegnelse

Forord **2**

Abstract **3**

Resume **3**

1.0 Indledning **8**

1.1 Problemformulering *9*

2.0 Problemanalyse **10**

2.1 Problemfelt *10*

2.2 Indledende designovervejelser *11*

2.3 Afgrænsning *11*

3.0 Metode **13**

3.1 Kvalitativ metode *13*

3.2 Kanban *17*

3.3 Collaborate og versionsstyring *20*

4.0 Virtual Reality **22**

4.1 VR som motivation *22*

4.2 Oculus Quest *23*

5.0 Unity **25**

5.1 Unity Hub *25*

5.2 Unity Editor *26*

5.3 Assets *30*

5.4 GameObjects, Components og Variables *35*

5.5 Meshes, Materials, Shaders og Textures *36*

5.6 Prefab *37*

6.0 Programmet **40**

6.1 Generelt om scripts *40*

6.2 Nedarvningshierarki i Unity *41*

6.3 Scripts i projektet *43*

6.4 Spilverdenen *50*

6.5 Hvorfor har vi lavet spilverdenen som den ser ud? *68*

7.0 Brugervejledning **73**
8.0 Afprøvning **75**
9.0 Diskussion **77**
9.1 Processen *77*
9.2 Forbedringer *78*
10.0 Konklusion **80**
11.0 Refleksion **82**
12.0 Litteraturliste **84**

Ordliste

AR	Augmented Reality
Bitmap billede	Et kort over punkter i et billede delt op i rækker og kolonner, hvor hvert punkt har sin egen farve
FPS	Frame Per Second
GUI	Graphic User Interface
I/O	Input/Output
UML	Unified Modeling Language
VR	Virtual reality
XR	Et plug-in til både AR og VR

Projektvideo på YouTube:

https://www.youtube.com/watch?v=l4GImZlj68k&feature=youtu.be&fbclid=IwAR0MHsUo4p0enG0W_zpZksvVoRPSY99IVDYCvDjpFfi7-lmxLoUB_PHRXw4

Projektfil på drev:

https://drive.google.com/drive/folders/1qMoz3JRu0439VmB2Yg7PDQ0pKOd0XSz5?fbclid=IwAR3OwnvchjQ1j7sP94MnZRz9IzCV0Lfu13s45AJ-B15IYP3uSMFdIUhoJrw

1.0 Indledning

De fleste mennesker kender til virtual reality (VR), eller har hørt om det før, men hvad kan VR egentlig bruges til? Mange associerer VR med spil, men i realiteten kan det bruges til meget mere end det.

Prototyper af produkter, som er meget dyre at designe og fremstille fysisk, kan designes virtuelt gennem VR. Dette gør det billigere at designe eksempelvis biler. Derudover kan det bruges til at give mennesker på hospitaler en oplevelse af at være et andet sted, selvom deres fysiske tilstand ikke tillader, at de forlader hospitalet **(Region Hovedstadens psykiatriske hospital & Andreasen, 2019).** VR er specielt brugbart i træningssituationer, hvor der skal trænes i noget, som kan være dyrt eller farligt at simulere. Dette kunne for eksempel være nyttigt for officerer i militærskolen, som skal træne til faldskærmsudspring (**Lichtenstein, 2017)**.

Hvor virtual reality i nogle institutioner er udbredt og brugt flittigt, har andre ikke fundet en brugbar use-case for VR endnu. Hærens Officersskole har indgået et samarbejde med Roskilde Universitet for at undersøge, om VR kan være brugbart for dem i forbindelse med kadetternes eksamen i faldskærmsudspring. Idéen er at undersøge, om VR kan være behjælpelig i forhold til højdeskræk gennem eksponeringsterapi (**Psykologisk Ordbog, n.d.**).

Eksponeringsterapi er en behandlingsmetode, som har til formål at behandle angst hos mennesker. Det involverer at udsætte det angstramte individ for angstkilden, eller konteksten af denne uden at udsætte individet for direkte fare. VR gør det muligt at udsætte en person for konteksten af en farlig situation, men i et sikkert miljø som hjemme i dagligstuen (**Jensen & Konradsen, 2017, 11**).

Vi ønsker med dette projekt at lave vores eget VR-spil, som skal agere som en form for eksponeringsterapi for akrofobi. Vi har udformet en problemformulering samt arbejdsspørgsmål til formålet at udvikle et VR-program i Unity, der ved hjælp af en Oculus Quest skal kunne hjælpe brugeren til at agere mere rationelt i fremtidige situationer med højde.

1.1 Problemformulering

Hvordan udvikles en VR-simulering i Unity, der kan afhjælpe højdeskræk for eventuelle officerer i militærskolen, og hvordan struktureres processen?

Arbejdsspørgsmål:

- Hvordan benytter vi os af Kanbanboardet til processen?
- Hvad er Unity?
- Hvordan opbygges et VR-spil i Unity?

2.0 Problemanalyse

I dette afsnit vil vi kort beskrive, hvad rationalet for projektet er og formålet med idéen. Vi vil forklare problemstillingen, samt hvilken målgruppe vi laver programmet til. Samtidigt vil vi forsøge at sætte ord på overvejelserne, der ligger bag designet for programmet.

2.1 Problemfelt

På Hærens Officersskole er det obligatorisk at gennemføre et faldskærmskursus, der skal give kadetterne et kompetencefelt i løbet af deres uddannelse, som vurderes væsentligt for den enkelte kadet og vedkommendes fremtidige karriere. Årsagen til denne væsentlighed beror sig blandt andet i arbejdsfeltet og sikkerhedsmæssige omstændigheder, der skal opfyldes. Essensen af at springe ud af et fly med faldskærm indebærer, at kadetten er i stand til at udsætte sig selv for noget, der kan fremstå som værende livsfarligt. Der er derfor nogle kadetter, der undervejs i uddannelsen frafalder på grund af det, som bliver antaget som værende højdeskræk. Dette gør det svært for den enkelte kadet at bestå dette kursus og derved deres uddannelse. Som følge af denne problematik har Officersskolen taget kontakt til Roskilde Universitet for at finde en løsning på, hvordan kadetterne kan få bearbejdet en mulig højdeskræk før vedkommendes endelige faldskærmsudspringsøvelse (bilag 1). Hertil blev det foreslået at foretage en simulering gennem virtual reality, der gennem gamification ville kunne træne og/eller udsætte de individuelle kadetter i højder. Denne løsning krævede derfor en undersøgelse om, hvorvidt forskningsgenstanden fremstilles, og hvorledes processen undervejs skulle struktureres.

Af denne årsag skulle vi i projektgruppen finde ud af, hvordan et videospil ville kunne udvikles, og hvordan processen skulle struktureres, herved hvilke modeller og programmer samt I/O-enheder ville være fordelagtige at benytte med hensyn til løsningen af problematikken. Først og fremmest fandt vi, at udviklingen af videospillet ville give det bedst mulige resultat gennem spiludviklingsprogrammet kaldet: 'Unity'. Hernæst skulle processen struktureres over, hvordan spillet skulle udvikles, fremfindes, og hvordan produktet skulle se ud i sidste ende.

2.2 Indledende designovervejelser

Målet med projektet er, som tidligere beskrevet, at realisere en virtuel verden til behandling af højdeskræk for Officersskolens kadetter. Gennem designprocessen har gruppen været i kontakt med kadetterne fra Officersskolen til et gruppeinterview, der havde til formål at indsamle empiri til design af den virtuelle oplevelse. Målet var at udvinde relevant information, der med fordel ville kunne være behjælpeligt til et VR-design, som bedst muligt ville kunne afhjælpe kadetternes højdeskræk.

Ved projektets begyndelse blev gruppen anbefalet at benytte Unity til design af den virtuelle verden, da Unity efter sigende er en brugervenlig platform. Unity gør det muligt at lave tredimensionelt grafisk design, samt at arbejde med scripts. Med andre ord har Unity alle de nødvendige funktioner, der kræves for at udvikle et spil uden brug af tredjeparts software.

Ved overvejelse af det virtuelle design, skulle der i gruppen tages højde for den begrænsede tidshorisont samt de kravspecifikationer, der er sat for semesterprojektet. Vi har diskuteret internt, at det visuelle og grafiske i virkeligheden er det vigtigste, da formålet med VR-spillet er, at kadetterne skal kunne afhjælpes med højdeskræk, som bærer præg af en virkelighedstro fornemmelse (**JENSEN & KONRADSEN, 2017, 11**).

2.3 Afgrænsning

Alle fem gruppemedlemmer har tidligere udelukkende været vant til at skrive i kodesproget Java. På nuværende semester (fjerde semester) er vi nu blevet udfordret i C#. Kodesproget er afgrænset til at være i C#, grundet det er det mest almindelige programmeringssprog i Unity, hvilket også vil betyde, at det vil være nemmere at søge viden om programmering i Unity, hvis det mest brugte sprog benyttes. Samtidig er dette sprog objektorienteret, og syntaksen minder om Java.

Vi har valgt at afgrænse os fra at beskrive de funktioner i GUI for Unity, metoder og klasser, hvor vi ikke finder denne viden nødvendig for forståelsen af vores eget program, eftersom det er langt fra alle dele af Unitys GUI, vi har taget i brug. Herved har vi valgt at fokusere på at beskrive de mest væsentlige metoder i de forskellige klasser i vores program og ikke alle metoder.

Vi startede ud med at bruge et asset til terrænet, der ikke var tilgængeligt gratis, men kostede penge. Efter køb af dette ene asset blev vi i gruppen enige om, at det ville blive for dyrt i længden, hvis vi skulle købe os til alle assets. Derfor benytter vi os næsten udelukkende af de assets, der er

tilgængelige gratis. I andre tilfælde ville assets, der er færdigudviklede og prædefinerede, kunne købes, så det visuelle og grafiske vil være mere veludviklet, samt vil spillet forekomme mere virkelighedstro. Det grafiske i vores VR-projekt er derfor afgrænset af økonomiske årsager.

Da vi startede ud med projektet, valgte vi at lave programmet kompatibelt til en Oculus Quest, eftersom det var den VR-brille, vi havde mulighed for at bruge og programmere til. Samtidig var det en fordel for os at udvikle spillet på Questen, da det er en stand-alone enhed, og vi på den måde var uafhængige af at være på studiestedet, men også kunne befinde os andre steder og udvikle samt teste programmet. Efterfølgende har vi dog ændret i nogle essentielle dele af programmet, der gør, at programmet nu er kompatibelt med alle former for headsets. Vi har derfor mulighed for at bruge vores program på både en Oculus Quest, Rift og HTC Vive.

Da vi skulle gøre vores program kompatibelt med alle former for headsets, var vi nødt til at ændre versionen, vi programmerede i. Vi benytter os af version 2019.3.14f1, eftersom det på daværende tidspunkt var den nyeste version af Unity, samt understøtter den det XR-plug-in, der muliggør brugen af samtlige headsets.

Til slut har vi også valgt at afgrænse teoretiseringen af, hvad gamification kan gøre for de øvelser kadetterne fra Officersskolen skal gennemgå. Projektet endte ud med at være et program, som ikke havde tilstrækkelige spilmekanikker og kunne derfor eftertragtes som en simulering over et videospil. Af denne årsag passede gamification delen således ikke som en teori, der skulle anvendes for at fremfinde designkarakteristika til programmet, men trods denne afgrænsning vælger vi stadig at diskutere teoriens mulige omfang gennem rapportens diskursive og refleksive afsnit.

Med dette sagt vil vi herfra se nærmere på hvorledes vi har undersøgt kadetterne gennem metodiske tilgange til empiriske resultater, der kunne bruges som inspirationskilde til designet af programmet.

3.0 Metode

I dette afsnit vil vi fokusere på de metoder, vi har brugt til at fuldende det endelige produkt. Hele projektet bygger på en problemstilling, som vi så vidt muligt vil finde en løsning til på basis af vores programmeringsevner. Vi har derfor haft en snak med kadetterne, som er dem, der har fremlagt problemstillingen for os. Da vi fik et godt overblik over, hvilken problemstilling kadetterne har, har vi forsøgt at strukturere programmerings- samt skriveprocessen ved hjælp af et KanbanBoard og verssionsstyring. Vores fremgangsmåde i forhold til både interviews med kadetterne samt strukturering af den videre proces vil blive uddybet i de kommende afsnit.

3.1 Kvalitativ metode

Undervejs i problemformuleringsfasen stod det for gruppen klart, at hvis produktet skulle kunne imødekomme projektets målgruppes behov, ville vi blive nødt til at foretage nogle undersøgelser. Dette er for at kunne belyse problematikken, samt hvilke spilelementer som med fordel kunne implementeres i projektets produkt. Af denne årsag valgte vi at invitere kadetterne og deres holdledere til et møde, hvorunder kadetterne kunne få lov til at opleve, hvordan virtual reality fungerer gennem et spil kaldet Richie's Plank Experience - dette i samarbejde med Benedicte Fleron, som er vores kontaktperson mellem Hærens Officersskole, og Adam Buschard, som er holdleder for Hærens Officersskole.

VR-spillet Richie's Plank Experience er et spil, hvor brugeren kan teste sig selv i højder gennem VR. I spillet kan brugeren gå ind i en elevator og herved blive præsenteret med forskellige knapper, som kan interageres med. Knapperne kan dermed føre testpersonen hen til forskellige niveauer af spillet, som har varierende formål. I vores eksempel fik vi kadetterne til at køre op til den etage, hvor brugeren kan gå ud på en planke, der er fastspændt højt oppe over bygningens fundament. Gennem brug af spillet er det også muligt at opstille en virkelig planke, hvor der med brug af VR-headsettets controllere kan indstilles på plankens dimensioner, således at planken i spillet reflekterer den virkelige planke (**"RICHIES PLANK EXPERIENCE", N.D.**). Dette gør, at når brugeren træder ud på planken, som de kan se det i VR-headsettet, vil de kunne mærke den reelle planke under deres fødder og herved have en mere virkelighedsnær oplevelse.

Undersøgelsen havde en kvalitativ tilgang, eftersom vi benyttede os af forskellige former af interviews (KVALE & BRINKMANN, 2015, 29), der kan beskrives som; semistruktureret gruppeinterviews og to stramt struktureret computerstøttet ekspertinterviews. De semistruktureret gruppe interviews havde kadetterne som informanter, hvor de stramt struktureret computerstøttet ekspertinterviews havde kadetternes holdledere som informanter. Årsagen til, at vi benyttede os af en kvalitativ tilgang frem for en kvantitativ tilgang, beroede sig i, at vi som projektgruppe ønskede at finde informanternes personlige holdning. Denne holdning skulle afspejle, hvad et VR-spil, der tester en person i højder, ville kunne gavne dem i deres kursus i faldskærmsudspring. Herudover ville vi også gennem undersøgelsen finde ud af, hvilke spilelementer der med fordel kunne implementeres i vores projekt.

Kvalitativ metode er, når der benyttes en bestemt tilgang gennem en metode til at opnå informationer vedrørende fænomener såsom forskningsinterviewet, der reflekteres gennem subjektive holdninger (KVALE & BRINKMANN, 2016, 48). Med denne tilgang havde vi mulighed for at komme tættere på informanternes livsverden vedrørende problematikken. Samtidigt havde vi mulighed for at opnå yderligere viden omkring informanterne som følge af opfølgende spørgsmål på diverse svar fra informanterne gennem de forskellige interviews (BRINKMANN & TANGAARD, 2015, 36). Havde vi tværtimod benyttet os af en kvantitativ tilgang, ville informanterne ikke have haft mulighed for at uddybe deres holdninger og erfaringer, og derfor ville vi miste chancen for at opnå potentielle vigtige faktorer angående problematikken og herved evidens for designgrundlag (BRINKMANN & TANGAARD, 2015, 446-450). Ydermere er personlige holdninger, der relateres til denne forskningskontekst heller ikke kvantificerbar, eftersom der er tale om kadetternes og deres holdlederes følelser, erfaringer eller holdninger .

Undervejs i mødet blev der som nævnt taget brug af gruppeinterviews til at interviewe kadetterne (bilag 2). Dette blev gjort henover 3 forskellige interviews, eftersom vi ønskede at interviewe dem før og efter brugen af VR-spillet Richie's Plank Experience. Hermed fik vi identificeret det generelle problem med at springe ud med faldskærm. Samtidigt kunne vi fastslå de forskellige elementer fra spillet, der med fordel kunne implementeres i vores projekt, som ville have en effekt på kadetten.

Gruppeinterviews er en interviewform, hvor forskeren interviewer mere end én informant. Herigennem får forskeren derfor mulighed for at dække en større mængde af mennesker til forskel fra det almindelige interview ("KVALITATIVE DYBDEINTERVIEW | FREMFORSK, CENTER FOR FREMTIDSFORSKNING," N.D.), hvorfor vi valgte at bruge gruppeinterviews i stedet for interviews hvor én enkelt informant deltager i interviewet. Dette gør sig netop gældende for vores projekt, da vi havde begrænset tid med kadetterne, fordi de skulle tilbage til Aalborg efter halvanden time. Samtidigt mødte der tolv kadetter op, hvorfor vi valgte at interviewe dem i to grupper, så vi kunne få plads til, at alle havde mulighed for at ytre sig.

Én fordel med gruppeinterview, der såmænd også kan karakteriseres som en ulempe ved brugen af denne interviewform, er, at der kan samles empiri ud fra, hvordan sandheden manifesteres gennem de sociale konstruktioner, og de fænomener der opstår undervejs i interviewet ("KVALITATIVE DYBDEINTERVIEW | FREMFORSK, CENTER FOR FREMTIDSFORSKNING," N.D.). Dette betyder, at informanterne er i stand til at påvirke hinandens optræden i interviewet, herunder også hvad svaret på det eventuelle spørgsmål indeholder ("KVALITATIVE DYBDEINTERVIEW | FREMFORSK, CENTER FOR FREMTIDSFORSKNING," N.D.). Det kan derfor siges, at gruppeinterviewet kan være med til at få folk til at være mere aktive i interviewet alt efter, hvordan deres sociale forhold fungerer i den givne gruppe. Det kan dog også være med til at skabe empirisk usikkerhed, eftersom informanterne kan påvirke hinanden med deres svar ("KVALITATIVE DYBDEINTERVIEW | FREMFORSK, CENTER FOR FREMTIDSFORSKNING," N.D.).

Hertil valgte vi at forholde gruppeinterviewet semistruktureret, eftersom informanterne gennem et semistruktureret interview ville have mulighed for at interagere med hinanden. Som forskere havde vi med denne struktur også mulighed for at have en vis styring over, hvilken retning interviewet gik (BRINKMANN & TANGAARD, 2015, 37-38).

Herudover benyttede vi os også af stramt struktureret computerstøttet ekspertinterviews til at komme tættere på kadetternes holdlederes erfaring med kadetternes kursus i faldskærmsudspring (KVALE & BRINKMANN, 2016, 201-202). Interviewene var opstillet således, at imens kadetterne var i gang med de øvrige aktiviteter, havde holdlederne mulighed for at besvare nogle specifikke, samt åbne spørgsmål, der kunne reflektere deres erfaring med kurset. Endvidere refleksion over

hvilke problematikker, der opstår ved kursets endegyldige faldskærmsudspring, og hvilke ting de gerne ville have implementeret i vores projekt (bilag 3 og 4).

Figur 1: *Kadetter på Roskilde Universitet*

Gruppen afholdt et oplæg for kadetterne inden afprøvning af VR. Privatfoto fra den 4/3, da kadetterne var på RUC og besøge os for at afprøve virtual reality (VR).

Selve mødet med kadetterne var opsat, således kadetterne havde mulighed for at prøve VR samtidig med, at vi kunne interviewe dem før og efter brugen af VR. Til start afholdt vi en kort introduktion af, hvad mødets agenda bestod af, hvad VR er, og derefter en opsplitning af kadetterne ud i to forskellige hold. Det ene hold forblev dermed i introduktionslokalet, hvor det andet hold blev sendt ind i et separat lokale, hvori de afprøvede VR-spillet: Richie's Plank Experience. Det tilbageblivende hold blev interviewet om, hvilke forventninger de havde til VR, og hvad VR ville kunne bidrage med til deres kursus i faldskærmsudspring. Efter dette skiftede holdene lokale og gav dermed det andet hold mulighed for også at prøve VR.

Figur 2: *Kadetter på Roskilde Universitet*

Kadetterne fra Hærens Officersskole afprøver VR. Dette foto er et privatfoto fra den 4/3, da kadetterne var på RUC og besøge os for at afprøve virtual reality (VR).

Det første hold, der netop havde prøvet VR-spillet, blev herfra interviewet om deres oplevelse med spillet og hvilke ting ved spillet, der i deres øjne fungerede godt i forhold til at påvirke deres sans for højde. Til slut, da alle kadetterne havde prøvet VR-spillet, samledes holdene og fik mulighed for at give inputs til, hvordan spillet kunne hjælpe til at forberede dem til deres kommende afslutning af deres faldskærmskursus.

Vi har herved redegjort for, hvorledes vores undersøgelse af kadetterne har fundet sted til udvikling af et program, hvis empiri vi i afsnit 6.4 vil beskrive nærmere. Her vil vi også beskrive, hvilke designvalg vi har foretaget på baggrund af de interviews, vi har foretaget i samarbejde med kadetterne. Med dette gennemgået vil der herfra ses nærmere på, hvordan udviklingsprocessen af programmet og denne rapport er blevet struktureret gennem Kanban metoden.

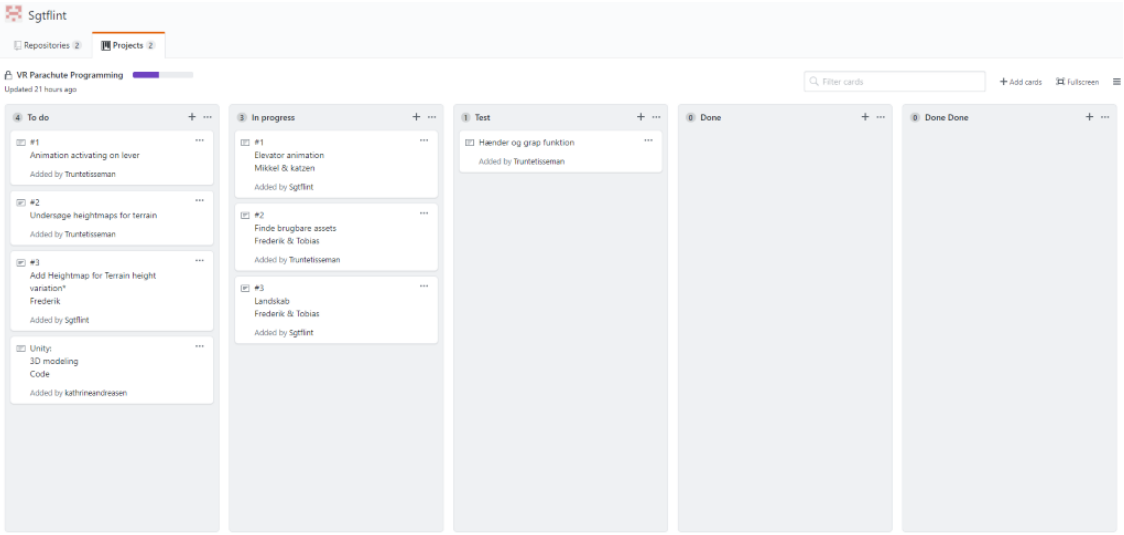
3.2 Kanban

Et Kanbanboard er et hjælpemiddel til at organisere og effektivisere arbejdsprocessen på en visuel og agil måde. Dette betyder, at der indføres fleksibilitet i planlægning af processen om systemudviklingen. Herved skaber processen fokus på prioriteringer af de dele af programmet der

er mest væsentlige samt et overblik over de dele af programmet, der fra "backloggen" er under udvikling. Fleksibiliteten ligger i at opgaverne i backloggen prioriteres, og prioriteterne løbende opdateres eksempelvis ud fra, hvor væsentlig opgaven er for funktionaliteten, eller hvor meget værdi en funktion eller opdatering gavner brugeren i forhold til, hvor meget udviklingstid og omkostninger det tager at implementere (**"HVAD ER KANBAN," N.D.**).

Kanbanboards visualiserer også processen, så det er nemt for alle gruppemedlemmer at se, hvilke opgaver der fokuseres på, og hvilke der er færdige. Det gøres ved at opstille en tabel og inddele arbejdsprocessen i forskellige stadier eller kolonner, hvor de forskellige arbejdsopgaver inddeles som kort. Kortene kan rykkes rundt mellem de forskellige kolonner, alt efter hvor langt opgaven er henne i processen (**ATLASSIAN & REHKOPF, N.D.**).

Vi benyttede GitHubs skabelon for Kanbanboard gennem hele projektet til at planlægge og have overblik over arbejdsprocessen. Både i forhold til udviklingen af selve spillet, men også for skriveprocessen af rapporten, som kan ses på henholdsvis figur 3 og 4.



Figur 3: *Kanbanboard for udviklingen af spillet*

Vi har opsat fem kolonner, som repræsenterer de fem stadier i processen af et færdigudviklet produkt (billedet er et skærmbillede af eget projekt). Link til GitHub: https://github.com/users/Sgtflint/projects/1

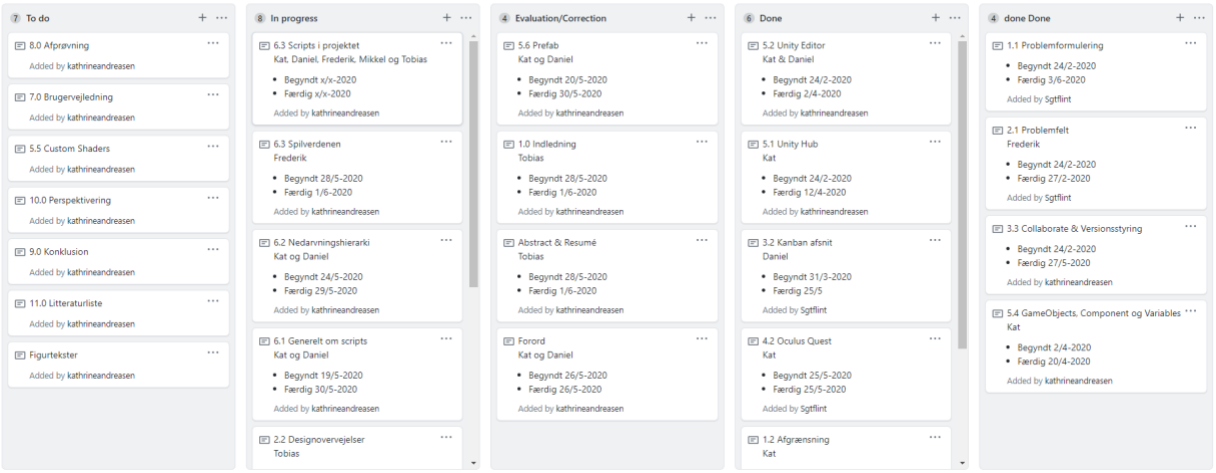
Vi inddelte arbejdsprocessen i fem stadier, som vi synes gav mest mening i forhold til at udvikle vores spil: "To-do", "In progress", "Test", "Done" og "Done done". Mellem disse kolonner vil vi rykke vores kort frem i takt med, at vi er kommet længere med den opgave på det enkelte kort. Vi

har sat de forskellige opgaver i kolonnerne i en prioriteret rækkefølge, således de øverste for hver kolonne er førsteprioritet - på den måde kan vi holde styr på rækkefølgen af vigtige opgaver, og hvilke vi skal gå i gang med først. Til en start havde vi prioriteret de forskellige opgaver ved at give dem numre, hvor dem med laveste nummer var vigtigst. Dette fandt vi dog at besværliggøre processen, når vi ville ændre på prioriteringen af opgaverne, da vi så var nødsaget til at ændre hver enkelt opgave med nyt nummer. Vi beskriver på de forskellige kort, hvem i gruppen der tager sig af de enkelte opgaver, samt hvornår vi ønsker, at den enkelte opgave skal være færdig.

Ved inddelingen af kolonnerne i fem stadier kan vi i den første kolonne beskrive og prioritere de opgaver, som vi ønsker at implementere eller få fixet. Kolonnen kalder vi To-do, da det er opgaver, der skal igangsættes. Idet en arbejdsopgave påbegyndes, overflyttes denne fra To-do til In progress kolonnen, så alle i projektet kan se, at der arbejdes på denne. Når der er blevet arbejdet på arbejdsopgaven tilstrækkeligt, skal denne testes - opgaven kan derfor flyttes over i Test. Vi benytter os af Debug.Log, som er en indbygget funktion i C#, til at teste metoder i programmet. Ved brug af Debug.Log kan vi teste, hvorvidt metoden bliver kørt - hvis det, vi skriver i Debug.Log, bliver printet i monitoren, ved vi, at metoden er blevet kørt. Derved kan vi teste, om der ligger et problem i at få kørt metoden, eller om der er et problem med syntaksen i metoden.

Inden opgaven kan betragtes som færdig og flyttes til Done, skal den forudgående test have været vellykket. Før kortet kan rykkes til den sidste kolonne Done done, skal opgavens funktionalitet præsenteres for resten af gruppen for at sikre kvaliteten og funktionaliteten af opgaven. Dette er også for at sikre, at alle gruppemedlemmer altid er opdateret og forstår, hvad der er blevet lavet. Når et kort er blevet rykket til Done done, kan opgaven anses som værende det endelige resultat.

Kanbanboardet for skriveprocessen fungerer på samme måde, bortset fra Test-kolonnen, som vi her har ændret til at være en evaluering og korrektion af tekstafsnittene, inden de kunne betragtes som værende og færdige og flyttes til Done-kolonnen. Kanbanboardet hjalp os med at organisere selve rapportskrivningen. Kanbanboardet for denne proces kan ses i figur 4.



Figur 4: *Kanbanboard for skrivningen af projektet*

Vi har opsat fem kolonner, som repræsenterer de fem stadier i processen af et færdigudviklet produkt (billedet er et skærmbillede af eget projekt). Link til GitHub: https://github.com/users/Sgtflint/projects/3?fullscreen=true

Ved brug af Kanbanboardet kunne vi også organisere arbejdsprocessen i forhold til Unitys Collaborate feature, som gjorde det muligt for flere gruppemedlemmer at arbejde i Unity samtidigt.

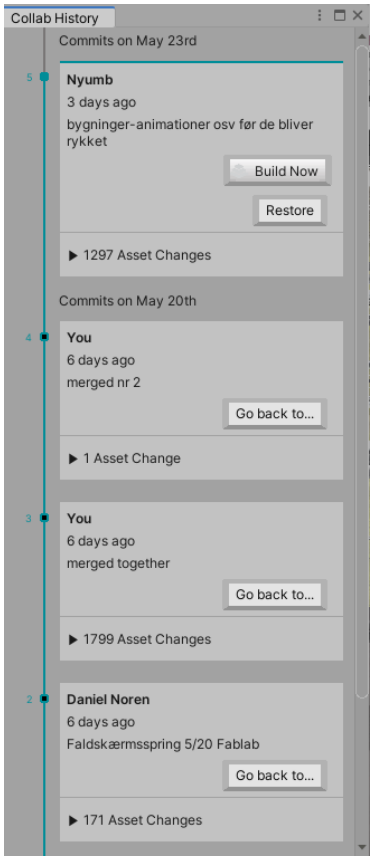
3.3 Collaborate og versionsstyring

Versionsstyring er et utroligt vigtigt værktøj, som er meget udbredt i forbindelse med softwareudvikling. I starten af projektet gjorde vi brug af Unitys Collaborate feature i forbindelse med versionsstyring, som er en indbygget funktion i Unity. Dette gjorde det muligt for os at samarbejde om spillets design og kodningen fra flere forskellige computere, samt inddele arbejdsopgaverne. På den måde kunne vi oprette, ændre, gemme, dele og synkronisere nye versioner af det samme projekt på forskellige enheder. Således kunne vi givetvis være uafhængige af hinanden i forhold til at lave ændringer på projektet (**UNITY TECHNOLOGIES, 2020, "UNITY COLLABORATE"**). Collaborate havde et månedligt gebyr for projekter over to medlemmer, hvilket betød, at vi måtte betale for at benytte denne feature, eftersom vi ønskede, at alle gruppemedlemmer skulle kunne redigere i programmet.

På trods af vi valgte denne løsning, som virkede mest oplagt, virkede Collaborate ikke helt som forventet. Vi har haft en masse udfordringer i forhold til sammenfletning af versioner fra forskellige computere, og vi har heller ikke altid kunne tilgå Unitys fjernlagringsserver. Det

resulterede i, at vi kun havde to computere, vi skiftede imellem at udvikle på, alt efter om vi var på RUC eller hjemme.

På trods af vi ikke brugte Collaborate feature som en gruppe som forventet, benyttede vi os stadig af Collaborate funktionen på de forskellige computerenheder henholdsvis på RUC og hjemme hos en af gruppemedlemmerne. Således fik vi mulighed for at pushe nye versioner til Collaborate, som vi efterfølgende kunne gendanne på samme computer, hvis vi kom til at lave uønskede ændringer i en version. På figur 5 kan et udsnit af vores versionshistorik for projektet ses.



Figur 5: *Versionshistorik*

I Collaborate feature kan versionerne af programmet hentes ned igen fra historikken (billedet er et skærmbillede af eget projekt).

Idet vi skiftede computer fra RUC og hjem og vice versa, overførte vi hele spilmappen til en ekstern SSD og manuelt flyttede over på den anden computer.

4.0 Virtual Reality

Virtual reality er en teknologi, der benyttes til at skabe virtuelle verdener, som en bruger af teknologien kan transporteres ind i. Dette opnås ofte ved brug af et VR-headset.
Vi benytter VR i form af et VR-headset til dette projekt, da der er markant mere indlevelse i VR end ved et typisk videospil på en computerskærm. Indlevelse i den virtuelle verden samt påvirkning af sanserne er vigtigt for at sælge simulationen til brugeren og give dem en følelse af at være højt oppe, som det er tilfældet i vores projekt.

4.1 VR som motivation

Der er længe blevet forsket i VR i forbindelse med læring. Flere studier sammenligner læring ved brug af VR-teknologi med læring gennem mindre indlevelsesrige teknologier, såsom traditionel klasseværelsesundervisning. I et studie fra 2016 fandtes der en tydelig forskel på læringsudbytte ved en sammenligning mellem VR og almindelig kendt computerbaseret læring (**JENSEN & KONRADSEN, 2017, 7**). VR kan også benyttes til affektiv læring, som henviser til følelsesmæssig læring. Affektive færdigheder kræver ligesom andre færdigheder, at brugeren øver sig gentagende gange for at blive god. En succesfuld VR-simulation til træning af affektive færdigheder skal helst være meget interaktiv, så brugeren let kan leve sig ind i den simulerede situation (**JENSEN & KONRADSEN, 2017, 9**). Selvom der ikke er så meget forskning indenfor området, så lader VR til at kunne benyttes indenfor affektiv læring. Træningssimulationer for affektiv læring er mindre afhængige af ydre enheder, som inkluderer kropsbevægelse i simulationen og mere afhængige af evnen til at vække følelser hos brugeren. Hvis en følelsesmæssig respons kan opnås hos brugeren gennem lyd og billede, vil simuleringen kunne benyttes til affektiv læring. (**JENSEN & KONRADSEN, 2017, 9**).
VR-teknologi bliver allerede i dag brugt til behandling af fobier og til øvelse af stresshåndtering (**JENSEN & KONRADSEN, 2017, 11**). Det er også det, vi i gruppen forsøger at opnå ved vores produkt.

Der findes flere forskellige spillekonsoller, der kan simulere et hvilket som helst miljø. Til at teste simuleringer af netop vores spil, har vi benyttet en Oculus Quest - derfor vil vi beskrive, hvordan den virker og bruges.

4.2 Oculus Quest

En Oculus Quest er et alt-i-et-spillesystem lavet til VR - hertil hører et VR-headset samt to controllers, som vist på figur 6. Questen kan fungere uden kabel (som en stand-alone enhed), men kan også kobles til en computer via et kabel, og derved kører over computeren. Questen kan spore brugerens bevægelser igennem kameraerne indbygget i Questen, selvom den ikke er koblet til en computer. Brugeren vil på den måde få en følelse af at gå inde i det computersimulerede miljø. Headsettet har derudover indbyggede højttalere, som understøtter stereo med rumlig lyd.
De to controllers, der medfølger, spores af sensorer i headsettet. Hver controller har et joystick, tre knapper på frontsiden, en trigger på indersiden af controlleren, som kan nås med tommelfingeren og en gripper yderst, som kan nås med pegefingeren (**UNITY TECHNOLOGIES, 2020, "INPUT FOR OCULUS"**).

Figur 6: *Oculus Quest controllers*

Til en Oculus Quest hører to controllers til, hvoraf der kan interagere med joysticket, de tre knapper øverst, samt trigger og gripper, som fremgår (https://www.grover.com/nl-en/products/oculus-quest-vr-64gb).

Når brugeren af VR-brillen starter en applikation, skal vagtsystemet opsættes, så det forhindres, at spilleområdet bliver forladt. Der bliver skabt et virtuelt gitter i VR omkring brugeren, som skaber spilleområdet og på den måde sætter en grænse for, hvor brugeren kan befinde sig. Vagtsystemet

bliver kun vist idet, det opsættes, eller hvis brugeren er for tæt på at forlade spilleområdet. På trods af sporingsteknologien fra Oculus er i stand til at scanne alle genstande i rummet for spilleområdet, så vil den ikke advare om genstande eller møbler, brugeren kan bumpe ind i under spillet. Vi vil fortælle mere om, hvordan vagtsystemet præcist opsættes i afsnit 7.0 under brugervejledningen.

5.0 Unity

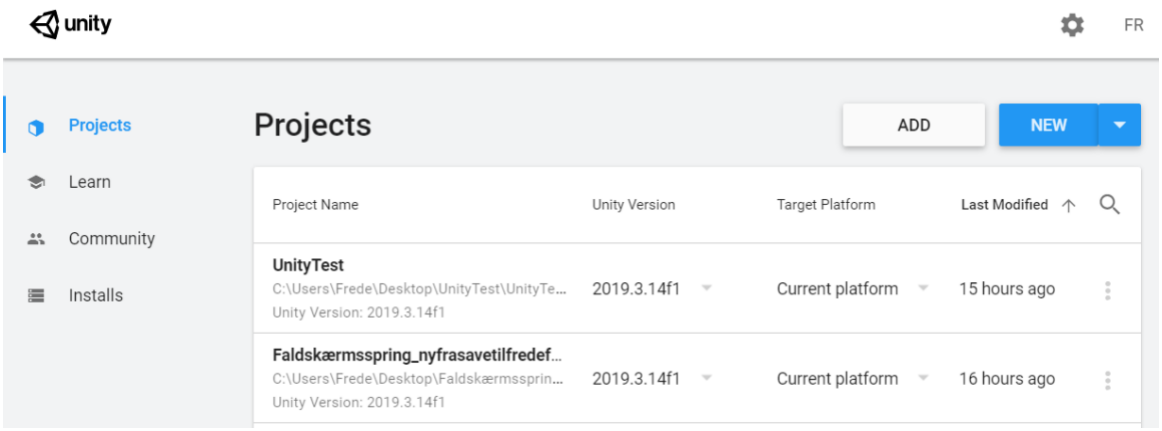
I dette afsnit vil der blive redegjort for Unity som spilmotor – herunder vil Unity Hub, Unity Editor, Collaborate Feature, Assets, Scripts og GameObjects med mere blive beskrevet, da det er dele af softwaren, vi kommer til at bruge meget undervejs.

Unity er én af mange spilmotor platforme, der har denne essentielle ramme for udviklingen af spil eller andet i både 2D og 3D – som blandt andet understøtter udvikling af virtual realityprogrammer. Herudover giver Unity os mulighed for at downloade, importere og samle assets, oprette eller importere animationer, skrive kode hvorved vi kan interagere med de oprettede objekter med mere. Som sagt blev Unity vores foretrukne platform til udviklingen af et VR-spil, da det er meget brugervenligt, samt er den fulde version gratis at downloade. Udviklingen af vores 3D-spil sker på forskellige computerenheder, hvoraf vi benytter os af Unitys egen Collaborate feature, som gør det muligt for os at hente den nyeste version af spillet ned uafhængigt af, hvem der har lavet de nyeste ændringer.

5.1 Unity Hub

Unity Hub er en applikation i sig selv, som gør det nemt at finde, downloade og overskue sine projekter og installationer. Det er Unity Hub, der skal bruges for at oprette et nyt projekt/spil - her vælges også, hvorvidt der skal bygges et 2D- eller 3D-spil (**UNITY TECHNOLOGIES, 2020, "STARTING UNITY"**). Ved valg af projektskabelon gør vi brug af Universal Render Pipeline, som er en helt ny skabelon, der giver mere optimeret grafik, hvilket er godt ved udvikling med VR (**UNITY TECHNOLOGIES, 2020, "UNIVERSAL RENDER PIPELINE OVERVIEW"**). Når der oprettes et nyt projekt besluttes der, hvorvidt det nye projekt skal opbygges fra start af i en helt tom scene, eller om det skal videreudvikles fra en prædefineret version. Vi har valgt at opbygge vores projekt fra en helt tom scene.

På figur 7 ses et skærmbillede af applikationens startside. Under "Projects" ses navnene på de projekter, som er gemt på computeren – her ses også hvilken version af Unity, der er blevet brugt til at oprette projektet, som i vores tilfælde er 2019.3.14f1 (**UNITY TECHNOLOGIES, 2020, "UNITY HUB"**).



Figur 7: *Unity Hub*

Unity Hub bruges til at oprette nye projekter samt finde, downloade og overskue sine andre projekter og installationer (billedet er et skærmbillede af eget projekt).

I selve Unity Hub findes og installeres supplerende moduler til eksisterende installationer af den editor, der er downloadet. Eksempelvis har vi skulle installere Android Build Support-modulet til den version af Unity, vi benytter, for at kunne tilgå vores Oculus Quest via Unity Editor. Derudover har vi installeret Unitys Standard Assets samt Visual Studio-modulet til det pågældende styresystem (for Mac, Linux eller Windows). Disse moduler kan enten downloades med det samme, når Unity Hub downloades, eller det kan gøres på et senere tidspunkt (**UNITY TECHNOLOGIES, 2020, "UNITY HUB"**).

Fra Unity Hub kan der klikkes på et af sine projekter og derved tilgå Unity Editor, hvor der kan bygges og ændres i scenerne, som er det, selve spillet kører fra.

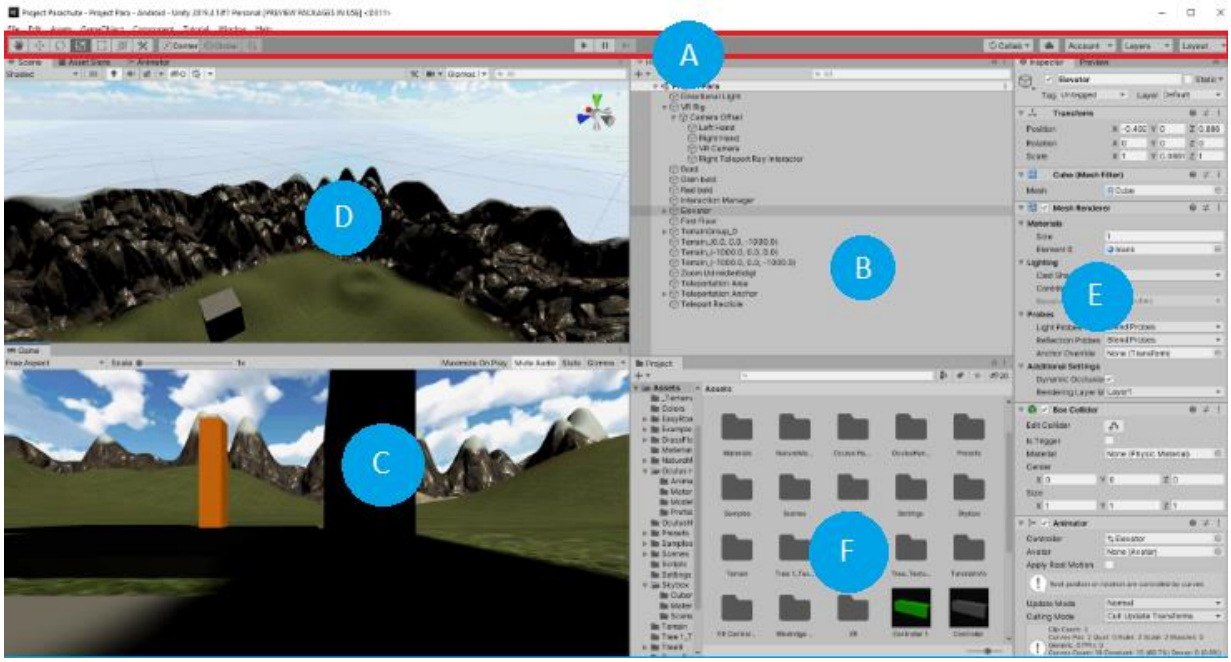
5.2 Unity Editor

Inden vi benytter os af Unity Editoren sætter vi indstillingerne i projektet op til at være kompatibel med VR, således projektet blandt andet understøtter brugen af et stand-alone VR-headset, som er det, vi arbejder med. Til det benyttede vi et nyt plug-in udviklet af Unity, som kaldes XR-plug-in management. Dette nye plug-in gør det muligt at integrere forskellige platforme direkte, og på den måde kan vi nemt skifte mellem forskellige VR-værktøjer (**UNITY TECHNOLOGIES, 2020, "ABOUT THE XR PLUG-IN MANAGEMENT PACKAGE"**). I starten benyttede vi Oculus Integration, hvilket udelukkende understøtter Oculus headsets. Vi kunne i princippet godt bruge

Oculus Integration, men dette ville umuliggøre processen med at gøre brug af de VR-headsets som RUC har til rådighed, hvilket var muligt igennem XR-plug-in management. En anden fordel er, at Oculus Integration er udviklet af Third-party, hvorimod XR-plug-in management package er udviklet af Unity. Dette giver en større garanti for vedligeholdelse af programmet, idet plug-in'et i vores program bliver understøttet af Unity.
Hvis udviklingen af et program sker i et professionelt gøremål, hvor programmet er integreret til kun én platform, og programmet efterfølgende skal udvides til andre platforme, vil det være en tidskrævende og dyr proces. Vi besluttede derfor at starte helt forfra med det nye plug-in, da vi fandt det hensigtsmæssigt at være kompatibel med flere forskellige platforme. Det viste sig at være en god beslutning, da vi fik muligheden for at benytte os af faciliteterne på RUC, hvor vi i stedet for at bruge et stand-alone VR-headset kunne videreudvikle med en HTC Vive. En HTC Vive fungerer ligesom Oculus Questen bortset fra, at denne VR-brille ikke fungerer som stand-alone enhed, men er afhængig af et kabel sat til computeren for at kunne køre et projekt.
Vi videreudviklede med en HTC Vive, da vi kom på RUC, eftersom RUC havde det fulde setup med computer tilsluttet med betydeligt bedre hardware, hvilket var virkelig praktisk og letanvendeligt. Der testede på samme tid det nye XR-plug-in, vi havde implementeret, som heldigvis virkede upåklageligt. Vi vil komme nærmere ind på, hvordan vi har implementeret XR-plug-in'et i Unity i afsnit 6.3.

Når der skal laves ændringer i et projekt, gøres det via Unity Editor. Hele editoren består af flere små vinduer, som kan rykkes rundt på i forhold til, hvad den enkelte finder overskueligt. Hvis der tages udgangspunkt i det primære vindue, som er repræsenteret i figur 8, så er der seks vigtige vinduer. Øverst ved den rektangulære, røde firkant ses ikonet A, som er værktøjslinjen. Til venstre i firkanten ses nogle forskellige værktøjer, der gør det muligt at ændre i scenen samt i de objekter, der måtte være i scenen. I midten af værktøjslinjen ses tre knapper, som gør det muligt at henholdsvis starte og stoppe. Til højre i værktøjslinjen er knapperne for Collaborate feature, som muliggør at push up og pull down ændringer - det er her vi også kan se versionshistorikken. Til sidst i højre side i værktøjslinjen har vi Layout. Layout er forskellige standardindstillinger, som kan bruges til at bygge vinduet af Unity Editoren op (**UNITY TECHNOLOGIES, 2020, "UNITY'S INTERFACE"**). Vi benytter en modificeret standardindstilling: Tall, således passer den bedre til

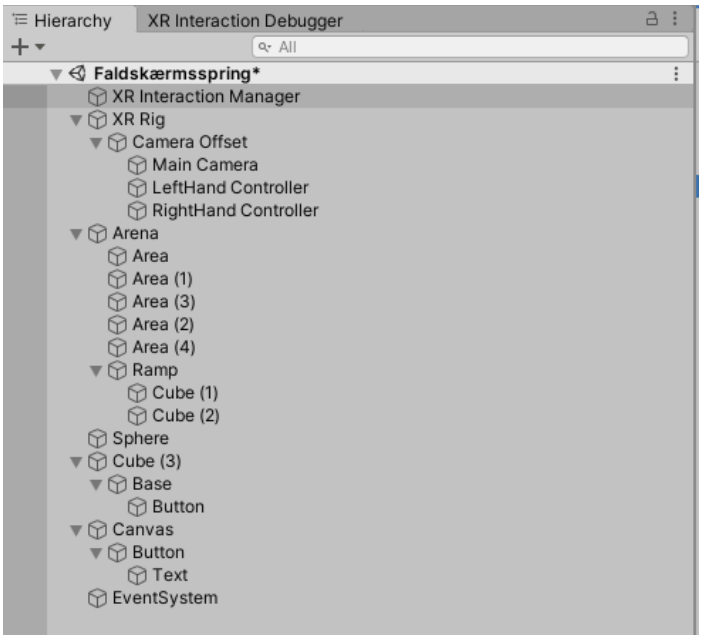
vores behov. Alle vinduerne kan trækkes henholdsvis til siderne eller op og ned, således der kan ses enten mere eller mindre af det valgte vindue.



Figur 8: *Unity Editor window*

*De seks vigtigste vinduer i Unity editoren (billedet er et skærmbillede af eget projekt med modifikationer inspireret fra Unitys egen hjemmeside) (**UNITY TECHNOLOGIES, 2020, "UNITY'S INTERFACE"**).*

B-ikonet på figur 8 repræsenterer Hierarchy vinduet. Det vindue indeholder en liste af samtlige GameObjects i den scene, der arbejdes i på nuværende tidspunkt. GameObjects bliver beskrevet i afsnit 5.5. De GameObjects, der måtte være i Hierarchy, har en hierarkisk anordning, hvoraf Unity arbejder med et begreb, der kaldes "Parenting". Parenting gør det muligt for nogle GameObjects at blive et "parent" af andre GameObjects, som så betegnes som "childs". Et child af en parent nedarver nogle egenskaber fra parent og ændrer sig relativt til parent og ikke selve spil verdenen. I vores VR-Spil benytter vi også parent og childs. På figur 9 ses GameObjectet XR Rig som en parent - det er dét GameObject, som bliver påvirket, idet spilleren flytter sig rent fysisk. Camera Offset er så et child af XR Rig, som yderligere indeholder tre childs: Main Camera, LeftHand- og Righthand Controller. Det resulterer i, at alle child GameObjects følger med, når spilleren flytter sig (**UNITY TECHNOLOGIES, 2020, "THE HIERARCHY WINDOW" ; TUTORIALSPOINT, 2020**).



Figur 9: *Hierarchy*

Et udsnit af vores Hiearchy med dertilhørende GameObjects og deres childs. (billedet er et skærmbillede af eget projekt).

Når programmet køres, kan det ses i spilvinduet, der repræsenterer C på figur 8. Det endelige produkt, som kan ses i VR-brillen, vises også her i runtime. Det eksakte billede, der vises, er en afspejling af scenekameraets synsvinkel. Når start-knappen fra værktøjslinjen benyttes, er det i dette vindue, programmet køres (**UNITY TECHNOLOGIES, 2020, "UNITY'S INTERFACE"**).

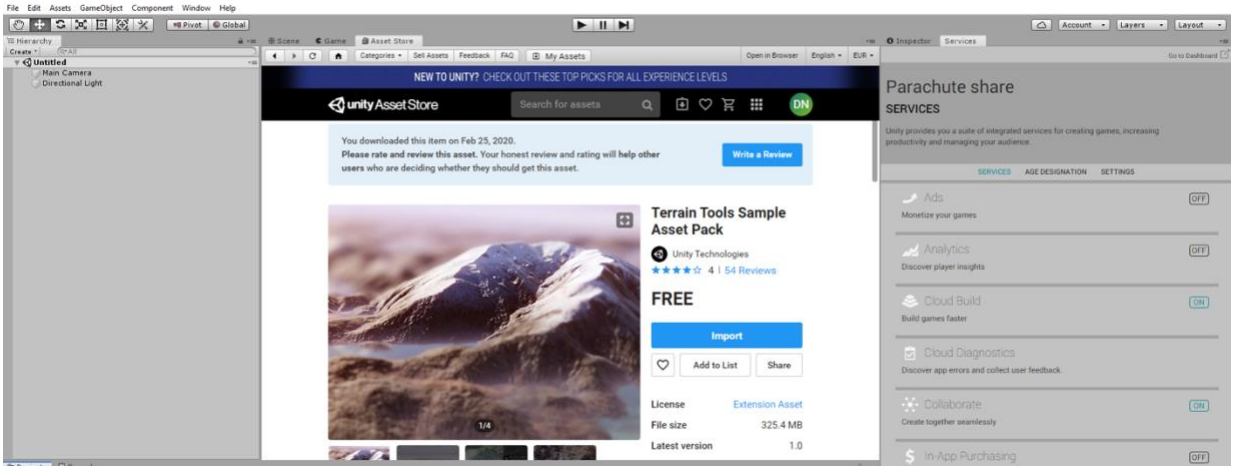
Ikonet D på figur 8 er en repræsentation af scenen. Scenen er her, hvor der kan redigeres og navigeres rundt i programmet. Scenen kan blive repræsenteret i henholdsvis 2D og 3D, hvoraf vores projekt bliver vist i 3D-perspektiv.

E-ikonet repræsenterer Inspector vinduet på figur 8. Her kan de forskellige komponenter og variable for et givent GameObject ses og manipuleres. Eksempelvis kunne det i Inspectoren fremgå, at Rigidbody er tilføjet som komponent til et givent GameObject. Ved blandt andet RigidBody-komponenten er der variabler, som kan manipuleres, og som ændrer, hvordan GameObjectet opfører sig inde i spillet. Ændringer i variablerne i komponenten til GameObjectet ændrer ikke på det underliggende script – dette muliggør at benytte det samme script, men stadig have forskellige værdier af variablerne på forskellige GameObjects (**UNITY TECHNOLOGIES, 2020, "UNITY'S INTERFACE**).

Det, der kaldes projektvinduet, ses ved ikonet F på figur 8. Her er de importerede biblioteker af Assets til det givne projekt repræsenteret, og de kan frit benyttes i projektet (UNITY TECHNOLOGIES, 2020, "UNITY'S INTERFACE").

5.3 Assets

Et Asset er en repræsentation af et eller flere objekter, som kan bruges i ens projekter i Unity. Unity har selv en masse standard assets, men der kan også importeres egne filer. Ydermere kan der hentes assets fra Asset Store, hvor vi eksempelvis har importeret Terrain Tools Sample Asset Pack til brug i vores eget projekt (UNITY TECHNOLOGIES, 2020, "ASSET WORKFLOW"). På figur 10 ses Asset Store i et vindue i Unity Editor.



Figur 10: *Asset store*

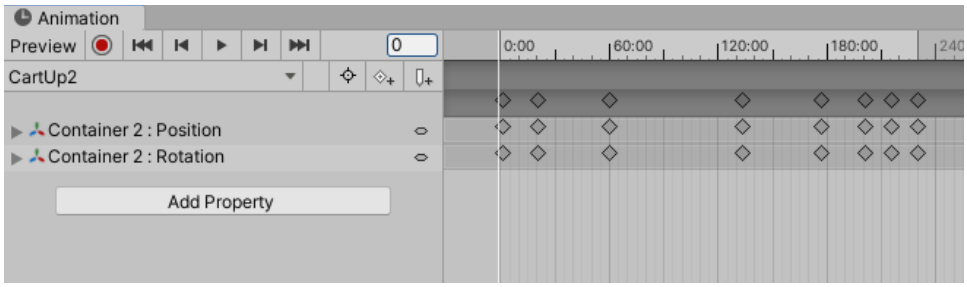
Asset Store kan ses direkte i Unity, hvor det er muligt at importere de ønskede assets. Vi benyttede os især af Terrain Tools Sample Asset Pack i projektet (billedet er et skærmbillede af eget projekt).

Når der importeres et asset til Unity, forbliver det uforandret, indtil det eventuelt modificeres. Som sagt har Unity selv en masse standard assets, som langt de fleste brugere af Unity benytter sig af; eksempelvis er der kameraer, som kan tilføjes til scenen, som gør, at synsvinklen fra kameraet, når spillet er startet, kan vises, og præciseres, alt efter hvor outputtet fra spillet ønskes at forekomme fra (UNITY TECHNOLOGIES, 2020, "COMMON TYPES OF ASSETS").

Der er forskellige typer af assets, som f.eks. billedfiler, animationer og lydfiler. Nogle typer af assets kan tilmed skabes i programmet eksempelvis en Animator Controller. En Animator

Controller muliggør at samle animationer, som skal bruges på et objekt - på denne måde skabes et overblik over de animationer, der bliver brugt. Selv hvis objektet kun skal indeholde én animation, er det stadig nødvendigt at oprette en Animator Controller i Assets. Unity har et Animator Controller vindue, hvor sammenhængene mellem animationerne i programmet for hvert objekt kan åbnes og ses – et eksempel på en Animator Controller kan ses på figur 12. Selve Animatoren bliver tilføjet til et objekt ved at oprette den som en komponent under det ønskede objekt (**UNITY TECHNOLOGIES, 2020, "ASSET WORKFLOW"**).

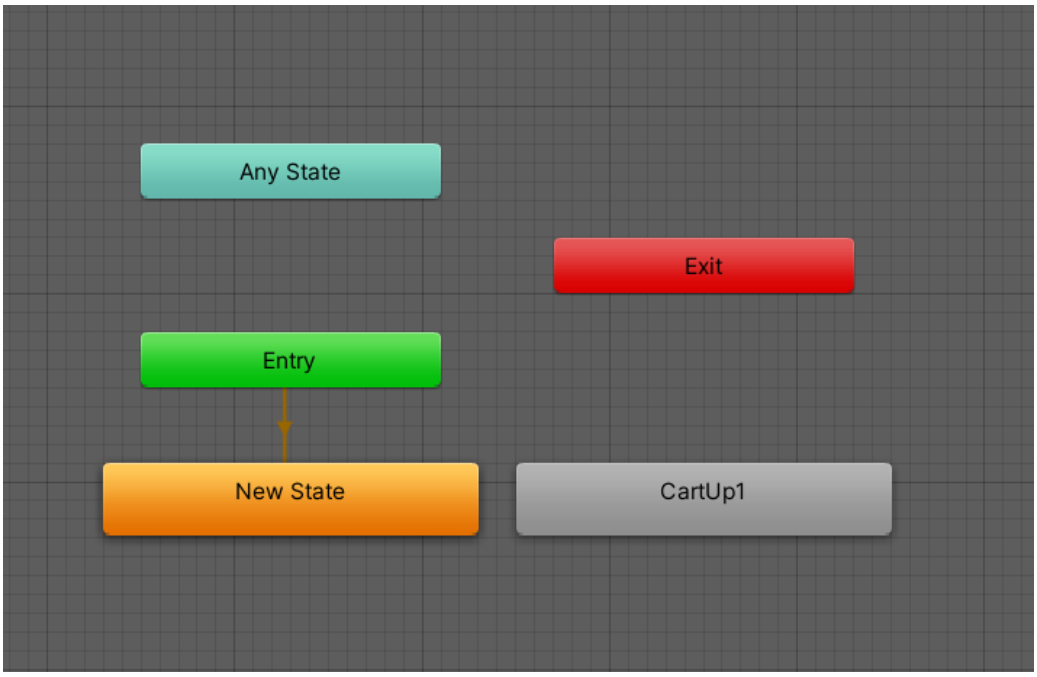
I vores program gør vi brug af bevægelse af objekter, hvilket forekommer igennem animationer. Vores animationer består af at ændre x-, y- og z-værdien af objektets position, skalering og rotation, ved at koble positionerne sammen med tid. Dette betyder, at vi tager et objekt og giver den en startværdi og en starttid og derefter vælger et punkt, som objektet skal være på efter en vis mængde tid. Dette betyder, at Unity bevæger objektet mellem punkterne fra start til slut. Hvert punkt repræsenterer koordinater for x-, y- og z-værdi i vores 3D-verden. På figur 11 kan det ses, at vi har tilføjet flere punkter, som vores objekt passerer over en bestemt tid ved hjælp af animationen. Det er programmøren af animationen, der sætter tidsrammen for animationen.



Figur 11: *Animation*

Animation er en indbygget del af Unity hvor der kan tilføjes egenskaber, der over en periode kan ændre sig - i dette tilfælde positionen og rotationen (billedet er et skærmbillede af eget projekt).

Når animationen er lavet, skal der derfor oprettes en Animator Controller, hvor de valgte animationer skal tilføjes. Dette har til formål at muliggøre en serie af flere forskellige animationer, som kan arbejde sammen. Derudover muliggør Animator Controlleren også at skifte mellem forskellige stadier af animationer.



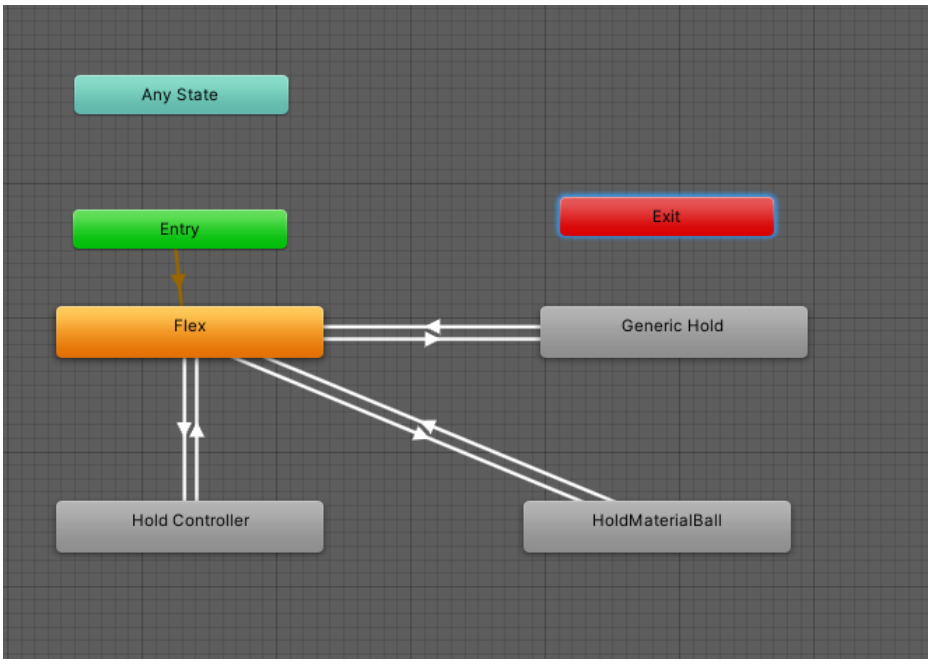
Figur 12: *Animator Controller State Machine*

Animatoren Controlleren består af en række stadier, som et objekt kan være i (billedet er et skærmbillede af eget projekt).

Animator Controlleren er bygget op omkring henholdsvis "Entry" og "Exit", hvilket er perioden, hvor animationerne er aktive. Fra "Entry" har vi lavet et "New State" hvilket er et tomt stadie, som har til formål at sætte animator controlleren i venteposition, således der kræves en interaktion i spillet for at afspille animationen "CartUp1". I vores VR-verden har hver knap en animator controller, som styrer animationen - hver animator controller bliver kaldt i det dertilhørende script, og animator controlleren starter derved animationen. Hvis vi tager udgangspunkt i figur 12, ville animationen her være "CartUp1", som bliver startet.

Afhængig af om de animationer, der forekommer i Animator Controlleren, er gentagende eller ej, kan Animator Controlleren sættes op på forskellige måder. I tilfældet med vores "CartUp1" er dette en animation, der kun skal forekomme en enkelt gang. Dette i modsætning til animationerne af vores hænder, som forekommer hele tiden, eftersom hænderne hele tiden skal være synlige. Derfor er Animator Controlleren opsat anderledes til vores hænder, hvor der i stedet for at være et tomt stadie er et stadie, som hedder "Flex". Flex stadiet har to funktioner, henholdsvis animationer af hænderne, hvis de er i kontakt med et objekt, og hvis de ikke er. Den første funktion, Flex stadiet

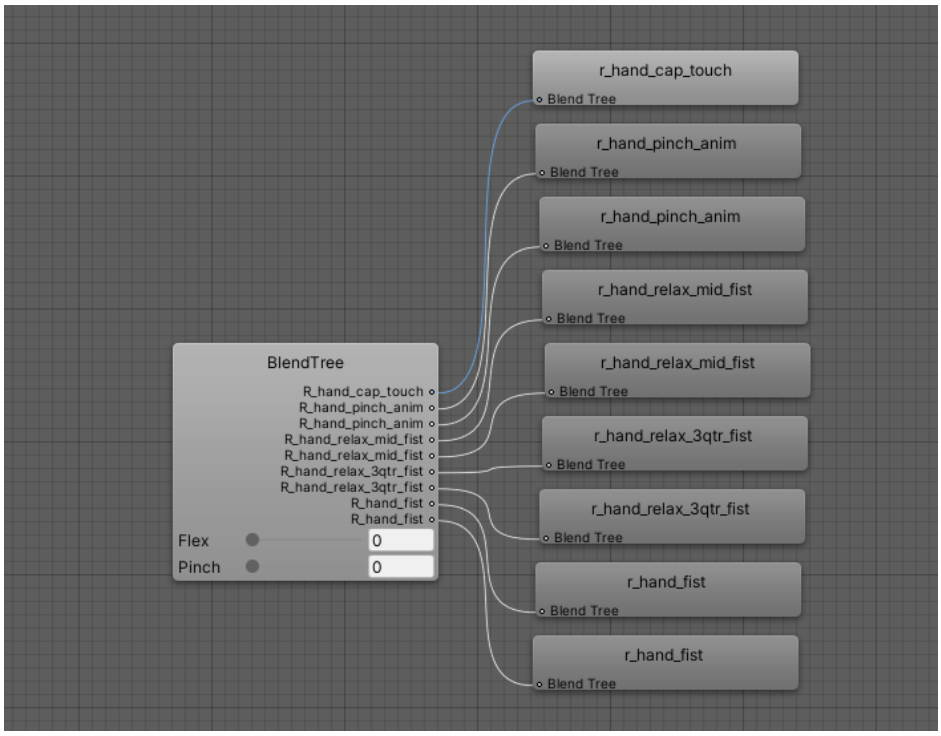
har, er illustreret på figur 13. Denne funktion er, når hænderne er i kontakt med et objekt som for eksempel en kugle. Imens denne kontakt forekommer, skal der køre en bestemt animation.



Figur 13: *Animator Controller til spillerens hænder*

Animator Controlleren kan sende stadiet rundt imellem flere forskellige stadier afhængig af, hvad hænderne gør (billedet er et skærmbillede af eget projekt).

Den anden funktion, Flex stadiet har, er illustreret på figur 14. Denne funktion er, når hånden ikke er i kontakt med noget objekt. I dette tilfælde vil hånden bruge animationen, der forekommer gennem "Blend Tree". "Blend Tree" er en måde, hvorpå forskellige animationer kan blandes sammen, således flere forskellige input kan have indflydelse på, hvordan hånden ser ud.



Figur 14: *Blend Tree*

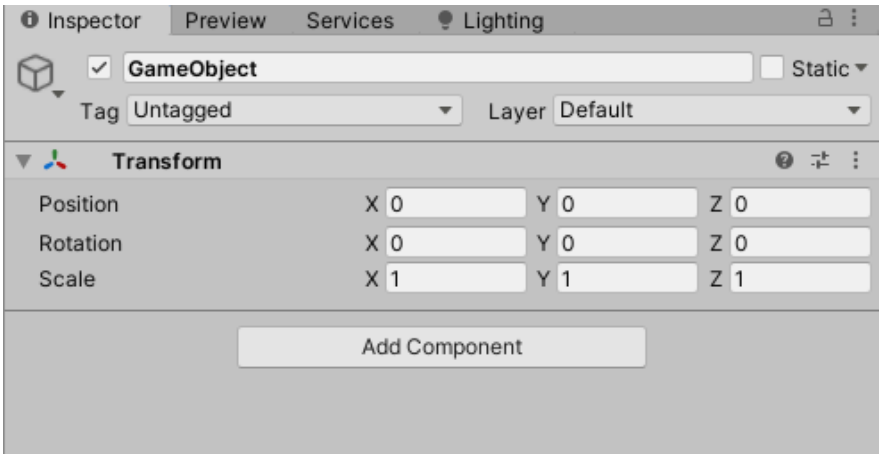
Blend Tree blander flere animationer sammen og viser denne blanding afhængig af Flex og Pinch input (billedet er et skærmbillede af eget projekt).

Det fungerer på den måde, at hver input faktor (gribber og trigger), der indgår, har en værdi mellem 0 og 1, som i vores tilfælde bliver afgjort i forhold til, hvor meget spilleren trykker på henholdsvis trigger og gribber. Disse animationsværdier bliver kaldt i et script, hvilket vil blive uddybet i afsnit 6.3 (**UNITY TECHNOLOGIES, 2020, "THE ANIMATOR CONTROLLER"**).

De forskellige animationer på figur 14 er vist i højre side, hvor hver animation har sin egen kasse, som hver for sig er koblet på Blend Tree. Derfor kan der i venstre side i Blend Tree ses listen med animationer, hvorunder denne liste fremgår to input faktorer, der afgør hvilken blanding af animationer, der skal vises. På figuren står "Flex" og "Pinch" til 0, hvilket er sin egen animation, hvor hånden er åben. Derfor er det blandingen af "Flex" og "Pinch", der afgør hvilke animationer, der skal blandes sammen for at få den ønskede position af hænderne.

5.4 GameObjects, Components og Variables

Udviklingen af al interaktivitet samt gameplay (det som spilleren gør) er baseret på tre essentielle byggesten: GameObjects, Components og Variables (**UNITY TECHNOLOGIES, 2020, "CREATING AND USING SCRIPTS"**). Ethvert objekt i et spil i Unity er et GameObject. Dette kan eksempelvis være en person i spillet, lys eller rekvisitterne (eventuelt importeret fra Asset Store). Et objekt har i sig selv ingen egenskaber, men objektet kan tilegnes egenskaber ved brug af komponenter samt variabler (**UNITY TECHNOLOGIES, 2020, "GAMEOBJECTS"**). Det betyder, at et GameObject ingen funktionalitet har foruden et eller flere komponenter. Når et GameObject oprettes, vil det derfor betegnes som et tomt objekt, som hverken kan ses eller har nogen former for egenskab, før der tilføjes komponenter. Alle objekter har dog på forhånd en Transform komponent, som muliggør ændringer i objektets x-, y og z-koordinater i forhold til både position, rotation og skalering i programmet – denne komponent kan ikke slettes fra et GameObject. Det er også denne Transform komponent, der tillader parenting konceptet mellem GameObjects. Komponenten er repræsenteret i Inspektoren som ved standardindstilling på figur 15 (**UNITY TECHNOLOGIES, 2020, "GAMEOBJECT"**).



Figur 15: *Component i Inspector*

Ethvert GameObject har Transform komponenten – selv når de er "tomme". Transform komponenten bestemmer GameObjectets position, rotation og skalering i forhold til både x-, y- og z-koordinatet (billedet er et skærmbillede af eget projekt).

Der findes GameObjects, der er lavet på forhånd - eksempler på det er kameraer, hvor der er tilhørende komponenter til, og primitive objekter. Primitive objekter er de typisk brugte former af

objekter så som kugle, kasse og kapsel, som kan sættes direkte ind i scenen og bruges, som de er, og efterfølgende kan de modificeres, hvis det ønskes. De kan også bruges som midlertidige objekter til at teste eventuelle funktioner hurtigt (UNITY TECHNOLOGIES, 2020, "PRIMITIVE AND PLACEHOLDER OBJECTS").

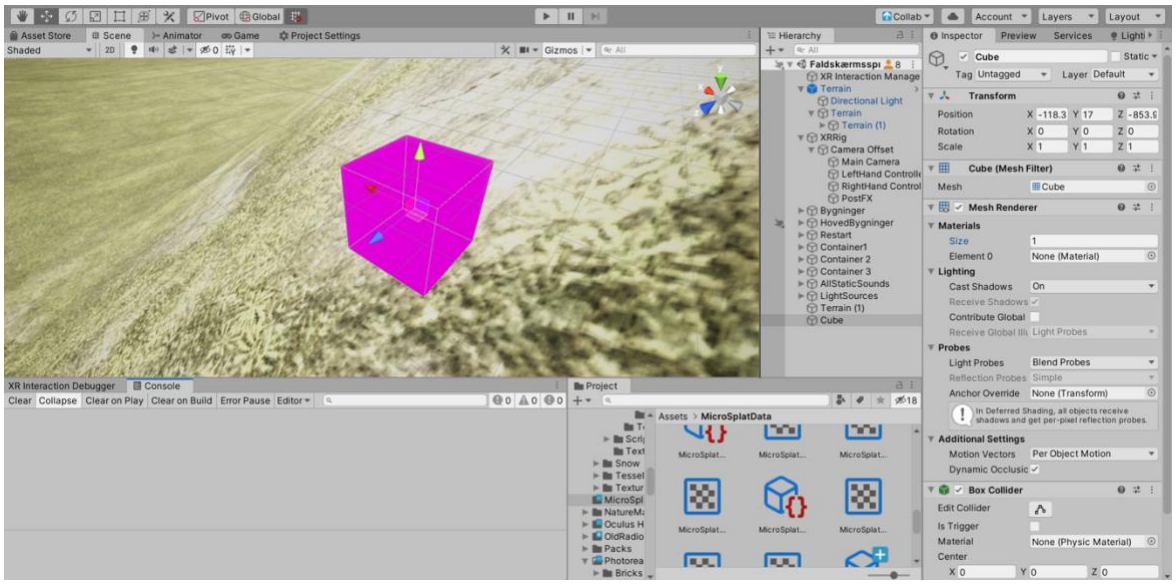
De forskellige komponenter tilhørende et GameObject kan ses ved at klikke på et af objekterne. Derved kan komponenterne ses, redigeres og eventuelt tilføjes ude i Inspectorvinduet til objektet. En variabel er ligesom komponenterne, noget der bliver tilføjet til GameObjects. Hvis en variabel i et script er sat til at være public kan den tilgås gennem editoren, mens private variabler kun kan tilgås gennem scripts.

GameObjects er som udgangspunkt ikke synlige, da de er tomme, men synligheden kommer ved tilføjelse af Meshes, Materials, Shaders og Textures.

5.5 Meshes, Materials, Shaders og Textures

Når et nyt tomt GameObject laves, er det ikke til at se inde i spillet. Der er intet visuelt, det har ingen form, ingen materialer, ingen shaders osv. Det eneste, det har, er en position, som er allokeret ved dets Transform komponent.

For at tilføje en form til GameObjectet findes der en indbygget klasse i Unity, der hedder Mesh (Filter). Klassen tager et 3D-objekt som input, og tildeler derved formen fra 3D-objektet til GameObjectet. Figuren vises stadigvæk ikke i spillet, da selve formen endnu ikke er blevet visuelt gengivet – til det benyttes en anden klasse: Mesh Renderer. GameObjectet kan nu ses i programmet, som set på figur 16, men det bliver per standardindstilling set som en stor lyserød klods. Dimensionerne kan som vist ikke antydes i et GameObject, som udelukkende er bestemt af Meshes. Under Mesh Renderer klassen kan dette ændres ved at tilføje et Material. Det er Materials, der bestemmer, hvordan overfladen af Objectet bliver gengivet. Unity har et indbygget standard grå Material (Albedo), men det er muligt at lave sine egne tilpassede Materials – det kan være alt fra simple ensfarvede Materials til komplekse billeder og teksturer. Egenskaberne for visningen af et material er bestemt af de shaders, der er tilføjet til materialet.



Figur 16: *Meshes*

Nyoprettet GameObjecemed Meshfilter og Mesh render komponenterne uden Material (billedet er et skærmbillede af eget projekt).

Shaders er scripts, der kan indeholde algoritmer, som ændrer, hvordan hver eneste pixel bliver gengivet i scenen. Det kunne for eksempel være en algoritme som re-orienterer mønsteret en smule for hver pixel. Det benyttede vi blandt andet på græsset i vores program for at undgå for meget gentagelse i mønsteret og derved bryde det ensformige udseende for materialerne. Denne proces bliver yderligere beskrevet i afsnit 6.4 om spilverdenen (**UNITY TECHNOLOGIES, 2020, "MESHES, MATERIALS, SHADERS AND TEXTURES"**).

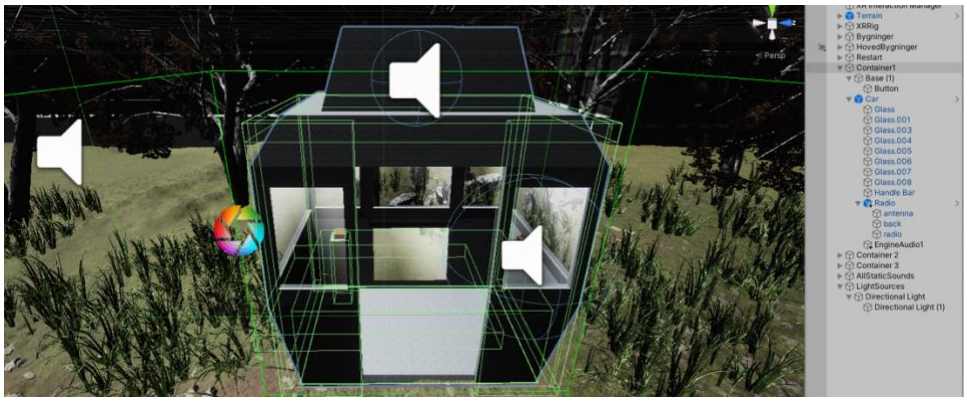
Til sidst er der Textures, som er bitmap billeder. Textures bliver tilføjet til objekter, der benytter materialer. Det er objektets Material, der benytter textures: for eksempel gennem brug af et Normal Map, kan der skabes dybde i objektet, så der forekommer små skyggeområder. Dette gøres for at skabe illusionen af at selv store, flade planer har en 3D effekt (**UNITY TECHNOLOGIES, 2020, "MESHES, MATERIALS, SHADERS AND TEXTURES"**).

5.6 Prefab

Prefab er et system i Unity, hvor skabeloner for et objekt med tilhørende komponenter, egenskabsværdier og child GameObjects kan oprettes og genbruges som et Asset. Prefab Assets fungerer som en skabelon, der frit kan deles mellem Unity projekter, hvorunder en del kan

importeres fra Asset Store og hentes ned til et Unity projekt. Et Prefab kan for eksempel være et pariserhjul, som består af blandt andet flere GameObjects, materialer og shaders, der til sammen udgør det fulde pariserhjul og dens udseende. GameObjects er blandt andet vognene i pariserhjulet. Ved at bruge Prefabs i scenen får vi mulighed for at redigere flere objekter på én gang, hvis der er blevet lavet flere instanser af et Prefab. Det vil sige, hvis der eksisterer flere kopier af et Prefab, vil alle kopierne blive synkronisereret med det oprindelige ud fra enkelte ændringer. Dette kunne for eksempel være, hvis vi ændrede farven på det oprindelige Prefab af et pariserhjul, så ville kopierne af netop dette Prefab blive synkroniseret med det oprindelige og dermed også få ændret farve. Herfor undgår vi ekstra arbejde ved at kopierer et Prefab et ønsket antal gange. Det betyder, at vi får mulighed for at genbruge GameObject med specifikke egenskaber. Det smarte i at benytte Prefabs er, at i stedet for at gå ind manuelt og ændre farve på alle knapper enkeltvis i alle GameObject i et Prefab, kan vi ændre farven på ét GameObject fra Prefabbet, der ændrer alle knapperne, som er med i det Prefab. Det gjorde vi brug af undervejs - vi eksperimenterede med, hvordan de knapper, der trykkes på i pariserhjulvognen, så ud med forskellige materialer. Da vi fandt den bedste, opdaterede vi Prefabbet med det materiale, således alle knapperne så ens ud (**UNITY TECHNOLOGIES, 2020, "PREFABS"**).

Dette betyder dog ikke, at alle forekomster af et bestemt Prefab nødvendigvis skal være identiske – det kan nemlig lade sig gøre at override indstillinger på Prefabs, hvis det ønskes at lade nogen skille sig ud fra de andre (**UNITY TECHNOLOGIES, 2020, "PREFABS"**). Eksempelvis har vi importeret et Asset, der indeholder et helt pariserhjul, og dette pariserhjul er gemt inde i et Prefab. Prefabbet indeholder forskellige GameObjects så som de vogne, der hænger på pariserhjulet. Vi har brugt en vogn fra pariserhjulet som en elevator, der har til formål at få spilleren op til forskellige stadier af højder. Denne pariserhjulvogn er illustreret på figur 17, som viser hvordan pariserhjulvognen ser ud, efter vi ændrede den til at passe til vores behov.



Figur 17: *Pariserhjulvognen*

Et Prefab som vi har lavet ved at ændre på et andet Prefab (billedet er et skærmbillede af eget projekt).

Til at starte med fjernede vi døren som lukkede pariserhjulvognen, for at give spilleren fri adgang til at gå ind og ud af pariserhjulvognen. Derefter ændrede vi på dimensionerne for, at den ikke skulle være for stor. Det sidste, vi gjorde, før vi lavede pariserhjulvognen til sit eget Prefab, var at ændre på materialet og teksturen, således den så ud, som vi gerne ville have den til.
Efter vi var tilfredse med udseendet af Prefabbet, tilføjede vi animationer, lydeffekter og script til GameObjectet som indeholder Prefabbet. Således kom vi et skridt nærmere det endelige produkt, som vi vil beskrive i næste afsnit.

6.0 Programmet

Med dette afsnit vil først og fremmest blive redegjort for hvad scripts er generelt, hvorefter der gennem et diagram vil blive forklaret hvordan nedarvningshierarkiet i Unity fungerer. Herefter vil de scripts og implementeringer af kode, der er blevet anført i projektets program, blive gennemgået, hvor der til slut vil blive rundet af med hvorledes spilverdenen er blevet skabt og dets empiriske grundlag.

6.1 Generelt om scripts

En essentiel del i et vilkårligt program i Unity er de scripts, der bliver skrevet. Unity understøtter C#, som er det kodesprog, vi har valgt at skrive scripts i. Scripts muliggør blandt andet, at applikationen kan reagere på bestemte input fra brugeren (**UNITY TECHNOLOGIES, 2020, "SCRIPTING"**).

Når et nyt script oprettes i Unity, ser det ud som vist på figur 18. Det nyoprettede script har fra start en public class, der er opkaldt efter det GameObject scriptet er tilknyttet til. Navnet på klassen, som ved eksemplet på figur 18 er "PlayerAtttach", bliver efterfulgt af et kolon samt MonoBehavior. Kolonet indikerer at klassen er afledt af MonoBehavior, som er en indbygget klasse i Unity. Den nyoprettede klasse kaldt PlayerAtttach, nedarver derfor metoder fra MonoBehavior klassen.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  {
       0 references
6      public class PlayerAtttach : MonoBehaviour
7      {
8          // Start is called before the first frame update
           0 references
9          void Start()
10         {
11
12         }
13
14         // Update is called once per frame
           0 references
15         void Update()
16         {
17
18         }
19     }
20 }
```

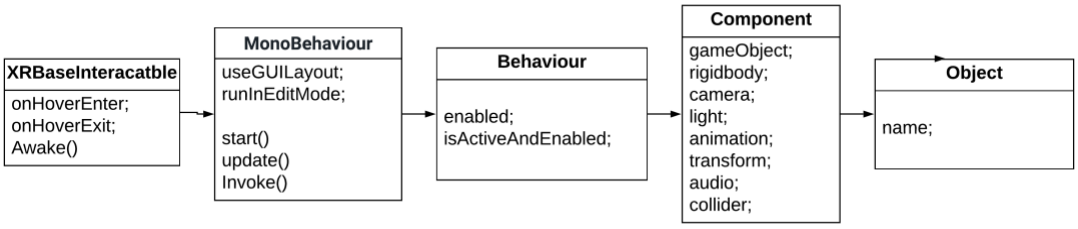
Figur 18: *Nyoprettet script*

Metoderne Start og Update er en del af standardindstillingerne for et nyt script. De to metoders funktion er tilmed beskrevet i en kommentar ovenfor hver af dem (billedet er et skærmbillede af eget projekt).

Under PlayerAtttach klassen i scriptet ser vi metoderne Start og Update, som automatisk bliver oprettet sammen med scriptet. Start metoden bliver kaldt som det første før gameplay begynder, og benyttes til enhver form for initialisering. Update metoden benyttes til at håndtere alt fra animation til bruger-input og bevægelse, alt som kræver en regelmæssig opdatering. Update metoden fungerer altså som et while-loop der bliver kørt igennem én gang per frame i spillet.
Udover Start og Update metoderne findes der også Awake metoden. Awake benyttes til at initialisere variabler før spillet starter. Metoden bliver kun kaldt en gang i scriptet levetid, ligesom startmetoden.

6.2 Nedarvningshierarki i Unity

XRBaseInteractable er den første klasse, vi bruger egenskaber fra. Det er ikke nødvendigvis alle scripts, der nedarver fra XRBaseInteractable – scripts kan også nedarve fra klasser senere i nedarvningskæden. XRBaseInteractable klassen arver en lang række egenskaber fra den tidligere klasse, som arver fra dens tidligere klasse, sådan fortætter det helt ned til Object klassen. For at gøre denne nedarvningskæde tydeligere, har vi lavet et UML-diagram vist i figur 19, som starter med XRBaseInteractable, for at visualisere rækkefølgen, hvor vi efterfølgende vil beskrive nogle af de vigtigste egenskaber for hver klasse.



Figur 19: *UML-diagram*

Klassediagram over nedarvningskæden i Unity ud fra vores program med metoder vi benytter i programmet.

Det er vigtigt at nævne, at alle metoderne i hver af klasserne ikke er beskrevet i diagrammet. Vi har kun medtaget de metoder, som vi benytter gennem hele programmet. En fuld liste af metoderne kan ses i Unitys dokumentation (**UNITY TECHNOLOGIES, 2020, "Class XRBaseInteractable","MonoBehaviour","Behaviour", "Component", "Object"**) eller i Visual Studio ved at trykke F12 på klassen.

XRBaseInteractable

Det første led i nedarvningskæden for vores program er XRBaseInteractable. Dette er en basisklasse, hvorfra funktionaliteten omkring interaktion med objekter til XR-interaktioner bliver nedarvet fra. Det er bindeleddet mellem to objekter, som enten har "Interactor" egenskab eller "Interactable" egenskab, hvilket dermed bestemmer hvilke objekter der kan interagere med andre objekter. Denne klasse er specifik for XR-udviklingsmiljøer (**UNITY TECHNOLOGIES, 2020, "Class XRBaseInteractable"**).

MonoBehaviour

MonoBehaviour er den vigtigste klasse i Unity, som alle klasser i Unitys scripts nedarver fra – ethvert Unity script siges at udvide MonoBehaviour klassen. Det kan ses på figur 18 på linje 5, at klassen "PlayerAtttach" udvider MonoBehaviour. Det er Monobehaviour klassen, der definerer den bestemte rækkefølge af udførsler af metoder (**UNITY TECHNOLOGIES, 2020, "MonoBehaviour"**).

Udover at scripts kan tilføjes til et GameObject, så har MonoBehaviour indbyggede metoder. Et eksempel på dette er Invoke metoden, som bruges til at kalde en metode afhængig af tid, før selve spillet starter. Den ville derfor ikke være mulig at benytte, hvis scriptet ikke nedarvede fra MonoBehaviour.

Efter MonoBehaviour kommer det andet led i nedarvningskæden, som er Behaviour, hvilken MonoBehaviour nedarver fra.

Behaviour

Behaviour er næste led i kæden af klasser, som har til funktion at aktivere eller deaktivere de forskellige components på et GameObject. En måde hvorpå der kan aktiveres eller deaktiveres kan være, når en handling forekommer. Eksempelvis kan en kollisionskomponent tilføjes, der tjekker for kollision mellem to objekter. Denne kollisionskomponent kan aktiveres, således der kan forekomme en anden handling. Komponenten bliver kun aktiveret hvis en bestemt handling forekommer – ved netop dette eksempel vil den derfor kun aktiveres, hvis der faktisk forekommer kollision. Ellers vil den være forekomme deaktiveret.

Klassen består kun af to metoder som vist på diagrammet på figur 19. Enabled metoden indebærer, at aktiverede behaviours stadigvæk opdateres og deaktiverede behaviours ignoreres og derved ikke opdateres. Den anden metode IsActiveAndEnabled tjekker både om GameObject er aktivt og dens script er aktiveret, hvis dette er tilfældet, returnerer metoden true (**UNITY TECHNOLOGIES, 2020, "Behaviour"**).

Component

Den næste del i nedarvningskæden er Component, som indeholder en lang række metoder, som påvirker GameObjects på forskellige måder. En komponent bestemmer, hvordan et GameObject opfører sig inde i spillet. På figur 19 kan de metoder, vi benytter i vores program, ses.
Det væsentlige at forstå her er, at ens scripts ikke direkte opretter en ny komponent, men at scriptet vedhæftes til et GameObject og på den måde bestemmer, hvordan det opfører sig og kan interageres med (**UNITY TECHNOLOGIES, 2020, "Component"**).

Object

Den næste og sidste del i nedarvningskæden er Object - denne klasse indebærer alle typer af objekter i Unity. Klassens overordnede funktion er at navngive, oprette og destruere objekter i Unitys motor. For eksempel er det Object klassen, som opretter et GameObject såsom kamera. Det kan tildeles nogle variabler, som tidligere beskrevet. De variabler vises i Inspectoren i Unity Editor's GUI, hvorfra værdierne for GameObject kan ændres ved hjælp af useGUILayout metoden fra MonoBehaviour klassen (**UNITY TECHNOLOGIES, 2020, "Object"**).

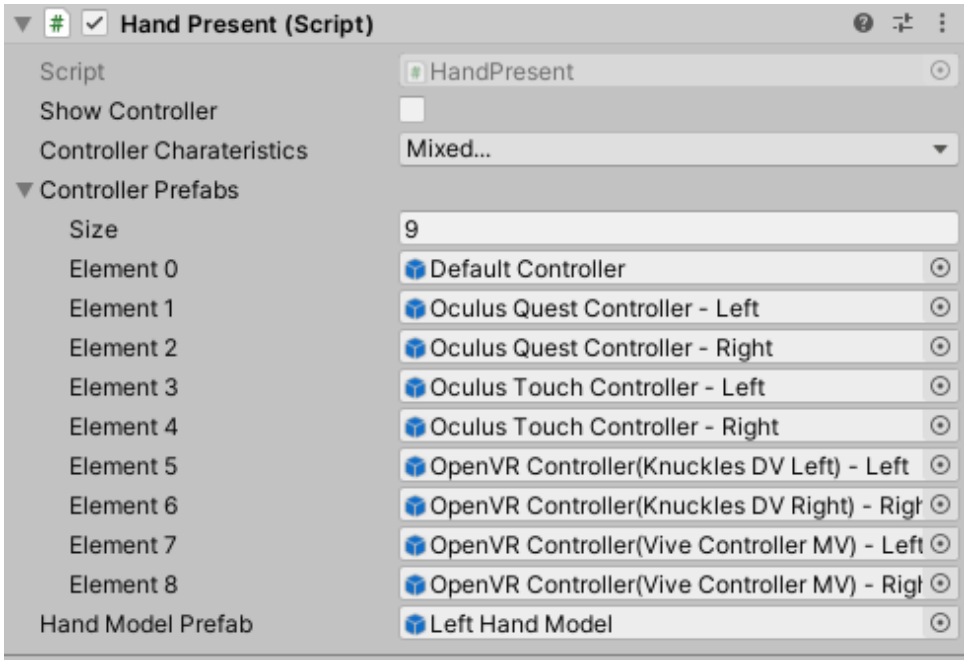
6.3 Scripts i projektet

For at tilegne de forskellige GameObjects egenskaber, har vi tilføjet komponenter i form af scripts til nogle af de GameObjects, vi har i spillet. Vi vil præsentere hovedfunktionerne for de forskellige scripts, hvoraf der vil indgå skærmbilleder, som viser de vigtigste dele af de forskellige scripts.

Handpresent

Vi bruger HandPresent scriptet til at vælge, om vi ønsker at have hænder eller de controllere, der passer til headsettet. Grunden til at spillet skal kontrollere, hvilket headset der bruges, er, fordi

headsettene har forskellige Controllerer, dog med de samme funktionaliteter. De forskellige udseender af controllere har vi fundet og importeret som eksisterende Prefabs, hvor vi har lagt det ind som en liste under et samlet Prefab for hver hånd - alle muligheder for udseende af hænder kan se på figur 20. Disse filer kan hentes fra YouTube-brugeren Valem via google drive, hvor linket dertil kan findes i kildelisten.



Figur 20: *Klasse HandPresent*

Alle de forskellige muligheder for hænder er repræsenteret af et Prefab (billedet er et skærmbillede af eget projekt).

Som nævnt tidligere vil vi her under HandPresent forklare, hvordan scriptet HandPresent arbejder sammen med Animator Controlleren, hvor "Flex" indgår. I scriptet HandPresent bruges UpdateHandAnimation, som kan ses på figur 21, til at beregne, hvilken værdi den har mellem 0 og 1, som i vores tilfælde bliver afgjort i forhold til, hvor meget spilleren trykker på henholdsvis trigger og Gribber. Desto længere knappen bliver trykket ind, desto højere værdi vil der forekomme. Denne værdi bliver så trukket ind til Animator Controlleren handAnimator, hvorfra animationen forekommer.

```
void UpdateHandAnimation()
{
    if (targetDevice.TryGetFeatureValue(CommonUsages.trigger, out float triggerValue))
    {
        handAnimator.SetFloat("Trigger", triggerValue);
    }
    else
    {
        handAnimator.SetFloat("Trigger", 0);
    }
    if (targetDevice.TryGetFeatureValue(CommonUsages.grip, out float gripValue))
    {
        handAnimator.SetFloat("Grip", gripValue);
    }
    else
    {
        handAnimator.SetFloat("Grip", 0);
    }
}
```

Figur 21: Værdi af trigger og gribber

Afhængigt af hvor meget henholdsvist trigger og gripper bliver trykket, bliver denne værdi sendt til Animator Controlleren, der styrer hånden. (billedet er et skærmbillede af eget projekt).

PlayerAttach

Vi har oprettet en klasse, vi kalder PlayerAttach, som har til formål at transformere brugeren af spillet til et Child af pariserhjulvognen. Dette script er tilføjet til GameObjectet Container1, som indeholder pariserhjulvogn. Scriptet kan ses i figur 22. Dette har vi gjort ved først at oprette en Collider, således at når vores XRRig kommer ind pariserhjulvognen, så bliver denne Collider aktiveret. Dette har vi gjort ved at sige, at hvis et GameObject kolliderer med Collideren og dette GameObject er XRRiget, så skal XRRiget så længe det kolliderer være et child af Container1. Dette har vi gjort, fordi Container1 har en animation som ændrer position, hvilket når pariserhjulvognen bevæger sig på x- og z-aksen ville bevæge sig væk under spilleren. Ved at gøre spilleren til et Child af pariserhjulvognen, vil animationen derfor også gælde for spilleren, som derfor følger animationen, og dermed undgår vi, at vognen bevæger sig væk under spilleren. Selvom spilleren bevæger sig sammen med vognen, har spilleren stadig mulighed for at bevæge sig rundt. Når spilleren forlader vognen og dermed ikke længe kolliderer med Collideren, bliver XRRiget ændret tilbage til sin egen Parent, som dermed ikke hører til noget andet.

```
private void OnTriggerEnter(Collider other)
{
    if (other.gameObject == Player)
    {
        Player.transform.parent = transform;
    }
}
```

```
10  private void OnTriggerEnter(Collider other)
11  {
12      if (other.gameObject == Player)
13      {
14          Player.transform.parent = transform;
15      }
16
17  }
```

Figur 22: *Class PlayerAttach*

Dette er en del scriptet, vi har brugt til at transformere brugeren af spillet til et child af pariserhjulvognen (billedet er et skærmbillede af eget projekt).

Den anden metode OnTriggerExit gør det modsatte, når brugeren flytter sig fra pariserhjulvognen. Den sætter brugerens Parent til null, altså fjerner den igen.

HandButton2

For at aktivere vores animationer og lydeffekter har vi tilføjet et script til hver knap, som vi har kaldt HandButton - dette kan ses på figur 23. Dette har vi gjort, da hver knap har nogle små forskelle i form af hvilken animation og lyd, der skal afspilles, men funktionen af scriptet forbliver den samme.

Scriptet er blevet tilføjet til det objekt, som registrerer kollision og derved starter animationen og lydeffekterne. Dette objekt har vi valgt skal være en rød knap, som tjekker for hvornår en kollision starter og stopper. Derudover kalder vi også på spilobjekterne “EngineAudio1” og “Radio1”, hvilket vi gør ved at kalde på deres Tags - Tag har vi tilføjet til objektet i Unity under Inspector. Brugen af Tags til at kalde på lydeffekter bruges også til animationen og bliver tilføjet på samme måde.

```
30
31      protected override void Awake()
32      {
33          base.Awake();
34          onHoverEnter.AddListener(StartPress);
35          onHoverExit.AddListener(EndPress);
36          elevator1Sound = GameObject.FindGameObjectWithTag("EngineAudio1");
37          radio1Sound = GameObject.FindGameObjectWithTag("Radio1");
38
39      }
```

Figur 23: *Objekt aktiveres*

Her tilføjes muligheden for at få objektet til at gøre noget så længe der er kollision, samt hentning af lydfiler (billedet er et skærmbillede af eget projekt).

Når controlleren kommer i kontakt med knappen vil StartPress igennem onHoverEnter registrere, at der er kontakt, hvilket ændrer y-værdien af knappen til at følge med hånden mellem knappens startposition og en minimumsværdi, således at knappen kan bevæge sig. Dette kan ses på figur 24. Derudover bliver vores lydeffekter, som tilhører pariserhjulvognen, aktiveret første gang, der bliver trykket på knappen.

```
hoverInteractor = interactor;
previousHandHeight = GetLocalYPosition(hoverInteractor.transform.position);
previousPress = true;
```

Figur 24: *Position af knappens y-værdi*

Så længe der er kontakt mellem hånden og knappen, følger knappens y-værdi håndens y-værdi uden at starte forfra (billedet er et skærmbillede af eget projekt).

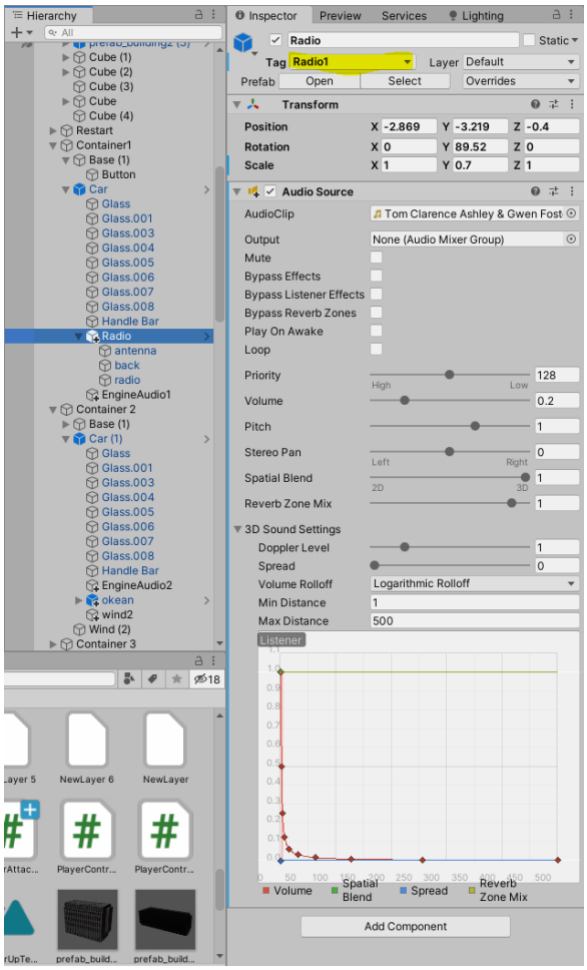
Ved at knappen følger med hånden, der trykker på den, skal positionen derfor opdateres, fordi det er spilleren, der bestemmer, hvor langt knappen skal trykkes ind og i hvor lang tid. Positionen af knappen, som vi aktiverer med StartPress, bliver her konstant ændret ud fra spillerens input.
Ved kontakt med knappen bliver der udover bevægelse af knappen kaldt til henholdsvis Animator Controlleren for at starte den valgte animation, samt kaldt til AudioSource om at få afspillet den valgte lydeffekt (som set på figur 25).

```
radio1Sound.GetComponent<AudioSource>().Play();
elevator1Sound.GetComponent<AudioSource>().Play();
hasBeenPushed = true;
```

Figur 25: *AudioSource*

De lydfiler, der tidligere er blevet hentet, bliver afspillet en gang (billedet er et skærmbillede af eget projekt).

AudioSources er blevet tilføjet direkte til knappen som en komponent, så der kan ændres på, hvordan lyden skal afspilles, hvilket kan ses på figur 26. Indstillinger, der bliver vist, har vi justeret for at skabe en fornemmelse af, at lyden kommer for radioen, der befinder sig inde i pariserhjulvognen. Det betyder, at afstanden mellem spilleren og radioen, samt hvilken vej spilleren kigger, har en betydning for, hvordan lyden bliver afspillet. Dette vil der blandt andet ses nærmere på i afsnit 6.4.



Figur 26: *Klasse*

I Inspectoren for Radioobjectet i pariserhjulvognen kan der ændres på, hvordan en lydfil afspilles, samt hvilken distance spilleren skal have for at kunne høre lydfilen (billedet er et skærmbillede af eget projekt).

HandButton scriptet har dermed to formål, hvor det første er at fungere som en knap, og det andet formål er, at starte handlinger når der bliver interageret med knappen. Uden dette script ville vi derfor kun have muligheden for at bevæge spilleren rundt.

MovementProvider

Dette script har til formål at give Main Camera (det kamera brugerens synsvinkel er baseret på) bevægelse gennem input fra controllerne. Bevægelsen af kameraet foregår i 3D, hvor vores controllere styrer bevægelsen ud af x- og z-aksen og headsettet, som styrer bevægelsen af y-aksen. Derudover har vi implementeret muligheden for Snap rotation, hvilket giver os muligheden for at dreje rundt med hjælp af controlleren.

For at skabe bevægelse gennem controllerne i 2D, var der flere steps, som vi skulle igennem. Til at starte med skulle vi sørge for, at alle controllerer, vi bruger, kunne afgive input til koden og dermed bruge controllernes joysticks, som ses på figur 27.

```
foreach (XRController controller in controllers)
{
    if (controller.enableInputActions)
        CheckForMovement(controller.inputDevice);
}
```

Figur 27: *Lyt til alle input enheder*

Controller input fra alle controllere (billedet er et skærmbillede af eget projekt).

Herefter tog vi og aflæste controllerens joystick i 2D og for at få en Vector for, hvilken vej joysticket var trykket til samt hvor meget joysticket var trykket til at bevæge vores kamera i en given retning og med en given hastighed - se figur 28.

```
if (device.TryGetFeatureValue(CommonUsages.primary2DAxis, out Vector2 position))
    StartMove(position);
```

Figur 28: *Bevægelse med joystick*

Controllerens joystick bliver aflæst, så spilleren kan rykkes herefter (billedet er et skærmbillede af eget projekt).

Selvom vi nu har fået bevægelse tilføjet til vores kamera, mangler vi stadig at rykke controllerne afhængigt af hvilken vej headsettet peger. Dette løser vi ved at bruge funktionen eulerAngles.y, som drejer kameraobjektet med headsettet.

Denne Euler funktion er også den, vi bruger til at benytte os af Snap Turn Provider som komponent, som har det formål at dreje den vej kameraet peger, uden at spilleren drejer hovedet. Dette kan ses på figur 29. Dermed er der ved hjælp af bevægelse og rotation gennem controllere

mulighed for, at spilleren ikke skal bevæge sig overhovedet, og derfor kan spille spillet på et meget mindre areal, end der er nødvendigt, hvis spilleren fysisk skal bevæge sig rundt.

```
Vector3 direction = new Vector3(position.x, 0, position.y);
Vector3 headRotation = new Vector3(0, head.transform.eulerAngles.y, 0);
direction = Quaternion.Euler(headRotation) * direction;
```

Figur 29: *Bevægelse i forhold til headsettet*

Vejen, headsettet peger, bliver taget i betragtning, når bevægelse og Snap Rotation bliver kaldt (billedet er et skærmbillede af eget projekt).

Udover spillerens input, som bruges til at bevæge sig med, bruger vi også dette script til at tilføje tyngdekraft til spilleren, således spilleren falder ned, hvis spilleren bevæger sig ud over en kant. Dette har vi gjort ved at bruge Vector3, hvor y-værdien har "Physics" egenskaben, som set på figur 30.

```
Vector3 gravity = new Vector3(0, Physics.gravity.y * gravityMultiplier, 0);
gravity.y *= Time.deltaTime;
```

Figur 30: *Tyngdekræft*

Tilføjer tyngdekraft som ved hjælp af tid sørger for at opdateringshastigheden ikke har indflydelse (billedet er et skærmbillede af eget projekt).

Denne "Physics" egenskab ville vi også kunne tilføje til x- og z-aksen, hvis vi ønsker, at spilleren skal falde sidelæns. Herfor kan Physics.gravity også tilføjes til andre akser, men er dog tilføjet til y aksen i vores tilfælde, eftersom der i vores tilfælde kun er brug for "tyngdekraft" på y-aksen.

Alle disse scripts lægger til grund for spilverdenens dimensioner, hvor terrænets størrelse og sigtbarhed sørger for visuel stimulans i forhold til følelsen af indlevelsen for spilleren i den aktuelle simulering.

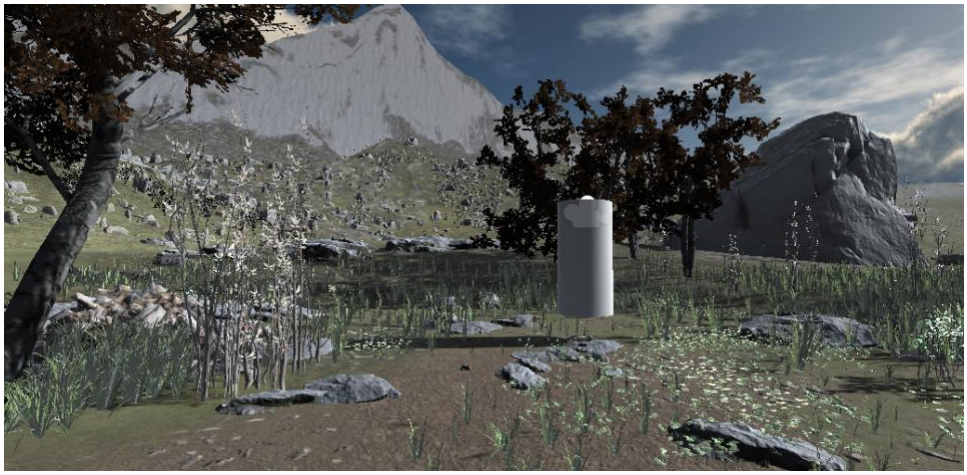
6.4 Spilverdenen

Efter at have gennemgået Unitys overordnede funktionaliteter, såsom Scripts, GameObjects og Prefabs, vil der herfra ses nærmere på, hvordan disse funktionaliteter til sammen har givet os mulighed for at kreere en spilverden ud fra diverse krav og forslag, vores informanter havde om, hvordan produktet skulle se ud. Af denne årsag vil vi med det næste komme ind på, hvordan vores

spilverden er blevet skabt, hvordan den fungerer og til slut runde af med spilverdenens designkarakteristika i en kontekst af, hvad vores empiri medbragte til projektet.

Terræn

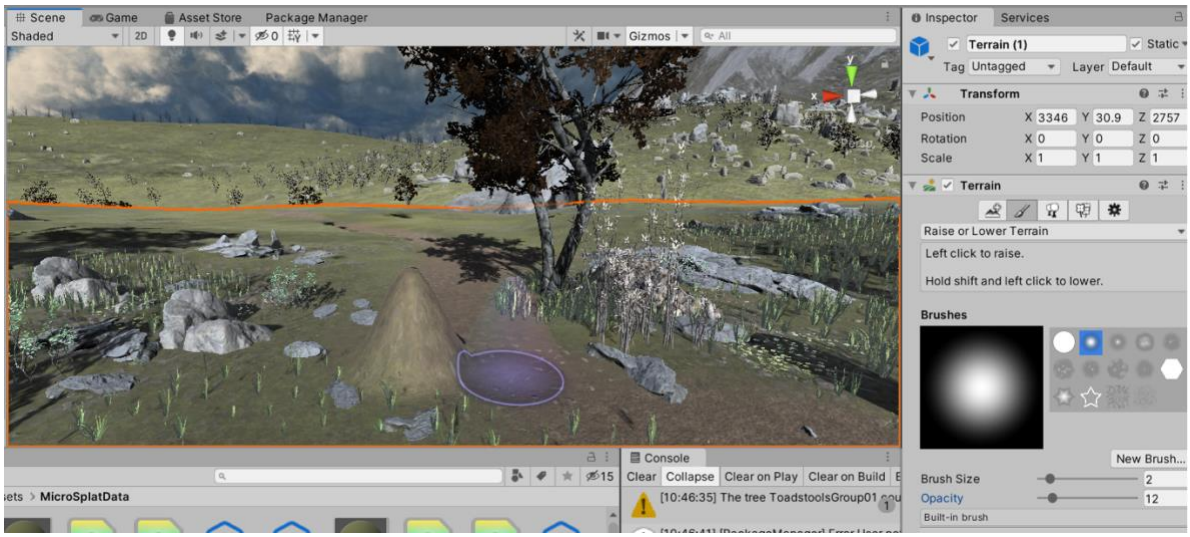
Spilverdenen, som brugeren kommer til at bevæge sig rundt i, er lavet gennem funktioner, der eksisterer lokalt i spilmotoren Unity og udefrakommende Assets, der er indhentet fra Unitys Asset Store. På figur 31 vises der et Terrain Object, der indeholder forskellige GameObjects, såsom træer, sten, kviste, en Skybox og et Directional Light Source.



Figur 31: *Terræn*

Et terrain skaber spilarealet med visuelle effekter og GameObjects (billedet er et skærmbillede af eget projekt).

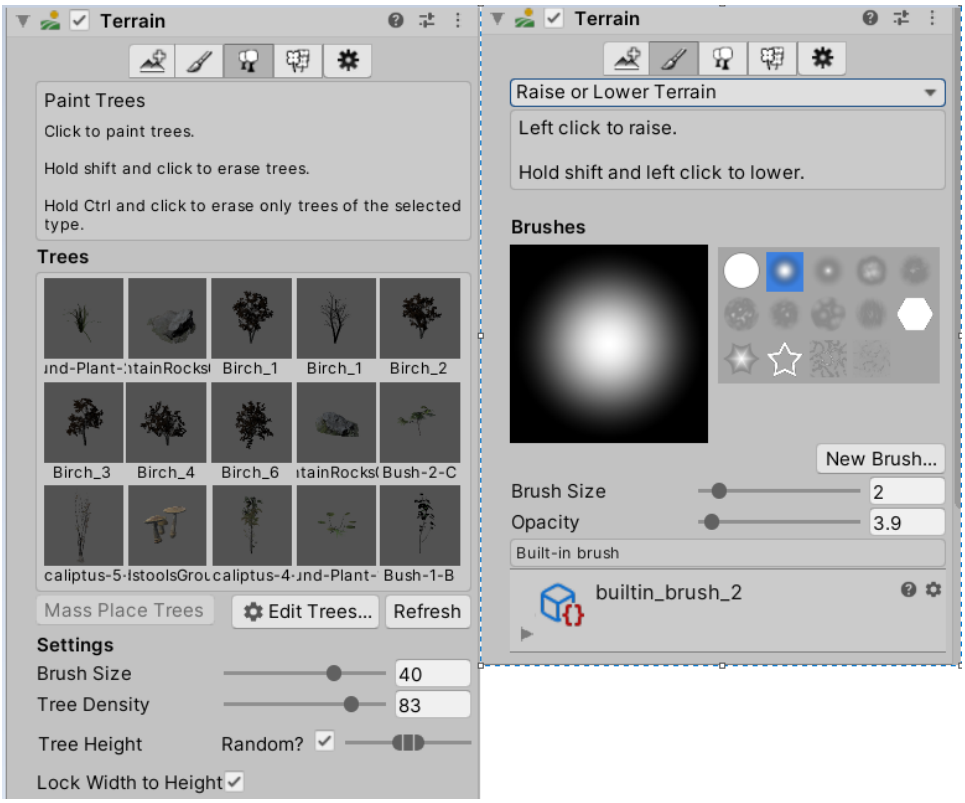
Terrænobjektet fungerer dog lidt anderledes end de andre objekter, der kan ses på figur 31. Observeres bjerget i baggrunden på billedet er det et eksempel på, at terrænets dimensioner kan modificeres til passende størrelser. Disse dimensioner kan modificeres gennem Terrain objektets indbyggede komponent kaldet; Terrain Tools. Denne komponent kan blandt andet forhøje eller trække terrænet ned i større eller mindre områder. Dette sker gennem Terrain Tools ene funktion, "Raise or Lower Terrain", hvorunder der kan vælges, hvor stor "penslen" der bruges til at modificere terrænet skal være og kræften på, hvor meget terrænet skal modificeres inden for penslens område. Dette kan ses på figur 32.



Figur 32: *Raise or Lower Terrain*

Et eksempel på raise og lower terrain, som bruges til at modificere terrænet (billedet er et skærmbillede af eget projekt).

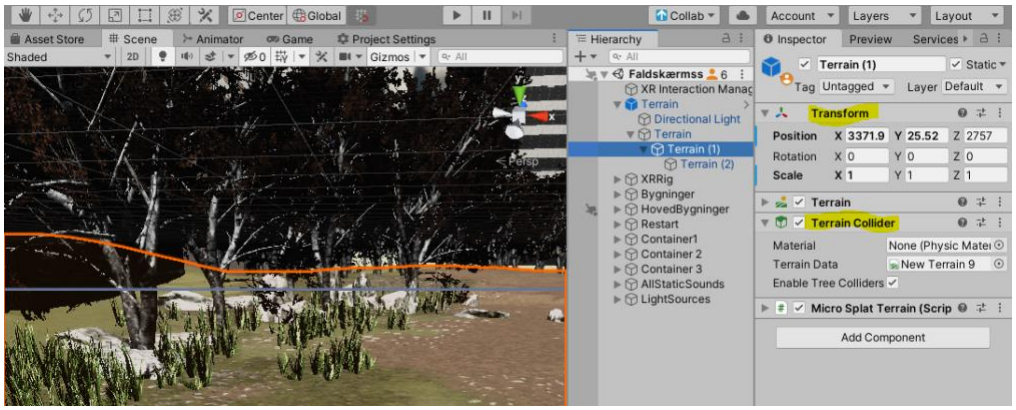
Herudover kan objekter også tilføjes forskellige områder af terrænet gennem Paint Trees funktionen i Terrain Tools. Under denne funktion kan forskellige Prefabs af objekter tilføjes til terrænet ved at angive med musen, hvor et træ eksempelvis kan placeres på terrænet. Disse tilføjelser kan således modificeres ud fra, hvor mange objekter der skal placeres inde for det område, angivet ved Brush Size, og hvor mange af de valgte objekter, der skal placeres indenfor det område gennem Tree Density slideren (se figur 33). Hertil kan disse objekters y- og x-værdier, der tilføjes, også modificeres per tilfældigvis igennem Tree Height og Tree Width, hvortil et interval kan angives ved at bruge slideren ud fra funktionen. Herved vil der med brugen af forskellige Prefabs, der skal forestille træer, hurtigt kunne anlægges objekter såsom træer, der har forskellige højder og spredning.



Figur 33: *Raise og lower funktion & Paint Trees funktion i Inspector*

Vi benytter to forskellige funktioner fra Terrain tools der kan modificere terrænet (billedet er et skærmbillede af eget projekt).

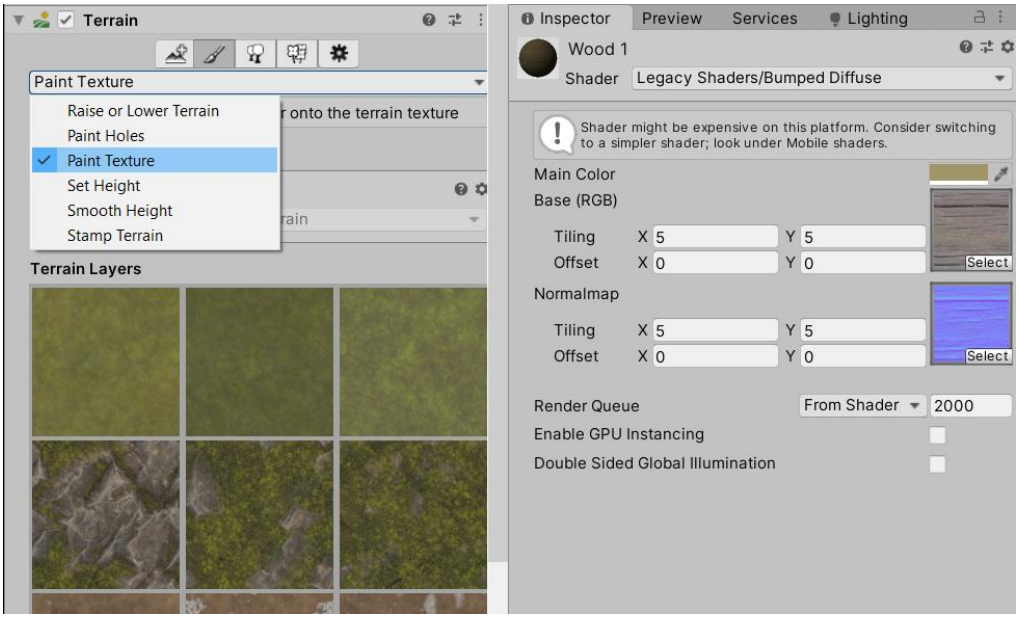
Terrain Objektet kommer desuden med en komponent kaldet; Terrain Collider, som gør, at andre objekter, der har komponenten kaldet: Rigidbody, kan kollidere med terrænet. Terrain Collider kan ses på figur 34. Rigidbody er tiltænkt som en form for tyngdekraft for det objekt, det er angivet på, og vil derfor trække objektet nedad, når spillet startes. Hvis objektet kolliderer med Terrain objektets Collider, vil objektet derfor stoppe og derved ikke flyve gennem terrænet. Hertil kunne vi derfor bruge Terrain objektet som et spilleområde, hvorpå spilleren ville kunne bevæge sig rundt.



Figur 34: *Terrain Collider*

Terrain Collideren for et Terrain Object bruges til at skabe en fysisk væg, der kan kollideres med (billedet er et skærmbillede af eget projekt).

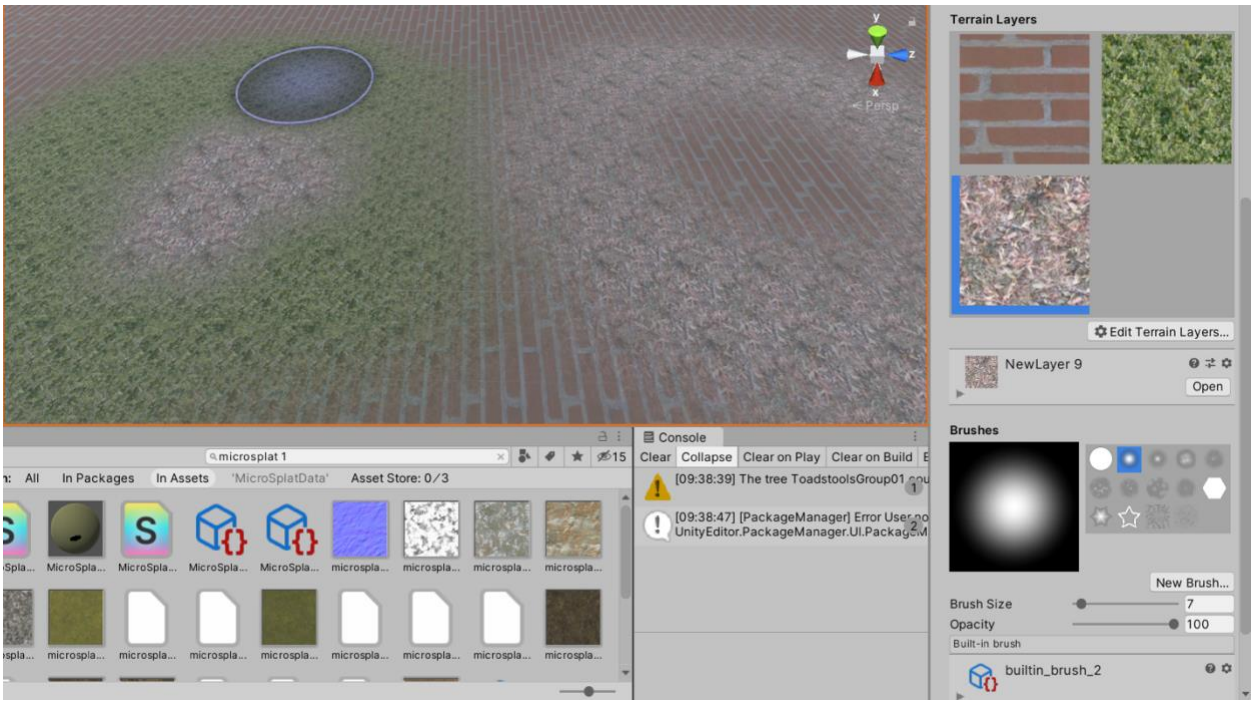
Lokalt i Unity under Terrain Tools kan Materials, eller Textures også tilføjes til terrænet ved at bruge Paint Texture funktionen, hvorunder Terrain Layers kan tilføjes. Textures er et fladt billede af for eksempel græs, der i en firkant kan tilføjes terrænobjektet, som anvist på figur 35. Herunder kan Materials udgøre flere forskellige typer af Textures, hvor et Normal Map, der skaber variation og et Height Map, der skaber dybde i Texturen, også tilføjes til terrænet udover den oprindelige Texture, der kaldes Albedo.



Figur 35: *Terrain Texture Layer & Material Settings*

Eksempel på tilføjelse af Textures til et terræn og et Materiale (billedet er et skærmbillede af eget projekt).

Normalvist ville de Textures eller Materials, der skal bruges på terrænet, blive tilføjet direkte her under Terrain Tools. Projektgruppen fandt dog denne løsning problematisk for besvarelsen af projektets problemformulering, eftersom når teksturen bliver tilføjet over et større område, opstår der et fænomen, som kaldes Texture Tiling. Hvis figur 36 observeres, kan det ses, at de teksturer der er blevet anlagt på terrænet gentages efter at være blevet anlagt side om side, da de anføres på terrænet i firkanter, der er tilsvarende teksturens billedformat. Dog betyder dette også, at der opstår et mønster i terrænets udseende, da disse firkanter af teksturer anføres på terrænet side om side. Dette mønster skred imod projektets krav om, at spilverdenen skulle være virkelighedsnær, eftersom denne form for mønstring ikke optræder ude i naturen, og derfor ville vi finde en løsning til at fjerne denne Texture Tiling. Hertil er der flere forskellige løsninger på hvordan denne Texture Tiling kan nedbrydes.

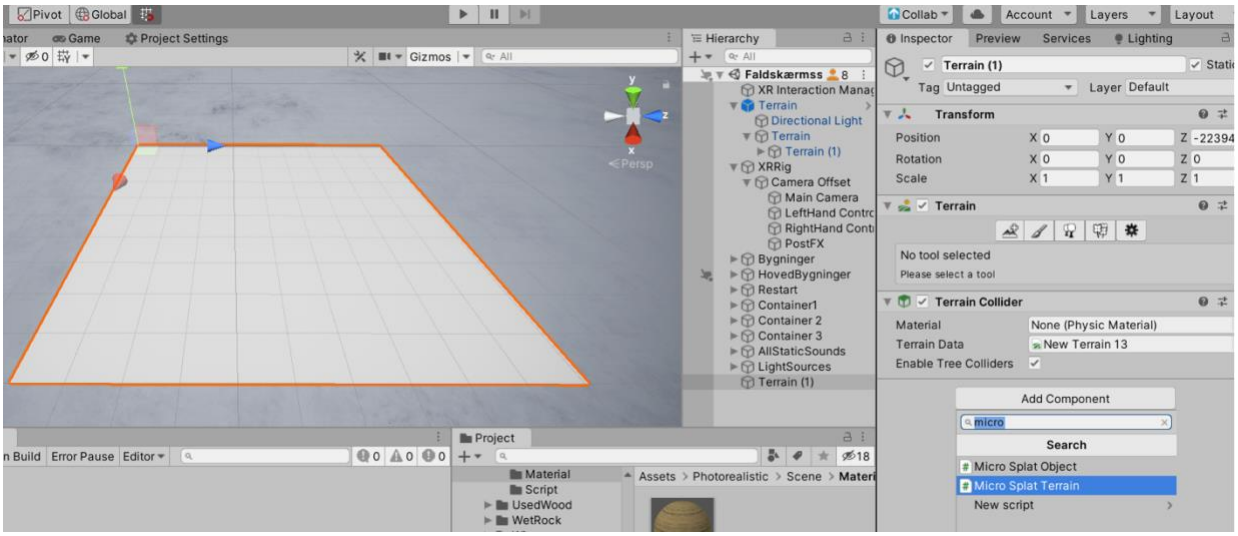


Figur 36: *Texture Tiling*

Eksempel på hvordan Tekstures bliver anlagt på et Terrain Object gennem Terrain tools funktionen Paint Textures (billedet er et skærmbillede af eget projekt).

I en søgning på internettet om at løse dette problem, var der i de fleste tilfælde tale om at lave et terrænobjekt i et eksternt 3D-program kaldet: Blender. Programmet Blender bruges til at lave 3D-

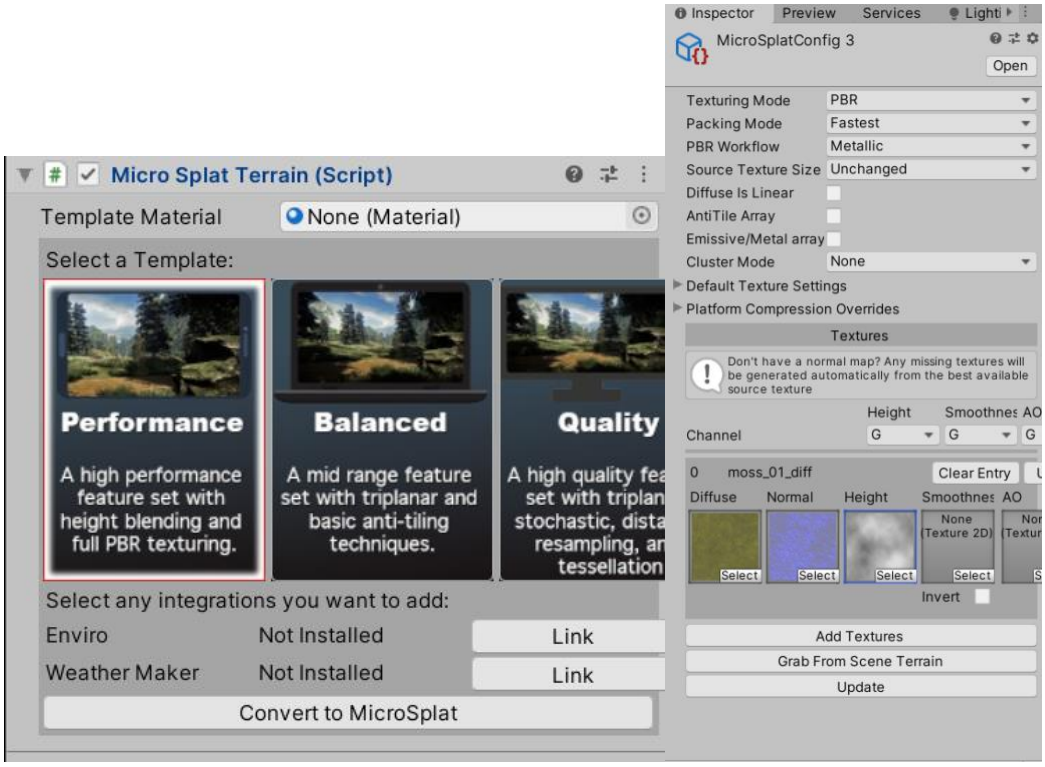
objekter, hvori de samme teksturer, der bruges i Unity, kan bruges. Herunder kan disse teksturer anføres per tilfældigvis, hvilket betyder, at teksturens orientering kan ændres til forskellige orienteringer gennem nogle plugins, der kan bruges i Blender-programmet. Herefter kan objektet, der er blevet udviklet i programmet Blender, overføres som et Prefab til Unity og derved bruges i Unity som for eksempel et terræn. Denne løsning er dog mere tidskrævende, end hvad projektgruppen søgte. Programmet, der blev udviklet gennem dette projekt, er en form for simulering, hvori personen, der bruger programmet, skal testes i højde. Af denne årsag skulle projektet også indeholde et meget stort spilareal, da programmet anskues gennem Virtual Reality og havde et krav om at være så virkelighedsnær som mulig. Projektgruppen fandt derfor en anderledes løsning, som bestod af et Asset kaldet Microsplat, der nedhentes fra Unitys Asset Store. Microsplat er en form for udvidelse til Terrain Tools, hvori forskellige moduler kan anføres og skabe forskellige opdateringer af et Terrain Object. Gennem Microsplat skal alle Textures dog indsættes i projektet på anderledes vis end ved brug af Terrain Tools. Når Microsplat og de forskellige ønskede moduler er nedhentet fra Unitys Asset Store, skal det Terrain Object, der skal bruge Microsplat, tilføjes en ny komponent, der hedder Micro Splat – Terrain, som set på figur 37.



Figur 37: *Microsplat komponent*

Eksempel på hvordan Microsplat bliver tilføjet som komponent til et terræn objekt (billedet er et skærmbillede af eget projekt).

Når dette er gjort, skal terrænobjektet konverteres til Microsplat ved at trykke på Convert to Microsplat. Dette gør, at et Script fra Microsplat komponenten bliver kørt, når terrænet er blevet konverteret til et Microsplat - se figur 38. Derved startes en proces, der omdanner det Terrain Object, der får Microsplat tilføjet, som gør, at objektet kan modificeres gennem de scripts, der hører til assetet Microsplat. Når terrænobjektet er blevet konverteret, skal teksturer tilføjes gennem et config, herunder kan teksturer der har tilhørende, Normal, Height, Smoothness, og Ambient Oclussion (AO) maps tilføjes, og når de teksturer, der ønskes, er blevet tilføjet, skal terrænobjektet opdateres ved at trykke på knappen "Update" - se figur 38.

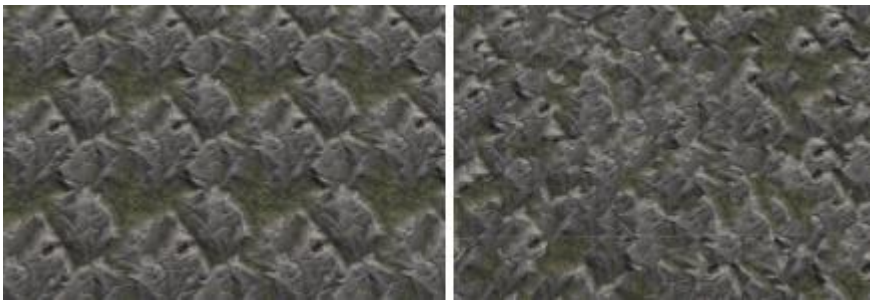


Figur 38: *Microsplat Script & Texture tilføjelse*

Eksempel på hvordan Microsplat initialiseres og hvordan teksturer tilføjes efterfølgende (billederne er skærmbilleder af eget projekt).

Når terrænobjektet er blevet konverteret til Microsplat, kan alle teksturer, der er blevet tilføjet, bruges gennem Terrain Tools, og modificeres gennem et Microsplat Material, der har forskellige indbygget moduler, som kan modificere de enkelte teksturer, der er blevet tilføjet til terrænet.

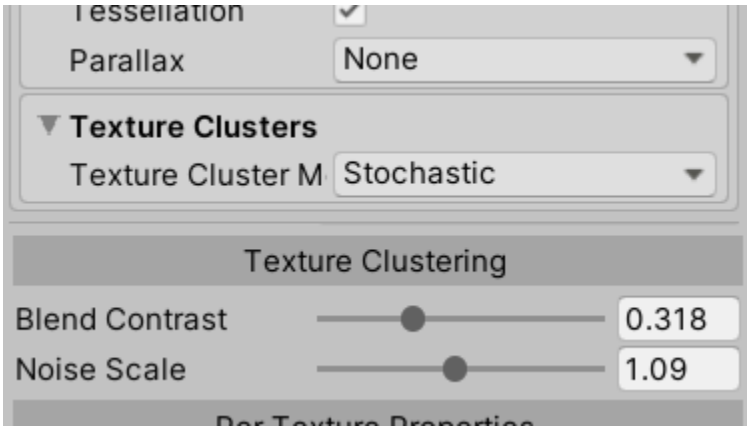
Heriblandt har vi benyttet os af fire forskellige moduler. Det meste væsentligt modul til Microsplat i forhold til projektet, er Texture Clustering modulet - se figur 39.



Figur 39: *Texture Clusters*

Eksempel på Stokastisk tekstur variation (billederne er skærmbilleder af eget projekt).

Som det kan ses på figur 39, skaber Texture Cluster modulet den tilfældige re-orientering af de forskellige teksturer, således terrænets udseende får den tilfældighed, der var ønsket. Denne tilfældighed fremkaldes ved at indstille Texture Clusters til en Stokastisk funktion (figur 40), som dermed giver den effekt af en tilfældig anførsel af de teksturer, der er anbragt på terrænet.

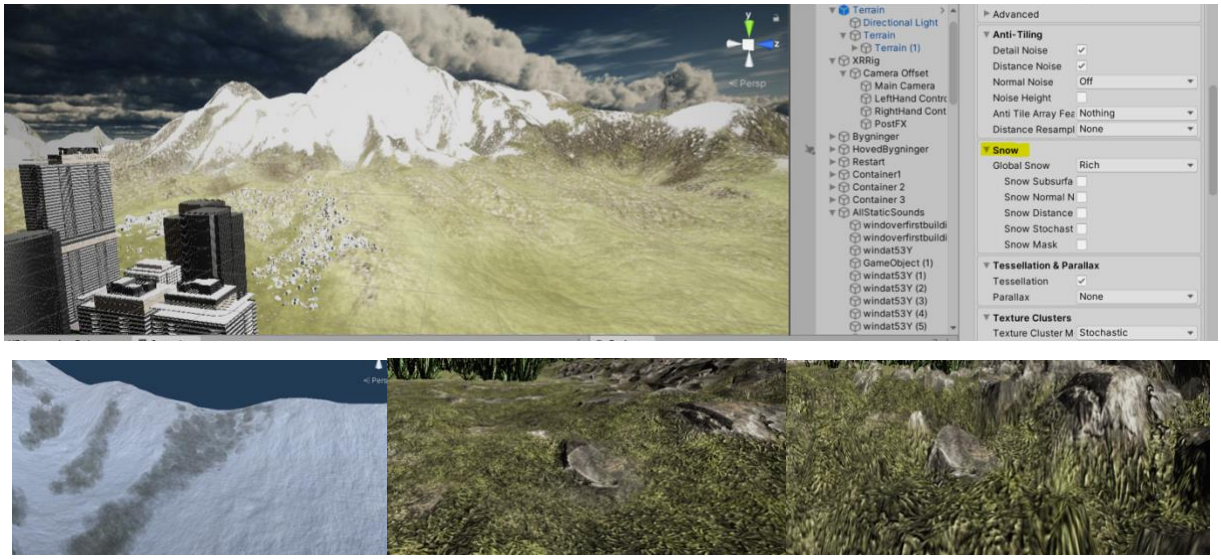


Figur 40: *Texture Clusters i Inspectoren*

Stokastisk algoritme der er taget i brug med henhold til at skabe variation i teksturernes orientering (billedet er et skærmbillede af eget projekt).

Herudover er der i projektet også taget brug af Anti-Tiling modulet, der indlægger et Noise Map hen over hele terrænet, som giver teksturerne mere variation i terrænets udseende. Hernæst er Snow modulet også taget i brug, som indlægger nye teksturer baseret på en angivet højde på terrænet. Herved kan høje steder såsom bjerget på figur 41 få tildelt sne, de steder hvor bjerget er

højest. Snow modulet tager også højde for hver eneste teksturs Height Map, hvis teksturen er inde for den angivede højde, hvor sneen skal falde. Det vil sige, hvis det Height Map der befinder sig på en tekstur, der får tildelt sne gennem Snow modulet, så vil sneen ikke lægge sig de steder hvor Height Mappet for teksturen er højest. Til slut er der også taget brug af Tessellation modulet, der forhøjer hver enkelte tekstur ud fra dets Height Map, som kan ses på figur 41.



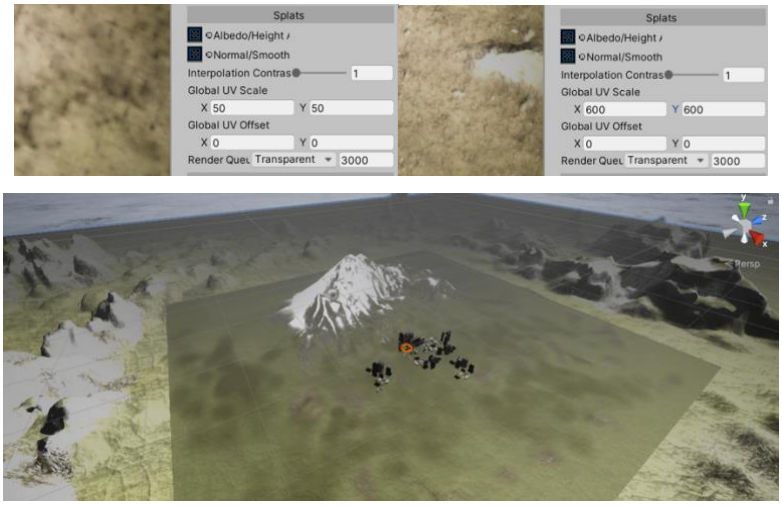
Figur 41: Microsplat moduler

Eksempler på forskellige Microsplat moduler der skaber variation i terrænet (billederne er skærmbilleder af eget projekt).

Til slut med henhold til terrænet, blev der tilføjet 3 forskellige Terrain Objects, der hver især havde fået tilføjet en Microsplat komponent, hvorpå det ene Terrain Object var betydeligt større end det næste - se figur 42. Spillet skulle bruge et stort terræn, eftersom spillet skulle skabe en fornemmelse af højde, og når det var tilfældet at brugeren skulle højt op, ville brugeren derfor også kunne se længere. Hertil fandt vi ud af, at der skulle bruges et stort terræn, eftersom brugeren ellers ville kunne se grænsen på terrænet fra det højeste punkt, hvor brugeren kunne befinde sig. Dog ved implementering af enkelt stort terræn forsvandt en stor del af detaljerne ved det enkelte store terræn objekts teksturer, hvorfor vi valgte at foretage en anden løsning.

Som beskrevet ovenfor havde vi lavet 3 forskellige terrænobjekter, der hver havde sin egen størrelse.

Det mindste terrænobjekt er, hvor spilleren kommer til at bevæge sig rundt. Dette objekt er indstillet til at være 100 Width x 100 Length i forhold til Unitys positions værdier. Det næste er sat 6000 x 6000, og det sidste og største er sat til 24000 x 24000. Hernæst er hver totale tekstur værdi per terræn blevet indstillet, således der optræder færre teksturer på de større Terrain Objects. Dette betyder, at hver tekstur, der bliver anbragt på terrænobjektet, kan fylde et mindre eller et større område ved at bruge Microsplat komponentens indbygget funktion kaldet Global UV Scale – se figur 42. Sættes disse værdier til et højere punkt, vil området af, hvad hver tekstur fylder, blive nedskaleret, hvilket gør, at teksturen vil blive anvist i et mindre format med flere gentagelser over det område, hvor teksturen er blevet anført. Sættes værdierne til et lavere punkt, vil teksturens format blive opskaleret og derved fremtræde færre gange hen over hele terrænobjektet.



Figur 42: *Global UV scale & Terræn Størrelse*

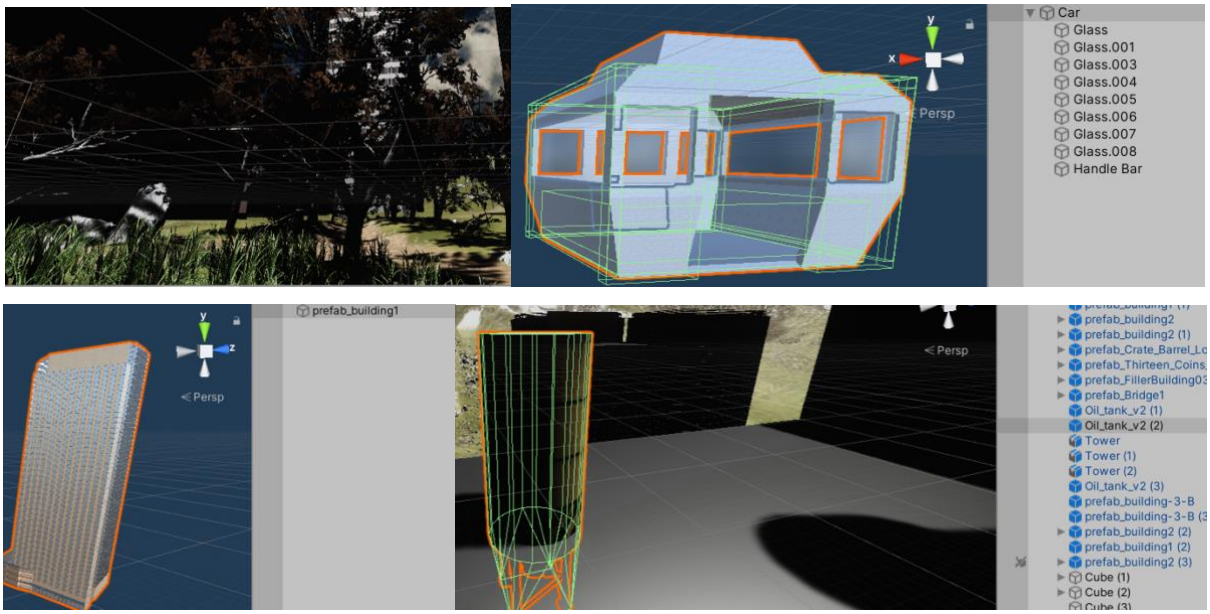
Global UV Scale af tekstur, hvorunder Teksturens størrelse på terrænobjektet forstørres (billedet er et skærmbillede af eget projekt).

Herfor fandt vi en løsning, hvorpå Terrænet i de forskellige lag kunne bevare dets detalje, samtidigt med at der skulle bruges mindre kræft fra enheden, der skulle køre programmet til at indtegne de forskellige teksturer. Herefter skabte vi forskellige bjergkæder på det største terræn, så grænsen på terrænet ikke kunne ses fra det højeste punkt, hvor spilleren kunne befinde sig. Hernæst var Microsplat også en løsning på at give teksturen en vis form for variation, der derfor gjorde terrænet mindre uniformt og mere naturligt at se på, når spillet afprøves.

GameObjects

I spilverden har vi brugt en del forskellige objekter, der tidligere i denne rapport også er blevet redegjort for, dog er der forskel på hvorledes objekterne er blevet brugt, hvorfor der vil foretages en gennemgang af de objekter der endnu ikke er blevet redegjort for.

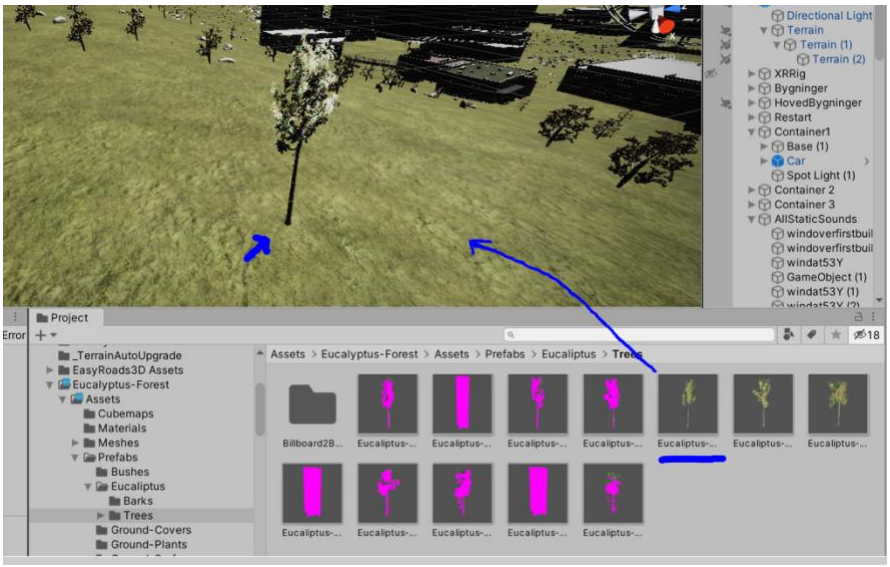
Spilverdenen, der udvikles gennem Unity, kan indeholde mange forskellige typer af objekter såsom terrænet, dog er det ikke det eneste spilverdenen kan indeholde. Eksempelvis har vi taget brug af objekter, der har tilegnet scripts, således objektet aktiveres, når spilleren interagerer med det. Ligeledes har vi en del andre objekter såsom bygninger, træer, sten og græs, der er med til at tilføje visuelle elementer til spilverdenen, og derved give brugeren noget at se på. Mere specifikt har vi nedhentet forskellige Prefabs af objekter, der skal forestille bygninger og træer, som er blevet brugt forskellige steder i spilverdenen - se figur 43.



Figur 43: *Prefabs*

Eksempler på Prefabs der er blevet brugt i projektet. (billedet er et skærmbillede af eget projekt).

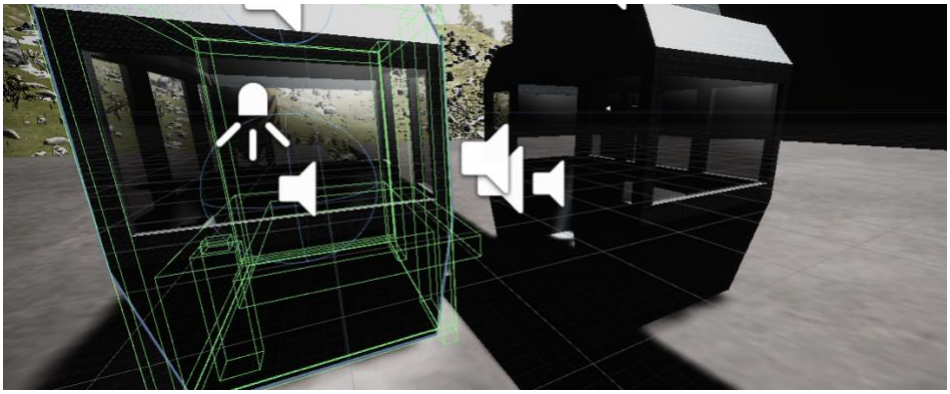
Nogle objekter er blevet tilføjet gennem Terrain Tools - Paint Trees funktionen, hvor andre objekter, såsom bygningerne og pariserhjulet, er blevet tilføjet ved at trække dem ind fra de Assets mapper, der fremgår af Project vinduet i Unity - se figur 44.



Figur 44: *Tilføjelse af prefab*

Prefab der bliver tilføjet scenen ved at trække det ind fra projektmappen (billedet er et skærmbillede af eget projekt).

Placeringen af objekterne har i de fleste tilfælde været placeret for at skabe et baggrundsudseende, der ikke bare var terrænet bestående af bjerget, bakker osv. Dog med henhold til spilleområdet skulle de objekter, som spillerne kunne interagere med, have en specifik placering såsom pariserhjulsvognen, der kan ses på figur 45. Dette var fordi, når spilleren er inde i spilverdenen, skal vedkommende bevæge sig fra en vogn til den anden. Denne bevægelse skal ske ved fysisk bevægelse i stedet for at bruge Virtual Reality headsettets medfølgende controllere. Af denne årsag har vi måtte placere de forskellige vogne indenfor en grænse af hinanden (se figur 45), eftersom det spilleområde, spilleren i virkeligheden kommer til at have, er begrænset til 5 gange 5 meter.

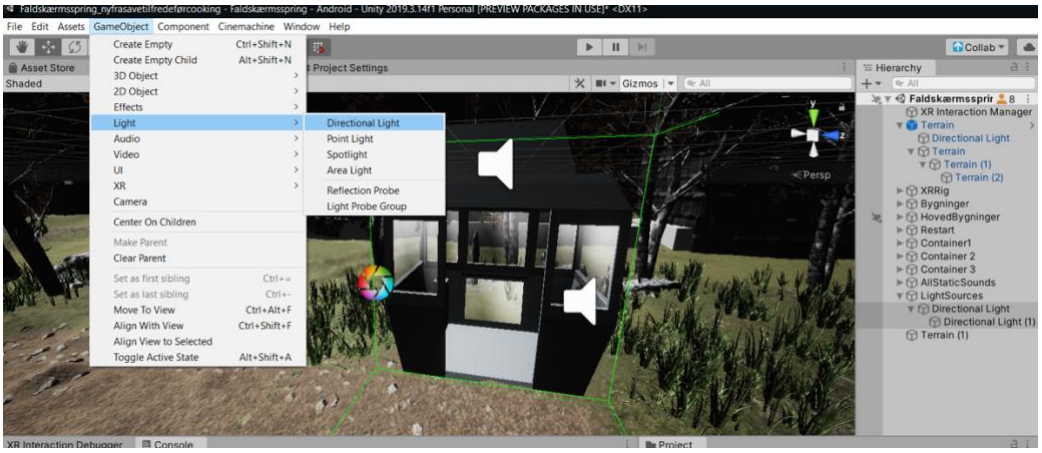


Figur 45: *Placering af Prefabs*

Specifikke placeringer af de objekter spilleren kan interagere med (billedet er et skærmbillede af eget projekt).

Light source

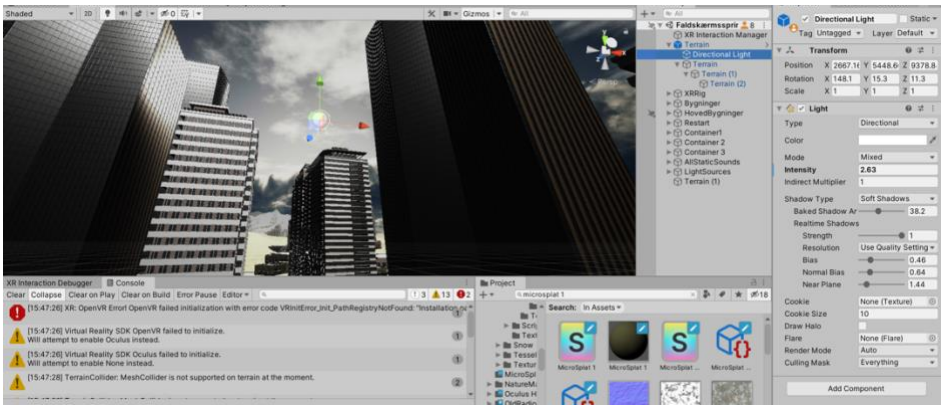
For at få en virkelighedsnær fornemmelse i spilverdenen, er der i projektet blevet forsøgt at efterligne naturligt lys. Lyset i spillet er en af de dele, der ikke er blevet lagt særligt meget vægt på, eftersom naturligt lys er nemt at efterligne i Unity. I spilmotoren eksisterer et GameObject kaldet Directional Light. Directional Light findes under Light kategorien under GameObjects - se figur 46 - og fungerer som en enkelt lyskilde, der lyser hele scenen op fra en bestemt vinkel. Dette betyder, at alle objekter, der er placeret i spilverdenen, ville blive belyst fra denne ene lyskilde. Hertil kan rotationen på Directional Light ændres, således lyset falder på scenen fra en anderledes vinkel.



Figur 46: *Directional Light*

Tilføjelse af et Directional Light objekt til scenen (billedet er et skærmbillede af eget projekt).

Derved indstillede vi objektet for vores Directional Light til at have den samme position som solen, der var inkluderet i den Skybox, der bruges i projektet. Herefter satte vi scenekameraet til at have samme position som det kamera, der ses gennem, når spillet køres. Således kunne rotationen på vores Directional Light indstilles til at have samme vinkel som solen fra den position, hvor spilleren ville befinde sig. Dette kan forklares ved, at lyskildens position og rotation var afgørende for, hvor lyset skulle falde, og skulle derved indstilles til at passe overens med den sol, der var inkluderet i Skyboxen for at give den naturlige belysning, vi eftersøgte (se figur 47).

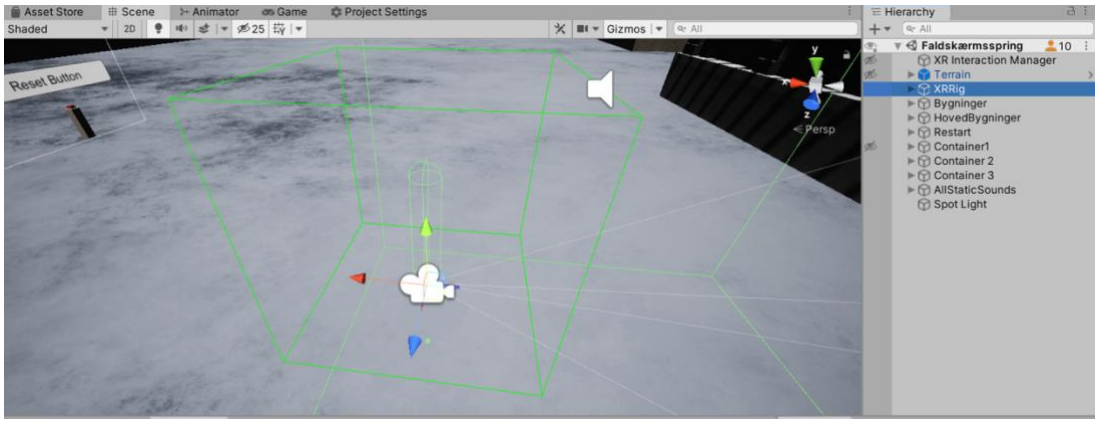


Figur 47: *Directional Light position*

Indstillingen af hvilken vinkel vores lyskilde (Directional Light) skulle have i forhold til den sol der var inkluderet i Skyboxen (billedet er et skærmbillede af eget projekt).

XR Rig

Efter vi har opsat vores terræn, samt importeret de assets vi har brugt, ønskede vi at tilføje et kamera, som skulle bruges i samarbejde med forskellige VR-briller, således spilleren kunne opleve verdenen igennem VR. Dette har vi gjort ved hjælp af XR-Plugin-Management som indeholder et VR-kamera (se figur 48). Dette VR-kamera, som er importeret, har et XRRig som Parent, et kamera offset som Child, som yderligere har kameraet, højre controller og venstre controller som Child af kameraoffsettet. Ved at bruge dette XRRig, som kommer igennem Unity, har vi mulighed for at bruge flere forskellige headset til at spille spillet, hvilket vil sige, at vi ikke begrænser brugeren til ét specifikt VR-headset. Dette XRRig viser dog ikke hænderne, selvom de eksisterer i hierarchy, hvilket betyder, at vi selv skulle lave et Prefab, som kan repræsentere vores hænder i form af enten controllere, eller noget der ligner rigtige hænder. Her valgte vi at tilføje begge dele, hvor der igennem HandPresent scriptet, som tidligere beskrevet, kan vælges mellem at vise controllere eller hænder.



Figur 48: *XRRig*

XR Rig er en samling af objekter, der udgør spilleren. Dette indebærer hovedkamera og hænderne (billedet er et skærmbillede af eget projekt).

Efter vi har fået hænderne repræsenteret i programmet, kunne vi godt tænke os muligheden for at bevæge os rundt i spilverdenen, hvilket betyder at rykke kameraet rundt. Som en del af XRRig'et havde vi allerede mulighed for at bevæge os rundt ved fysisk at gå rundt eller dreje rundt. Dog er der en del begrænsninger til fysisk at bevæge sig rundt i spilverdenen, hvilket betød, at vi også valgte at implementere muligheden for at bevæge XRRig'et rundt og dermed kameraet rundt. Dette gjorde vi ved at tilføje en funktion til hver af controllernes hænder, således der på den venstre controllers joystick kan styres, hvilken vej XRRig'et skal bevæge sig, og den højre kontrollers joystick kan dreje rundt om sig selv for at ændre kameravinklen. Dette implementerer vi igennem henholdsvis scriptet movementProvider og komponenten Snap Turn Provider, som begge hører under XRRig'et.

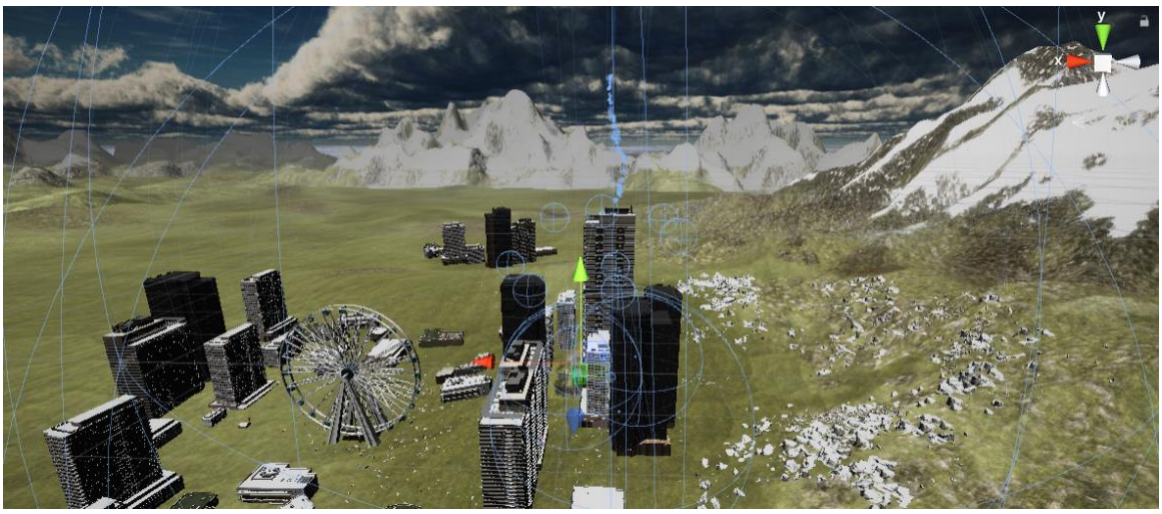
Som en afsluttende feature ønskede vi, at spilleren skulle kunne interagere med objekter, hvilket vi løste ved at tilføje komponenten XR Direct Interactor til hver af vores hænder.

Lyddesign i spilverdenen

Lyd er et vigtigt sanseelement for skabe en god VR-oplevelse. Specielt i vores VR-projekt, hvor vi gerne vil sælge højdeelementet, der sørger lyden for ekstra realisme af simulationen. Lyddesignet i spilverdenen er designet ved at oprette en række tomme objekter. Disse objekter kan indeholde lyde som for eksempel vind, motorlyd og musik. Lydene er nedhentet fra forskellige sider på nettet og klippet sammen med et gratis lydredigeringsprogram kaldet Audacity. Her har

vi redigeret lydene så de opfylder vores behov. Vi benytter os af statiske lydobjekter samt lydobjekter, der er Child af vognene. De statiske lydobjekter repræsenterer hovedsageligt vind, som er placeret rundt omkring vognanimationernes baner. I stedet for at der er en konstant vind, der følger med i brugerens ører, falder vinden ind og ud af brugerens høre rækkevidde. Det har vi gjort for at gøre vinden mere realistisk. Udover dette har vi placeret flere vindobjekter tæt på hinanden i de højere lag og givet disse forskellig pitch for at øge intensiteten af vinden. Formålet med dette er at give brugeren en følelse af at være omsluttet af vind fra alle sider og at give en fornemmelse af, at vedkommende befandt sig højt oppe.

Lydobjekterne, som er Child af vognene i scenen, følger med vognene i deres animationer. Disse lydobjekter inkluderer musikken fra radioen og motorlyde til vognene. Lydene bliver aktiveret, når spilleren trykker på knappen i vognen, og er redigeret til at matche tiden af vognanimationen. I modsætning til lydobjekterne, der er Child af vognene, kører de statiske objektlyde konstant på loop og er aktiveret gennem Awake metoden.



Figur 49: *Lydobjekter*

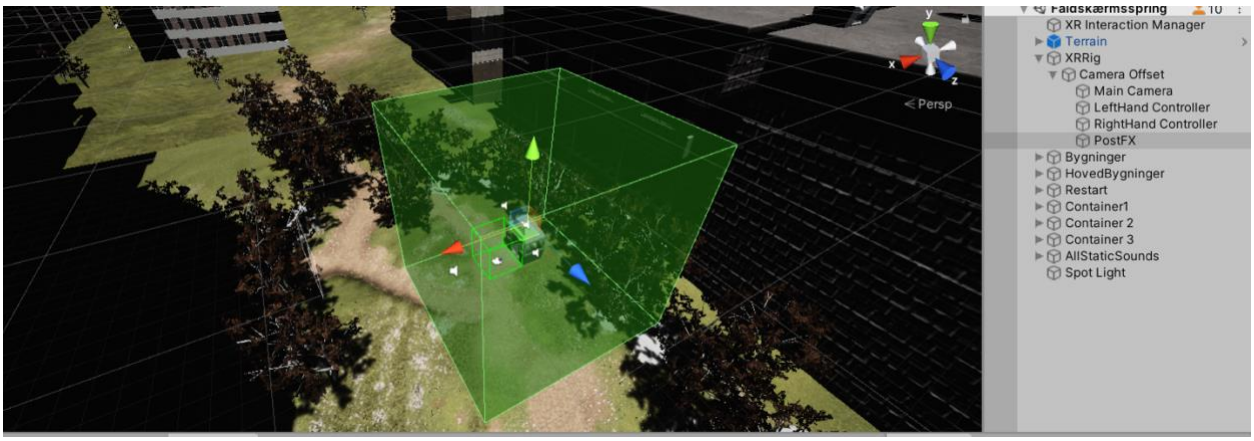
Alle statiske lydobjekter, der afgiver lyd inden for de blå zoner, indikeret på billedet (billedet er et skærmbillede af eget projekt).

Vi håber med lyd-designet af spilverdenen at skabe et ekstra lag af realisme for simulationen, som kan være med til at øge produktets kvalitet i sammenhæng med højdeelementet.

PostProcessing

Post Processing er en effekt der kan tilføjes på kameraet som et specifikt Layer, ligesom et kamerafilter, herunder kan forskellige ting justeres såsom farve, sløring, Depth of Field og andre ting. Denne effekt skal udover at blive tilføjet som et Layer på kameraet, skal det også tilføjes til et tomt objekt. Dette objekt skal indeholde komponenten Post Processing, hvorunder de forskellige effekter kan tilføjes og redigeres i styrke. Objektets kollisionszone eller den Collider, objektet indeholder ved skabelse, kan fungere som et område, hvor denne effekt fra Post Processingen skal tiltage for kameraet.

Dette betyder, at der for eksempel kan skabes et bestemt område, hvor kameraet skal have et bestemt filter, og hvis spilleren bevæger sig ind i den zone, vil effekten derved tiltræde. I vores tilfælde har vi valgt at have en zone omkring XR-Rigget, hvor denne effekt skal tiltræde. Denne effekt er sat som en del af XR-Riggets Prefab, hvilket gør, at når XR-Rigget bevæger sig, så følger Post Processing objekteffekten med Rigget - se figur 50.



Figur 50: *PostProcessing*

Eksempel på PostProcessing zone, der er sat som et Child til XRRigget (billedet er et skærmbillede af eget projekt).

Den specifikke årsag til brugen af Post Processing var på grund af, der eksisterer en Anti-Aliasing funktion inden i Post Processing komponentet. Anti-Aliasing fungerer som en form for falsk opløsning. De steder, hvor der er kanter på objekter, som er skrå, bliver vist som små kanter. Anti-Aliasing kan fjerne de fleste af disse firkanter ved i teorien at indsætte flere mindre firkanter ind de steder, hvor der forekommer skråninger (**GAMINGSCAN, 2020**).

Hermed er der blevet redegjort for, hvordan de forskellige elementer af vores spilverden er blevet implementeret, hvorfra vi nu vil se nærmere på de forskellige årsager til implementering, med henhold til empirien fra vores undersøgelse af kadetterne fra officersskolen.

6.5 Hvorfor har vi lavet spilverdenen som den ser ud?

Vi vil nu tage udgangspunkt i de designovervejelser, vi på forhånd havde gjort os, samt det design, vi faktisk har implementeret og holde det op mod det kadetterne fortalte os, i de interviews vi har foretaget omkring deres overvejelser om, hvad spillet skal kunne, og hvad det skal indeholde.
Den indledende idé til spildesignet var at lave en simulering med fly og faldskærm, som minder mest om den situation kadetterne vil blive udsat for. Under interviewet med Officersskolen ændrede denne idé sig, da flere kadetter udtalte sig om, at det ikke nødvendigvis ville gøre oplevelsen mere virkningsfuld i forhold til deres situation, da de blev adspurgt:

Kadet:

"Det kommer an på, hvad der ligesom er formålet med jeres projekt, og det kommer også an på de, altså modtagere I har, og jeg tror for mit vedkommende, vil det være at tage skridtet ud af flyet, så det ville være den del, og det er derfor jeg tænker at her med jægertårnet ville det være effektivt i hvert fald for mit vedkommende."

(Bilag 2)

Samtidig ville designprocessen i dette scenarie være banalt at opstille, da det umiddelbart er svært at opbygge en spænding i forhold til højden i et fly, hvor der er lukket af for udsigten. Designet har derfor i højere grad taget form omkring Richie's Plank Experience, da de fleste kadetter syntes, at dette design grundlæggende fungerede rigtig godt i forhold til at udfordre dem, da de blev adspurgt, kort efter de havde prøvet spillet:

Kadet:

"Jeg tror, vi blev overrasket over i hvert fald os, der var på lidt til sidst at, hvis man stod og så på det og tænkte ja, men lige så snart man står der selv, jeg kan mærke, at jeg ryster stadigvæk, og man ved jo godt, at det kun er tre centimeter, du skal hoppe, men jeg blev fucking skræmt alligevel."

(Bilag 2)

Vi har derfor valgt at drage inspiration fra Richie's Plank Experience, eftersom det var dette spil kadetterne prøvede. Interviewet vi havde med dem efterfølgende tog også form af netop dette spil, da det var mange af kadetternes eneste reference til VR. En af kadetterne fortalte blandt andet, at hun synes, det fungerede rigtig godt med elevatoren, hvor der var en lille revne mellem de to døre, således "(…) man får den der fornemmelse af, at du kommer op ad (…)" (bilag 2). En anden kadet tilføjede derefter, at det var fedt at spændingen blev bygget op, men efterspurgte, om der kunne laves glas hele vejen rundt i elevatoren for at styrke effekten. Det var en af grundene til, at vi valgte vogne fra pariserhjulet, da det gav et 360 graders udsyn.

Selve Richie's Plank Experience er ikke en længerevarende oplevelse, hvoraf en af kadetterne efterspurgte et produkt, hvor VR-oplevelsen gerne skulle være længere end 15-20 sekunder i varighed; "(…) hvis man virkelig skal have fornemmelsen i kroppen, og man skal ind og påvirke alt, hvad man kan fysiologisk i kroppen (…)" (bilag 2). Samtidig skal oplevelsen også være lidt længerevarende for at " (…) have lov til at opleve oplevelsen og sluge indtrykkene til sig, så man kan få den der fornemmelse af, hvad er det jeg har behov for at arbejde med, når det er, at jeg begynder at mærke, at jeg bliver påvirket (…)".

Vi har lavet scenen i spillet således, at brugeren af spillet står i en park, hvor der er en by med et pariserhjul og bygninger omkring som vist i videoen og på figur 51. Bygningerne skal visualisere højde i forhold til, når brugeren også selv kommer op i højden - på denne måde afspejler omgivelserne, hvor højt oppe brugeren føler sig at være, på trods af brugeren i virkeligheden stadig står på et gulv. Vi havde derudover mange overvejelser om, hvilken form for objekt der skulle transportere brugeren op ad – vi er endt ud med at vælge en vogn fra pariserhjulet, da denne har

den passende størrelse i forhold til, at brugeren kan kunne bevæge sig ud og ind ad denne, og samtidigt være indenfor vagtsystemets grænser. Pariserhjulvognen har vi tilegnet egenskaber, så denne kan fungere som en elevator og få brugeren op i højde. Pariserhjulvognen har vi også modificeret, således der ikke er nogen døre i - på denne måde øges anspændelsen sandsynligvis for brugeren i vognen, da fornemmelsen af eventuelt at kunne falde ud og ikke ville falde ud stiger.

Figur 51: *Scenen*

Scenen i Unity projektet har udformet sig ud fra de designovervejelser, som gruppen har gjort sig i samarbejde med kadetterne fra Hærens Officersskole (billedet er et skærmbillede af eget projekt).

Vi har derudover lavet flere platforme, således den første pariserhjulvogn får spilleren op til det første stadie, hvorefter spilleren enten kan gå ind i en anden vogn og blive transporteret endnu højere op eller tage springet ud fra bygningen. Dette var noget flere af kadetterne ønskede, da vi spurgte om, hvilke former for spilmekanikker, der med fordel kunne implementeres i vores program:

Kadet:

"Det kunne også være level baseret tænker jeg jo, et til ti og så og så dem der kommer de næste 10 eller et eller andet."

(Bilag 2)

Kadet:

"Man kan måske stille og rolig skalere op, sådan så man starter for et eksempel på planken, lad os sige en planke derinde, der så går på at næste skridt måske er at tage en tre skridt op" (...)
(Bilag 2)

En anden kadet gav også inspiration til delen med at spillet skulle have forskellige niavuer gennem idéen om at der også skulle være noget frihed til spilleren, så de selv skulle kunne træffe et valg om at hoppe ud eller ej, da de blev spurgt hvad de syntes om højder generelt:

Kadet

"Det er ikke behageligt vil jeg sige, det er det jo ikke at stå deroppe, du får den der sus, som gør at man sådan står og okay, tør jeg det her, tør jeg det her, tør jeg det her. Men det handler jo om at skubbe sig selv ud i det, ikke."
(Bilag 2)

Dette blev derfor kilden til at bruge de forskellige Container objekter, som forskellige stadier, spilleren kunne benytte sig af, som både var til for at opbygge en form for spænding idet at komme højt op, men samtidig for at få spilleren til at vurdere egne evner.
Tør spilleren derved at tage den næste pariserhjulvogn og derved komme endnu højere op, vil de derfor skulle tage en ny beslutning om at de så skal tage den næste vogn derefter.
For netop at kunne gøre dette bedst muligt, skal scenen nødvendigvis være præget af et virkelighedstro terræn - dette har kadetterne også selv beskrevet i de interviews, vi har foretaget med dem:

Kadet:

"Når jeg tænker over det, så er det lige så vigtigt, at der bliver tilføjet de elementer som vind eller noget, du ikke rigtig ved hvad er. Det er de ting, som er så meget, bare det er virkelighedstro og så tæt på som overhovedet muligt."
(Bilag 2)

Vi har derfor fokuseret på at gøre terrænet så detaljeret som muligt, da det er denne del af programmet, vi har haft mulighed for at gøre virkelighedstro. Samtidig skal vi også prioritere at få implementeret passende scripts til funktionaliteten af programmet.

Ud fra dette har vi præsenteret en analyse af de dele af vores empiriske resultater, som har givet grundlag for VR-spillets design karakteristik, og er derved repræsentativt ud fra de subjektive forhold vores interview har bragt. Samtidigt har analysen givet nogle af de elementer, der har givet programmet væsentlighed i forhold til projektets problemformulering. Desformedelst er der blevet skabt et grundlag for, hvorledes projektets genstand kan tage form. Herfor vil vi se nærmere på, hvorledes dette program skal tages i brug af kadetterne ved en eventuel implementering.

7.0 Brugervejledning

Her vil vi beskrive, hvordan vores program kører ved brug af Oculus Questen. Ydermere vil vi inkludere billeder, som viser et eksempel på, hvordan vagtsystemet opstilles. Brugervejledningen tager udgangspunkt i, at spillet allerede er downloadet til Oculus Questen.

Fysisk sikkerhed er vigtigt, selvom der rejses ind i en virtuel verden. Derfor benytter Oculus Questen et såkaldt vagtsystem, som brugeren skal sætte op inden brug af produktet. Vagtsystemet fungerer som en virtuel indhegning, brugeren kan se inde i headsettet, og skal opsættes således at det matcher det ryddede spillerum brugeren har. Under opsætning af vagtsystemet kan brugeren stadig se den fysiske verden omkring sig igennem brillen, ved hjælp af passthrough. Passthrough er et kamera på ydersiden af brillen, der sender et livefeed af omgivelserne til brugeren under opsætning.

I første step ved opsætning af vagtsystemet tages Questen på, og den ene controller holdes mod gulvet. Når controlleren holdes mod gulvet, registreres bunden af spilleområdet. Derefter registreres spilleområdets ydre parameter ved at pege controlleren rundt mod det yderste af det ryddede område (se figur 52). Brugeren kan således optegne spilområdets ydre vægge med controlleren.

Nu er vagtsystemet opsat og spilleren kan begive sig ind i en virtuel verden. Når der spilles, kan den virtuelle indhegning ikke ses før spilleren kommer i nærheden af den. Der kan ændres på, hvor tæt spilleren skal være på den virtuelle indhegning, før den viser sig. Det er også vigtigt at gøre opmærksom på, at vagtsystemet ikke vil kunne registrere kæledyr, eller folk der bevæger sig indenfor spilleområdet under brug. Der skal derfor benyttes et område som brugeren er sikker på, ikke bliver obstrueret.

Figur 52: *Vagtsystemet*

Således indstilles gulv-niveauet og vagtsystemet. Skærmbilleder er fra videoen https://www.youtube.com/watch?v=pVFcsedWsKE

For at køre vores VR-produkt på Questen skal det være downloadet ned på headsettet, derefter kan det findes i Libraries under Unknown Sources. Vi anbefaler et spilområde på fem kvadratmeter for at få den bedste oplevelse.

Ved start af vores VR-oplevelse, skal spilleren stå i midten af spilområdet og bruge controlleren til at placere sig direkte foran åbningen på den første vogn. Herefter kan spilleren gennemføre hele oplevelsen, ved at bevæge sig fysisk rundt i spilområdet uden at benytte controlleren. Vores VR-oplevelse er meget intuitiv. Det eneste spilleren skal gøre, er at bevæge sig fra den ene vogn til den anden og trykke på knapperne.

8.0 Afprøvning

I løbet af processens udvikling har vi afprøvet vores spil internt. Vi startede et samarbejde med kadetterne fra Hærens Officersskole i starten af marts, hvor vi aftalte med dem, at de ville få lov at følge processen tæt samt få mulighed for at mødes, så de kunne afprøve produktet løbende og endeligt. Det havde været det mest optimale, hvis vi havde fået kadetterne til at afprøve vores spil, da vi ønsker at lave et produkt, der kan hjælpe dem – på grund af Covid-19 kunne dette dog ikke lade sig gøre. I løbet af processens udvikling har vi derfor afprøvet vores spil internt i stedet. Vi har testet spillet ud fra et Black-box-testnings princip, som er en metode til softwaretestning, hvor vi udelukkende fokuserer på funktionaliteten af programmet og dermed ikke fokuserer på dets interne strukturer. Dette er på trods af, at alle gruppemedlemmer har haft indsigt i softwaren.

Ved afprøvning satte vi computeren til at kunne vise de resterende gruppemedlemmer, hvad testpersonen af programmet kunne se i VR-brillen. På den måde kunne vi nemt kommunikere om, hvad testpersonen så i programmet, i forhold til hvad der var godt, og hvad der forekom mindre godt.

Under vores første afprøvning af spillet brugte vi meget tid på at forsøge at vurdere, hvor meget plads der var brug for omkring spilleren, samt at justere afstandene inde i selve spillet til afstande, som spilleren fysisk kunne bevæge sig rundt i. Hvad vi fandt ud af gennem denne test af vagtsystemet var, at vi blev nødt til at placere de forskellige pariservogne tæt på hinanden for at undgå at spilleren bevægede sig ud af vagtsystemet samtidigt med, at spilleren ikke skulle bruge controllerne til at bevæge sig med. Ved andre tests af spilverdenen benyttede vi os af controllers til at bevæge os rundt i, men for at opnå den mest optimale virkelighedsfornemmelse ønskede vi, at spilleren kunne bevæge sig rundt i spillet fysisk. Vi testede spillet med et areal på fire gange tre meter, hvor spilleren lige akkurat havde mulighed for at bevæge sig rundt ved udelukkende fysisk bevægelse uden at blive begrænset af vagtsystemet. Derfor kom vi frem til, at den bedste oplevelse med programmet ville fungere bedst på et areal af fem gange fem meter, da dette ville give det ekstra frirum til eventuelle fejl ved opsætningen af spilarealet.

Under vores anden afprøvning oplevede det gruppemedlem, der afprøvede spillet, at knappen i pariserhjulvognen, der skal trykkes ned for at igangsætte elevationsanimationen, blev aktiveret lidt for hurtigt i forhold til, hvornår personen rent faktisk havde hånden hen over knappen. Denne funktion krævede derfor mere præcision.

Derudover oplevede testpersonen også, at når elevationsanimationen gik i gang, begyndte hele spilverdenen at ryste – dette kaldes jitter. Hvis programmet jitter, er det svært for brugeren af programmet at få opbygget denne spænding, da det ødelægger indlevelsesmomentet (**JENSEN & KONRADSEN, 2017, 6**). Vi har muligvis lokaliseret fejlen i softwaren til denne problemstilling, men vi har endnu ikke fundet en brugbar løsning.
Efter at have lavet disse test kunne vi ændre nogle småting, som gjorde oplevelsen af spillet bedre. Dog blev det nødvendigt for os at nedsætte animationselevationens hastigheden gevaldigt, så det netop ikke jitter, når vognen kører op. Dette får konsekvensen af en meget langtrukken tur op med den tredje og sidste pariserhjulvogn, da vi sigter efter at nå op til 400 meters højde, som er den højde, kadetterne skal springe fra, når de laver selve faldskærmsudspringet (bilag 3).

Da vi havde fået testet vores program internt og var tilfredse med resultatet, fik vi mulighed for at optage en video af resultatet, som også agerer som forlængelse af afprøvningen – videoen ligger på YouTube, og der er link nederst i rapporten under litteraturlisten ved "Filer til download". Til selve optagelsen har vi benyttet open broadcaster software (OBS), som er et skrivebordsoptagelsesværktøj. Vi har benyttet en green screen til at have bag testpersonen, således vi havde mulighed for at bygge videoen op af flere lag – baggrunden udgør en video af det testpersonen ser i VR-brillen, og i forgrunden ses selve testpersonen i aktion i spilverdenen. Videoen er et eksempel på en afprøvning, som skal gøre forståelsen af programmet nemmere for læseren af denne rapport.
Ved denne test samt optagelse var det ikke muligt for os både at få lyden til at blive optaget af OBS og samtidig få lyden til at spille i VR-brillen. Vi valgte derfor, at testpersonen ved denne optagelse ikke skulle kunne høre de lydeffekter, vi har sat ind i spillet, da det var vigtigere at lydeffekterne blev repræsenteret i videoen og derfor optaget af OBS.

9.0 Diskussion

Vi vil nu diskutere processen i arbejdet for projektet, samt diskutere vores produkt sammenholdt med de interviews vi afholdt med kadetterne. Ydermere vil vi diskutere, om der er ting, vi kunne have gjort anderledes eller forbedret både med henhold til processen og selve produktet.

9.1 Processen

Helt fra start af valgte vi at lave to forskellige Kanbanboards for henholdsvis skriveprocessen og programmeringen. Dette gjorde vi for at kunne skille de to ting ad og derved have bedre overblik over, hvad der var lavet/skrevet, og hvad der ikke var, men skulle igangsættes. Vi har valgt Kanban fremfor Scrum, da vi fandt denne agile metode at føre projektudarbejdelsen bedre for vores situation. Dette er blandt andet baseret på baggrund af de nærværsmøder Scrum forudsætter, som ikke har kunne lade gøre under Covid-19. Derudover har vi fundet det passende, at vi ikke nødvendigvis er styret af sprint perioder, som Scrum er, men at vi løbende hele tiden har kunne genoverveje situationen i projektet. I Scrum er der typisk én, der er ansvarlig for hele processen, og her har vi også bedre kunne tilslutte os Kanban metoden, da vi som gruppe ønsker, at hele teamet skal være ligeværdige og ansvarlige. Vi har i høj grad også været begrænset af egne evner i forhold til arbejdet med Unity, da vi på grund af Covid-19 ikke har kunne opsøge den samme form for hjælp. I den forstand har Kanban også været en bedre løsning for os, da vi ikke har haft mulighed for at gøre det hurtigere, end den tid det tog for at tilegne os den nødvendige viden. Dette både i forhold til Unity som spilmotor og kodning i C#, som er helt nyt sprog, vi har skulle lære. Det har været svært at sætte tidsestimater på opgaver, vi ikke har tidligere erfaring med.

Generelt kan det diskuteres, om Scrum havde været en bedre løsning som agil projektstyring for os. Ved Scrum er der bedre kontrol over projektets forløb, men der skal samtidig også bruge tid på at lede projektet den rigtige vej. Ved Kanban er projektforløbet mere flydende, og kontrollen kan derfor være svær at opretholde. Scrum kunne i nogle tilfælde have sat bedre rammer for projektforløbets gang – hvis vi havde haft et tæt forløb med kadetterne som "kunde", havde dette været fordelagtigt, idet vi også bedre ville være i stand til at oplyse dem om, præcist hvor langt vi er i processen. Kadetterne ville på denne måde være bedre klar over, hvad der er blevet implementeret i programmet, og dermed også bedre kunne komme med input til videreudvikling.

Det skaber gennemsigtighed i produktudviklingen, hvilket medfører en form for tryghed og tillid for både kadetterne og internt i gruppen.

9.2 Forbedringer

Hvis vi havde haft endnu mere tid, kunne det have været interessant at koble spillet til en open-source elektronisk platform, som kunne gøre det muligt at måle pulsen på brugen af spillet. Med dette ekstra redskab, ville vi have mulighed for at udvikle spillet i en retning, hvor spillet benyttede pulsen af spilleren som input. For højdeskræk gælder, at individet umiddelbart har en rationel skræk og tankegang, hvoraf individet bliver stresset, og ens puls stiger. Udgangspunktet må derfor nødvendigvis være, at hvis spilleren har en høj puls, er spilleren stresset, og herfor skal sværhedsgraden for spillet sænkes, da vi med spillet ønsker at skabe et roligt sind omkring uvante situationer. Hvis spilleren allerede er stresset, har spilleren derfor ikke brug for flere stressfaktorer. For eksempel hvis spilleren har en høj puls, er der ikke behov for, at vognen begynder og gynge og køre så højt op. Hvis spilleren forholder sig forholdsvis rolig kan flere stresselementer gradvist introduceres, som at vognen skal begynde gynge, eller køre hurtigere, det kan begynde at regne osv. Præcis hvilke elementer, der skulle begynde, er svært at sige, men der ville vi også kunne benytte pulsmåleren i vores afprøvning af programmet og få noget konkret data på, hvilke spilelementer der rent faktisk virker, og hvilke der er mindre gode. Vi ville også benytte denne open-sourceplatform til inkorporere eksterne faktorer som en blæser til at tænde på det korrekte tidspunkt.

Ved brug af pulsmåler kunne det overvejes at lave en database, hvori data overføres. Det ville både kunne benyttes til at implementere gamification, hvor jo lavere puls jo flere point ville der opnås og derved motivere spillere til at blive ved med at spille og øve sig i at forholde sig roligt i uvante situationer. Det ville også kunne benyttes af holdlederne til at få overblik over hver enkel kadet og føre statistik over deres fremskridt og tidligt i processen udpege, hvilke kadetter der måske skal arbejdes lidt mere med. Der ville også kunne overvejes, om databasen skal være tilgængelig for alle kadetterne, så der kommer konkurrence dem imellem, eller om resultaterne kun skal være tilgængeligt for den enkelte kadet, så de kan arbejde med sig selv, uden at blive påvirket af de andre.

Derudover har produktet taget meget form af netop det spil, kadetterne prøvede, som er Richie's Plank Experience. Eftersom mange af dem ikke havde prøvet VR før, blev de påvirket af indtrykket af spillet. Det kan diskuteres, om spillet havde taget form på en helt anderledes måde, hvis vi havde taget udgangspunkt i et andet spil, hvoraf kadetterne havde fået et andet indblik i, hvad VR egentlig kan. Ydermere blev de måske også påvirket af det interview, vi havde med, hvori vi i nogle tilfælde stillede dem ledende spørgsmål (bilag 2). Vi kan altså have haft indflydelse på deres svar i forhold til, hvad vi selv har haft af designovervejelser.

10.0 Konklusion

I dette projekt har vi undersøgt, hvordan en VR-simulering kan skabes gennem spilmotoren Unity og hvordan sådan en udviklingsproces struktureres. VR-simuleringen er lavet med det formål at skulle kunne afhjælpe højdeskræk hos kadetter i Hærens Officersskole.

Unity er et brugervenligt program, der har alle de nødvendige remedier for at skabe en vellykket VR-oplevelse. Unity er dog også et stort program med mange aspekter, som gruppen har skulle sætte sig ind i for at kunne udvikle et godt produkt. I starten af udviklingsprocessen tog det lang tid at få lavet de ting, vi ønskede, men i takt med at vi blev bedre bekendt ved Unity, øgedes effektiviteten også. Læringsprocessen ved Unity blev også fremskyndet af den store mængde information, der findes på nettet i form at YouTube-videoer og manualer på Unity programmet. Udover det gjorde Unity Asset Store den visuelle designproces hurtigere, da vi i gruppen har kunnet downloade færdig lavede 3D-modeller til vores scene i Unity.

Vores brug af en kvalitativ tilgang til metode gennem de forskellige typer af interviews, bragte de informationer om problemstillingen vi i første omgang ønskede. Ydermere gav denne form for undersøgelse også inspiration til, hvordan programmet skulle se ud gennem de empiriske resultater, metoden frembragte. Herfor var der med denne metodiske tilgang tale om, at der gennem samtaler i forskellige lag kunne skabes en fælles fornemmelse af, hvordan et virtual realityprogram skulle se ud, hvoraf principper om programmets design kunne etableres.

Udviklingsprocessen af VR-oplevelsen valgte vi at strukturere ved hjælp af Kanbanboard. I gruppen fandt vi, at Kanbanboardet hjalp til at effektivisere og strukturere arbejdsprocessen, samt at give overblik over, hvad de forskellige gruppemedlemmer arbejder på, og hvad der mangler at blive gjort.

Fobier kan behandles med VR ved at udsætte brugeren for angstkilden i et sikkert miljø. Formålet med vores produkt var at behandle akrofobi (højdeskræk) med kadetterne fra Hærens Officersskole som målgruppe. Det havde været ideelt at kunne afprøve vores produkt på kadetterne fra Hærens Officersskole, men grundet Covid-19 var vi nødsaget til at teste internt i gruppen. Vi afprøvede og evaluerede således programmet i løbet af udviklingsprocessen. Vi vil i gruppen vurdere, at vores VR-oplevelse vækker en følelse af at være højt oppe i luften. Et aspekt ved VR-oplevelsen, som virkede mindre godt, var jitter problemet. Jitter problemet opstår, når der køres op i vognene

(elevatorerne), og kan medføre at indlevelsen i VR-oplevelsen forringes. Det lykkedes dog at fjerne det værste jitter ved at køre vognanimationerne langsommere. I gruppen vil vi derfor konkludere, at vores produkt har opnået den ønskede virkning med plads til forbedring. For at afslutte udviklingen af programmet kunne det være en god ide af afprøve programmet med kadetterne gennem videre undersøgelser.

11.0 Refleksion

Vi har forsøgt at skabe et program, som har til formål at skabe en fornemmelse af højde, således kadetterne på Hærens Officersskole kunne arbejde med højdeskræk. Dette kræver, at der bliver skabt en virkelighedsfornemmelse. En måde, hvorpå vi kunne øge virkelighedsfornemmelsen på, ville være at gøre brug af eksterne dimensioner, hvilket kunne være en blæser eller en planke, som spilleren skulle gå på. I eksemplet med blæseren ville pointen være at koble den sammen med vores AudioSource, som skaber en blæsevejrslyd, således spilleren både kan høre og føle blæsten. Med eksemplet med en planke spilleren skal gå på, ville pointen være, at spilleren dermed ville kunne mærke kanterne og dermed fysisk kunne mærke begrænsningerne. For at finde ud af hvilke eksterne dimensioner, der ville give mening at bruge til vores program, ville det kræve, at vi i samarbejde med kadetterne, som programmet er udviklet til, testede de forskellige eksterne dimensioner og evaluerede på, hvilke dimensioner der var med til at skabe virkelighedsfornemmelsen. Dette kunne dog ikke realiseres, eftersom Covid-19 pandemien satte en begrænsning på antallet af mennesker, der måtte samles, samt mente vi ikke, at det var forsvarligt at lade flere forskellige mennesker bruge det samme VR-headset til at teste spillet. Så hvis Covid-19 pandemien ikke havde været en faktor, ville vi derfor gerne forsøge at bruge kadetterne løbende til at komme med input til, hvilken fornemmelse de fik ud af programmet. Udover at komme med feedback, til hvordan virkelighedsfornemmelsen var, kunne kadetterne også bruges til at komme med ideer til, hvordan programmet skulle have udviklet sig. Derfor kunne programmet muligvis have set anderledes ud, hvis vores testpersoner havde haft muligheden for at komme med input.

I forhold til vores egen arbejdsproces har vi grundet Covid-19 pandemien ikke været i stand til at gøre brug af de ressourcer, som RUC har til rådighed i en længere periode, hvilket betød, at vi var nødsaget til at udvikle programmet uden for RUC og dermed uden RUC's ressourcer i denne periode. Det besværliggjorde processen utrolig meget, da vi i starten ikke kunne mødes på RUC.

I forhold til at gøre terrænet mere virkelighedstro, beskrev vi, hvordan vi benyttede en algoritme til at re-orientere græsset og derved bryde det ensartede billede ved hjælp at Microsplat, som er et asset, vi skulle betale penge for. Inden da var det meningen, at vi ville kode og implementere en algoritme selv, og vi brugte rigtig lang tid på det, hvilket var virkelig ærgerligt. Vi fandt en del

løsninger i programmeringssproget GLSL, som vi prøvede at konvertere til HLSL. Det er muligt, da de minder ret meget om hinanden, og det er kun HLSL som Unity understøtter. Selve konverteringen gik fint, og vi fik ingen error messages i Visual Studio, og Unity genkendte også shaderen, men da vi valgte den gjorde det absolut ingenting, hvilket var meget frustrerende, og vi endte aldrig med at finde ud af hvorfor. Det resulterede i, at vi i sidste ende havde spildt alt den tid, vi havde brugt. Set tilbage på det var det måske ikke den bedste prioritering af tid, da selve algoritmen fra det importerede asset højst sandsynligt ville give et flottere slutresultat rent visuelt, men det havde været virkelig fedt for vores projekt at kunne implementere og beskrive algoritmen selv og spare pengene - derfor prioriterede vi det.

Der er mange måder at løse problemstillingen på og mange scenarier, hvorpå der kan simuleres højde. Nogle af kadetterne foreslog, at vores spilverden skulle forestille deres træningsbane, hvor der er et tårn, der kaldes Jægertårnet, som de springer ud fra (se bilag 2). Det ville måske skabe en mere realistisk oplevelse, eftersom dette scenarie er relaterbart for dem og vil give den samme form for vægtløshed, der opleves ved udspring fra højde generelt. Vi valgte dog ikke at implementere dette, da vi gennem vores program gerne ville træne kadetterne til at overkomme deres frygt i ukendte rammer, eftersom det bedre afspejler, hvordan deres træning skal benyttes i fremtiden. I en virkelig situation, hvor det gælder, bliver de også bedt om at springe ud i det uvisse og ikke deres vante træningsbane.

Vi ville have gavn af muligheden for at teste programmet med kadetterne og høre, hvilken måde de oplever programmet. Med denne test og opfølgning af vores første interview ville vi forhåbentligvis få yderlige informationer omkring hvilke elementer, der skal videreudvikles.

12.0 Litteraturliste

Unity kilder:

- Unity Technologies. (2020, May 7). Unity Hub. Retrieved May 13, 2020, from https://docs.unity3d.com/Manual/Gettin,,gStartedUnityHub.html

- Unity Technologies. (2020, May 7). Starting Unity. Retrieved May 12, 2020 from https://docs.unity3d.com/Manual/GettingStarted.html

- Unity Technologies. (2017, July 12). Input for Oculus. Retrieved May 26, 2020 from https://docs.unity3d.com/560/Documentation/Manual/OculusControllers.html?fbclid=IwAR3FnG1C-o5PP3V5FMwsGQlZTdUPelR5JX2hh07hPqppNlDwfoouCXJ3BB0

- Unity Technologies. (2020, May 7). Unity's Interface. Retrieved May 13, 2020, from https://docs.unity3d.com/Manual/UsingTheEditor.html

- Unity Technologies. (2020, May 15). Universal Render Pipelines overview. Retrieved May 15, 2020, from https://docs.unity3d.com/Packages/com.unity.render-pipelines.universal@8.1/manual/index.html

- Unity Technologies. (2020, May 7). Asset workflow. Retrieved May 11, 2020, from https://docs.unity3d.com/Manual/AssetWorkflow.html

- Unity Technologies. (2020, May 7). Common types of Assets. Retrieved May 11, 2020, from https://docs.unity3d.com/Manual/AssetTypes.html

- Unity Technologies. (2020, May 15). Unity Collaborate. Retrieved May 15, 2020, from https://docs.unity3d.com/Manual/UnityCollaborate.html

- Unity Technologies. (2020, May 15). The Hierarchy Window. Retrieved May 18, 2020, from https://docs.unity3d.com/Manual/Hierarchy.html

- Unity Technologies. (2020, May 21). Prefabs. Retrieved May 26, 2020, from https://docs.unity3d.com/Manual/Prefabs.html

- Unity Technologies. (2020). About the XR Plug-in Management package. Retrieved May 18, 2020, from (https://docs.unity3d.com/Packages/com.unity.xr.management@3.2/manual/index.html

- Unity Technologies. (2020, May 15). Creating and Using Scripts. Retrieved May 18, 2020, from https://docs.unity3d.com/Manual/CreatingAndUsingScripts.html?_ga=2.216135894.157531698.1589810484-1367491362.1581503491

- Unity Technologies. (2020, May 15). GameObjects. Retrieved May 18, 2020, from https://docs.unity3d.com/Manual/GameObjects.html

- Unity Technologies. (2020, May 15). GameObject. Retrieved May 18, 2020, from https://docs.unity3d.com/Manual/class-GameObject.html

- Unity Technologies. (2020, May 15). Primitive and placeholder objects. Retrieved May 18, 2020, from https://docs.unity3d.com/Manual/PrimitiveObjects.html

- Unity Technologies. (2020, May 28). Primitive and placeholder objects. Retrieved May 29, 2020, from https://docs.unity3d.com/Manual/Shaders.html

- Unity Technologies. Class XRBaseInteractable. Retrieved May 22, 2020, from https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@0.9/api/UnityEngine.XR.Interaction.Toolkit.XRBaseInteractable.html

- Unity Technologies. (2020, May 21). MonoBehaviour. Retrieved May 22, 2020, from https://docs.unity3d.com/ScriptReference/MonoBehaviour.html

- Unity Technologies. (2020, May 21). Behaviour. Retrieved May 22, 2020, from https://docs.unity3d.com/ScriptReference/Behaviour.html

- Unity Technologies. (2020, May 21). Component. Retrieved May 22, 2020, from https://docs.unity3d.com/ScriptReference/Component.html

- Unity Technologies. (2020, May 21). Object. Retrieved May 22, 2020, from https://docs.unity3d.com/ScriptReference/Object.html

- Unity Technologies. (2020, May 21). Scripting. Retrieved May 22, 2020, from https://docs.unity3d.com/Manual/ScriptingSection.html

Hjemmesider:

- Advanz, Hvad er Kanban. (n.d.). Retrieved May 13, 2020, from https://advanz.dk/blog/kanban-board-kanbanflow/

- Atlassian, & REHKOPF, M. (n.d.). What is a kanban board? Retrieved May 13, 2020, from https://www.atlassian.com/agile/kanban/boards

- Forskerzonen, Videnskab.dk, Fremtidens angstbehandling foregår i Virtual Reality. Skrevet af Mia Beck Lichtenstein. Sidst updateret 14 Juni 2017. Link: https://videnskab.dk/teknologi-innovation/fremtidens-angstbehandling-foregaar-i-virtual-reality

- Gamingscan. (2020, May 8). What is Anti-Aliasing? [Simple Explanation]. Retrieved June 2, 2020, from https://www.gamingscan.com/what-is-anti-aliasing/

- Kvalitative dybdeinterview | Fremforsk, Center for fremtidsforskning. (n.d.). Retrieved May 6, 2020, from https://www.fremforsk.dk/kvalitative-dybdeinterview/

- Modern War Institute, MWI PODCAST: FOUR COMBAT JUMP VETERANS TELL THEIR STORIES, Retrieved May 31, 2020, from https://mwi.usma.edu/mwi-podcast-four-combat-jump-veterans-tell-stories/

- Psykologisk Ordbog. (n.d.). Hvad er eksponeringsterapi? | PJKP. Retrieved June 2, 2020, from https://www.pjkp.dk/ordbog/eksponeringsterapi/

- Region Hovedstadens psykiatriske hospital, & Andreasen, M. (2019, November 6). Virtual Reality – når teknologien hjælper patienter i virkeligheden. Retrieved June 2, 2020, from https://www.psykiatri-regionh.dk/presse-og-nyt/Nyheder-og-pressemeddelelser/Sider/Virtual-Reality-%E2%80%93-n%C3%A5r-teknologien-hj%C3%A6lper-patienter-i-virkeligheden---.aspx

- Richies Plank Experience. (n.d.). Retrieved May 6, 2020, from https://store.steampowered.com/app/517160/Richies_Plank_Experience/

- Tutorials Point, Unity – Tranforms and Object Parenting, Retrieved May 18, 2020, from https://www.tutorialspoint.com/unity/unity_transforms_and_object_parenting.htm

Bøger

- Kvale, S., & Brinkmann, S. (2016). *Interview: det kvalitative forskningsinterview som håndværk* (3rd ed.). Bern, Switzerland: Hans Huber. (ISBN: 978-87-412-6377-9)

- Tanggaard, L., & Brinkmann, S. (2015). *Kvalitative metoder og tilgange* (2nd ed.). Bern, Switzerland: Hans Huber. (ISBN: 978-87-412-5904-8)

Artikler

- Lasse X Jensen & Flemming Konradsen (2017). *A review of the use of Virtual Reality head-mounted displays in education and training.*

Filer til Download

- Hænder filer fra YouTube brugeren Valem, hentet på google drive https://drive.google.com/file/d/1J0eCepiQ7A47NjfFYhBBaw8lLqO8YmRZ/view?fbclid=IwAR0_fXgsB77vA_79O1s516JZzswOBlOx09Tfb7-z4EfxKg0vf2neqHYbqDM

- Controller filer fra YouTube brugeren Valem, hentet på google drive https://drive.google.com/file/d/1HXBYHHg9KPdgVYF6zY1HLBmp6k4OsYcz/view?fbclid=IwAR15ToQbL-pInWRbFSh7f5afcC_nfTJvBN28gDG3nyukgFfr4K426CRJ3v4