The Maze Chase

Et Virtual Reality spil udviklet i Unity

4. Semester. Gruppenr. S2026631262

Cecilie Higson Wilde (66390), Charlie Valente (66369), Jonas Tommy Petersen (68894) & Olivia Skytte (66382)

Vejleder: Martin Kirkegaard

Abstract

During this semester the group has begun development on a VR horror title adequately named ‘The Maze Chase’. As newcomers to Unity we have learned the inner workings of this editor to profit us in the development of our game, and by frequent testing in the borrowed Oculus Quest headset we have strived to make it a playable experience with room for further development if so desired. There have been obstacles on the way in the shape of the nation-wide quarantine following the outbreak of COVID-19, but the group has done their best to keep morale high and still give their best for this project. In this report you can expect to see our reasonings for choices in the development process, explanations for scripts and AI choices as well as a brief overview of the troubles we encountered coding and testing this game.

Indholdsfortegnelse

The Maze Chase **1**

Abstract **2**

Indholdsfortegnelse **3**

Indledning **5**

Problemformulering **6**

Arbejdsspørgsmål 6

Brugervejledning **6**

Introduktion til spillet 6

Spillerkontrol 8

Brug af Kanban board **8**

Hvad er et Kanban board? 8

Hvordan har gruppen anvendt Kanban Board? 9

Hvad har gruppen fået ud af at bruge Kanban Board? 11

Brug af logbog **12**

Teknikken bag **14**

Unity 14

Monobehaviour 18

Oculus Quest 20

Oculus Integration 21

Bevægelse af spilleren 21

AI adfærd 22

AI adfærd i vores spil 23

Tekniske Udfordringer 23

Problematikker under programmering 24

Se gennem vægge 24

Fastlåst på patrulje 25

Beskrivelse af programmet **25**

Kodning af Monster AI 25

NavMesh & NavMeshAgent 25

Script til EnemyFollow 27

Script til PlayerHealth 29
Script til Løb og Hop 30
Script til Baneskift og (Gen)Start af spillet 32

Testing **33**
Resultater fra første test 34
Opsummering af første test 38
Resultater fra anden test 39
Opsummering af anden test 41

Process og refleksion under fjernarbejde **42**

Refleksion **43**

Konklusion **44**

Litteraturliste **45**
Artikler 45
Bøger 45
Hjemmesider 45
Assets brugt fra Unity Asset Store 46
Journaler 47
Videoer 47

Indledning

Siden kickstarteren af Oculus Rift i 2012 (Sherman, 2018), har Virtual Reality været den helt store vinder hos de kommercielle brugere, da muligheden for fuld indlevelse i spilverdenen kan udføres i hver enkelt brugers private hjem.

Filmen 'The Matrix' som udkom i 1999 har siden sået ideen om at kunne træde ind i en anden verden eller virkelighed været fængende, mystisk men også spændende, og det ses helt tydeligt i måden spilverdenen har udviklet sig lige siden (Dempsey, 2019).
Længe før filmen 'The Matrix' udkom har mange videnskabsfolk arbejdet med ideen om at kunne 'træde ind' i en virtuel verden, dog har teknologien der skulle gøre det muligt været længe undervejs. Først i senere tid og efter filmen er der begyndt at vise sig teknologi som på bedre vis kan trække spilleren med ind i spillet og gøre spilleren til en essentiel del af spiloplevelsen.

Google streetview var den helt store inspirationskilde til at udvikle tredimensionelle verdener og udviklingen af bl.a. 3D konsoller tog fart (Barnard, 2019).
Da Oculus Rift og Oculus Quest udkom, blev det nu muligt for brugeren at kunne bevæge sig mere flydende i spillet, ved hjælp af et sæt controllere der kan styre karakteren (altså spilleren selv) rundt inde i spillet.

Gruppen lod sig inspirere af de mange labyrint spil der allerede eksisterer - spil som *'Spooky's Jumpscare Mansion'* har særligt været en stor inspirationskilde til gruppens idé om at lave et labyrintspil med en enemy AI der jager spilleren, da dette spil bygger på samme grundprincipper om at spilleren skal klare sig igennem en række levels, der undervejs stiger i sværhedsgrad, og bliver mere og mere udfordrende for spilleren, alt imens de jages igennem banerne af skræmmende små monstre.
Gruppen har også haft flere film med sig som inspirationskilde, bl.a. har filmen *'Harry Potter og Flammernes Pokal'* været en inspiration for spillet og udviklingen af selve labyrinten, da der også her gøres brug af labyrinter og transport nøgler, som videresender deltagerne/spilleren til næste labyrint og i gruppens tilfælde, videresendes spilleren til næste level.

Ydermere er ideen med Virtual Reality, hvor man med hjælp fra et par briller, er i stand til at træde ind i en anden virkelighed, et interessant koncept for gruppen, så derfor ønskede vi at kunne anvende denne teknologi til vores spil. Med denne teknologi vil spilleren kunne få en større indlevelse i spillet, og føle at de befinder sig i labyrinten og skal flygte fra monstrene.

Problemformulering

Hvordan skaber man et labyrintspil, til Oculus Quest, med tracking AI i Unity, og samtidig holder alle karakterer inden for væggenes grænser?

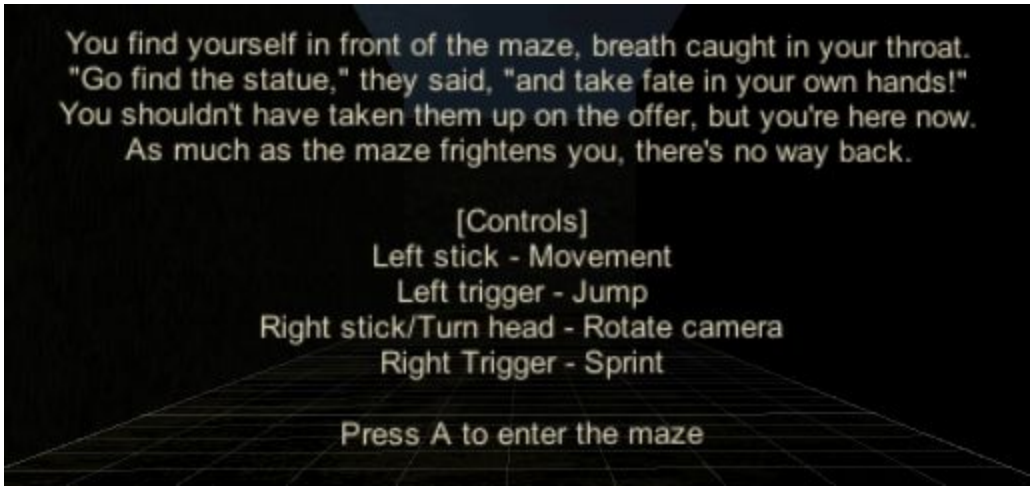
Arbejdsspørgsmål

- Hvordan kan brugen af Kanban og en logbog skabe struktur i arbejdsprocessen?
- Hvordan fungerer Unity?
- Hvad er Oculus Quest og hvordan kan det bruges i udviklingen af et VR spil?
- Hvordan fungerer en tracking AI, og hvordan kan vi bruge den i vores spil?
- Hvordan kan testing forbedre spiloplevelsen?

Brugervejledning

Introduktion til spillet

“The Maze Chase” er et Virtual Reality spil udviklet til Oculus Quest gennem platformen Unity. For at spille The Maze Chase er det nødvendigt at påføre sig et Oculus Quest headset og håndtere de medfølgende Oculus Touch controllere, da det ellers ikke er muligt hverken se spillet eller bevæge sig i det.



Introskærm til The Maze Chase, 28/05-2020

Når spillet startes, introduceres spilleren til en mørk labyrint uden muligheden for at bevæge sig. Spilleren kan se en tekst som introducerer spillets korte historie, og hvilke handlinger spilleren kan foretage sig i løbet af spillet. Her bliver spilleren også introduceret for sit mål; find statuen inde i labyrinten. Spilleren bliver ikke introduceret for et monster i denne skærm, hvilket er gjort med vilje for ikke at give spilleren en forventning om at blive jagtet - ønsket er at give en falsk fornemmelse af sikkerhed, som lyddesignet i spillet kan spille på til at gøre følelsen af rædsel større, når man endelig møder et monster.

Monstrene i The Maze Chase patruljerer de lange, mørke gange i labyrinten i et mønster som spilleren ikke kan se, men som igen øger risikoen ved at løbe rundt uden at se sig for. Selvom de kan høres på afstand, er det ikke til at sige hvorfra de kommer, og en uforsigtig spiller kan hurtigt finde sig selv fanget af et monster hvis de ikke passer på. Når spilleren fanges af et monster, vil de blive sendt tilbage hvor de kom fra, og må nu igen orientere sig i labyrinten før de kan forsøge at finde målet. Spilleren har 3 chancer for at finde statuen i hvert level, selvom der ikke er en indikation om dette - dette valg er bevidst idet en HUD (Heads up display) kan fjerne indlevelsen i spillet, så lyd og omgivelser ikke længere har den samme effekt. Hvis spilleren taber alle 3 chancer for at finde statuen, vil de blive sendt til en Game Over-skærm, hvor de får muligheden for at prøve igen ved at trykke på en knap. Hvis der trykkes på denne knap, vil de blive sendt tilbage til introduktions-skærmen, og spillet vil starte forfra. Dette gør essentielt at det er muligt at blive ved med at prøve spillet indtil man gennemfører, uden at skulle lukke spillet ned og starte det op igen, hvilket også øger chancen for at spilleren overhovedet har lyst til at prøve igen.

Spillerkontrol

Spilleren er blevet valgt til at have nogle få, men vitale muligheder for at bevæge sig: Med den venstre Oculus Touch controller kan spilleren bevæge sig frem, tilbage, til siderne og diagonalt ved hjælp af joysticket der sidder på oversiden. Det tilsvarende joystick på højre Oculus Touch controller kan snappe kameraet rundt, hvilket vil sige at kameraet kan dreje i et fastsat antal grader på et enkelt frame. Dette antal grader er fastlagt, og kan ikke ændres på, af spilleren.
På højre controller findes der desuden en knap foran, som kan trykkes ned og holdes i bund for at spilleren kan bevæge sig hurtigere; dette kan hjælpe spilleren med at slippe væk fra monstre, eller hvis de blot vil løbe gennem labyrinten hurtigere.
På venstre controller er der en tilsvarende knap, som kan trykkes på for at spilleren hopper. Det at hoppe har ingen funktion i spillet på nuværende tidspunkt andet end sjov for spilleren, men kunne være blevet brugt til at gøre en bane sværere længere henne i udviklingsprocessen. Gruppen har valgt at beholde muligheden for at hoppe, selvom det hverken hjælper eller giver modstand for spilleren, da der er en mærkbar glæde i at hoppe i spil.
Derudover bruges 'A' knappen, befindende på højre controller, til at starte og genstarte spillet. Der er ikke nogen pauseskærm eller menu, idet dette kunne bryde indlevelsen; i stedet må spilleren pause ud til Oculus hovedmenuen, hvis de ønsker en pause uden at risikere at blive fanget. For at gå til hovedmenuen trykkes der på Oculus-knappen der befinder sig i midten af både højre og venstre controller.

Brug af Kanban board

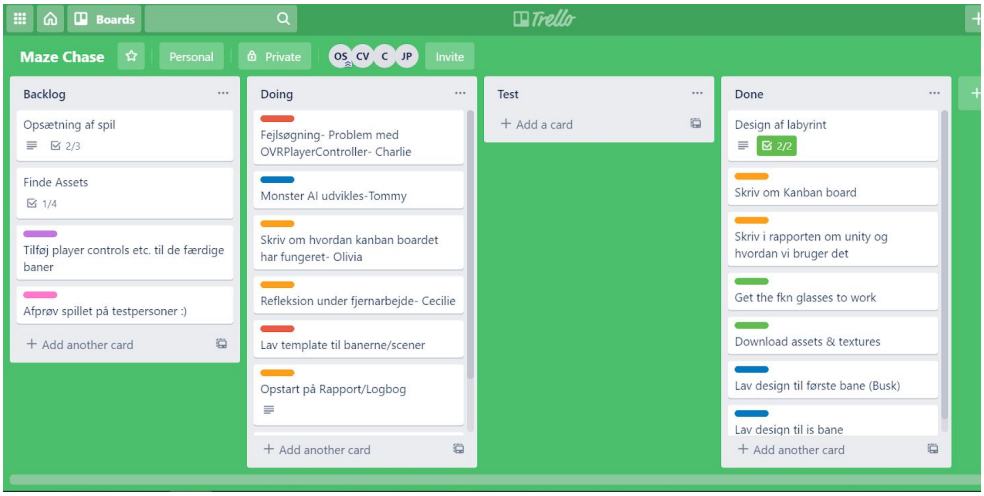
Hvad er et Kanban board?

Et Kanban board er et værktøj som bliver anvendt til at visualisere arbejdet, minimere work-in-progress, samt forbedre effektiviteten i ens projekt (Anderson et al., 2016). Ens board består af nogle kort med de opgaver som skal løses beskrevet på kortene. Disse kort sættes i kategorier i form af kolonner, hvor der står en overskrift. F.eks. kunne ens overskrifter på disse kolonner hedde *'Backlog', 'Doing', 'Test'* og *'Done'*. Dette skaber overblik over hvilke opgaver ens projektmedarbejdere er i gang med, hvilke er færdige og hvilke opgaver skal løses.

Man kan vælge at udarbejde ens Kanban board med post-it notes på en tavle eller boardet kan blive udarbejdet digitalt ved hjælp af sider som *GitHub, Trello* m.fl.

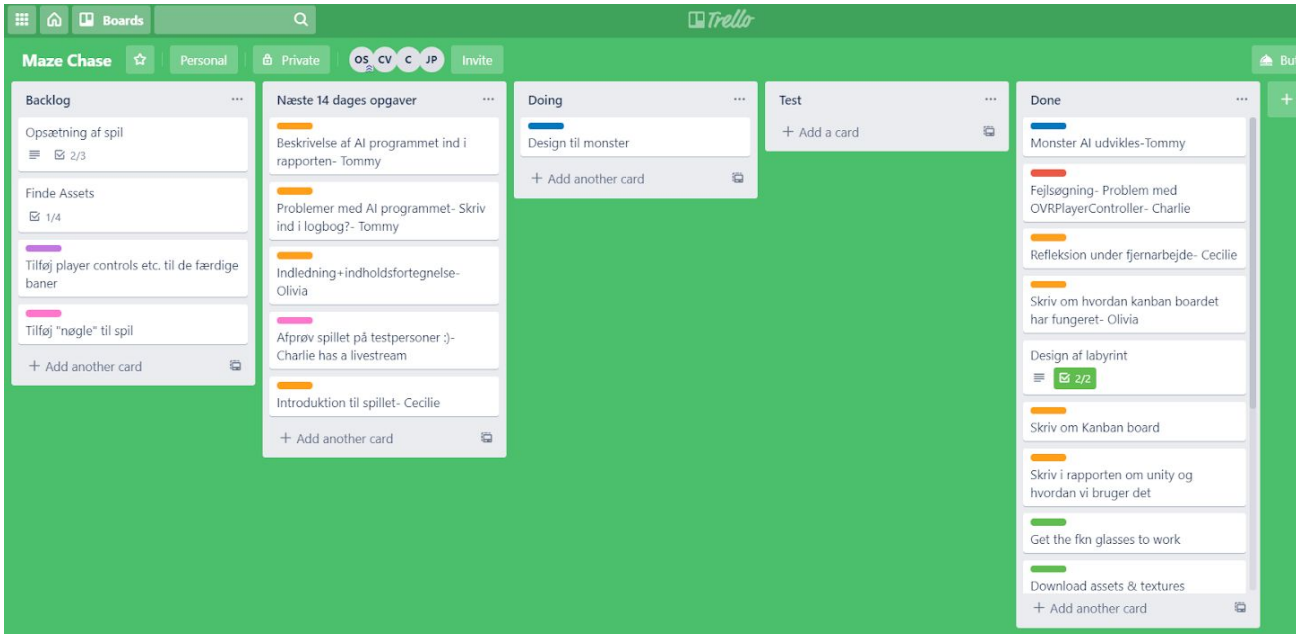
Hvordan har gruppen anvendt Kanban Board?

Gruppen har benyttet sig af hjemmesiden *Trello.com* til at opsætte vores Kanban Board. På ens *Trello board* kan man tilføje nye kort og kolonner. Derudover kan man give ens kort forskellige farvede etiketter som kan være med til at give bedre overblik over hvilke slags opgaver der høre til de forskellige kategorier.



Screenshot taget d. 8/4-20

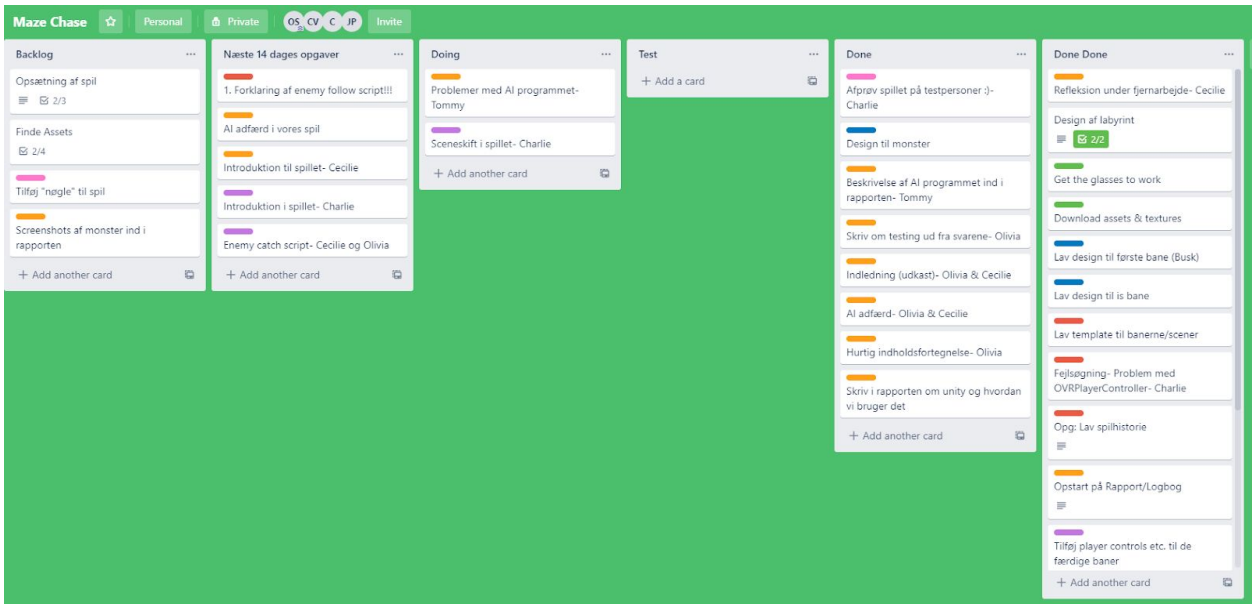
Billedet ovenfor viser *Trello boardet* udarbejdet af gruppen, hvor man kan se kolonner med overskrifter, samt opgaverne der skal løses, er i gang med at blive løst og dem som er færdiggjort. Ydermere er der også en kolonne dedikeret til testningen af vores spil.
Overskrifterne til kolonnerne har vi valgt at kalde *'Backlog', 'Doing', 'Test'* og *'Done'.*
Backlog-kolonnen indeholder de opgaver som skal løses, men som ingen arbejder på endnu. Hvis man derimod vælger at løse en af opgaverne i backlog, så rykkes kortet med opgaven hen i *Doing-kolonnen*.
Test-kolonnen er dedikeret til testningen af spillet. Her indsættes de opgaver som har noget med testing af spillet af gøre. Dette er for eksempel; test af spil på prøve personer etc.
Når gruppen så havde fuldført en opgave, rykkes kortet med opgaven hen i *Done-kolonnen*.



Screenshot taget d. 5/5-20

Billedet ovenfor viser vores kanban board senere i processen, hvor der er kommet en ekstra kolonne. Denne kolonne består af nogle opgaver som skulle løses indenfor en periode på 14 dage. Formålet med at have et tidsbegrænset periode er at opfordre til effektivitet i forhold til udarbejdelse af opgaverne.
Ydermere, har disse opgaver hver en specifik farve som indikerer hvilken slags opgave det er, f.eks. den orange farve betyder at opgaven har noget med projektskrivning at gøre, hvor at den blå, pink og lilla er opgaver indenfor udviklingen af spillet at gøre. Derudover tilføjes navne på de ansvarlige til den specifikke opgave, så der skabes et bedre overblik over fordelingen af arbejdsopgaverne, for de resterende gruppemedlemmer.

Sidst i processen blev der tilføjet en kolonne ekstra til Kanban boardet, kaldt *'Done Done'* som er den kolonne der er den endelige - altså her rykkes alle opgaverne hen når de er helt færdige og der er rette og læst korrektur, og alle gruppemedlemmer har godkendt.
Undervejs i skriveprocessen kan der opstå nye idéer og nyt materiale til rapporten og derfor kan det være en nødvendighed at trække en opgave tilbage til *'Doing'* kolonnen, da den skal redigeres, før den er endeligt færdig.
'Done Done' kolonnen har skabt et bedre overblik over hvad der stadig skal rettes og ændres, f.eks. efter testing af spillet på testpersoner, og hvilke opgaver der er endeligt færdige og ikke skal tilrettes eller på nogen måde ændres inden aflevering.



Screenshot taget d. 20/5-20

Hvad har gruppen fået ud af at bruge Kanban Board?

Kanban boardet har været med til at skabe en struktur, samt overblik over de opgaver som der opstod under projektet. Det har ydermere bidraget til en mere fair opgavefordeling blandt gruppemedlemmerne, idét at når et opgavekort oprettes er det muligt at skrive navnet på et gruppemedlem i opgavebeskrivelsen så alle i gruppen har styr på hvem der har ansvaret for hvilke opgaver. Ydermere har hjemmesiden, *Trello*, en let platform som også har gjort det lettere for gruppen at rykke rundt i de forskellige opgaver i kolonnerne og på denne måde været med til at visualisere vores arbejdsflow. En visuel repræsentation af vores flow har bidraget til effektivitet i projektarbejdet idét at vi kan se hvilke opgaver vi har fået fuldført, og hvilke der endnu ikke er færdige.

Derudover viste vores Kanban board at være en stor hjælp, da gruppen blev nødsaget til at benytte sig af fjernarbejde grundet Covid-19. Fjernarbejdet medførte at gruppen ikke var i stand til at kunne møde fysisk, men var nødsaget til at holde online møder, samt live-coding sessioner. I denne situation har vores kanban board vist sig at være en stor hjælp, da det var med til at give et overblik over hvilke opgaver der skulle løses, hvad der blev arbejdet på og

hvilke der allerede var blevet løst. Dette sikrede at vi ikke stødte ind i situationer, hvor flere gruppemedlemmer arbejdede på samme opgave uden at vide det.

Brug af logbog

Gennem hele processen har gruppen ført en logbog til at holde styr på hvilke problematikker der er opstået under arbejdsprocessen, og hvordan problematikkerne kunne løses, så spillet kunne ende med at fremstå bedst muligt.
Formålet med logbogen var at kunne skrive noter fra møder og seminarer ned, og bruge det til endeligt at udforme spillet til en sjov og god brugeroplevelse.
Midtvejsseminaret viste sig at være en stor hjælp for udformningen af spillet, da der blev fremlagt god og konstruktiv kritik af spillet, som det så ud på daværende tidspunkt - f.eks. blev der spurgt ind til monsterets funktion og flere idéer til hvordan gruppen kunne udforme monsteret så det fik den rigtige funktion i forhold til spilleren - nemlig at jage dem.
En stor del af kommentarerne og feedbacken er blevet benyttet under udarbejdelsen af projektet, dog er der også blevet sorteret få af dem fra, da det skulle begrænses så projektet endte ud med at stemme overens med gruppens vision.
Nedenfor vises et udsnit af feedback og kommentarer fra midtvejsseminaret (noterne er på engelsk, da mødet foregik på engelsk):

Dag 4 - D. 11/3-20:

Noter fra midtvejsseminaret:

- Predator character - how will it move around and how will it chase the character? (pathing for the predator)
- Maybe some kind of help? or a checkpoint? Maybe the player left their footsteps? (like pacman)
- What is the point with having a monster?
- Possibly some hiding spaces
- The predator is doing a patrol
- What about speed? Is the character faster than the predator?
- Sounds in the game?
- Why am I in the maze? What is the idea with the game?

- Different difficulties (as you progress, does it get harder to escape?)
- The predator behavior

Suggestions:

- Find people to test the game
- Ask some questions about improving the game
- Go for things that bring something to the game
- How can we reuse the code and bring value to the game
- Code explanation - description of the program
- How does unity work?
- Requirements for the game - bullet points
- Write about testing
- Theory about the monster
- let the predator have a complete blik over banerne

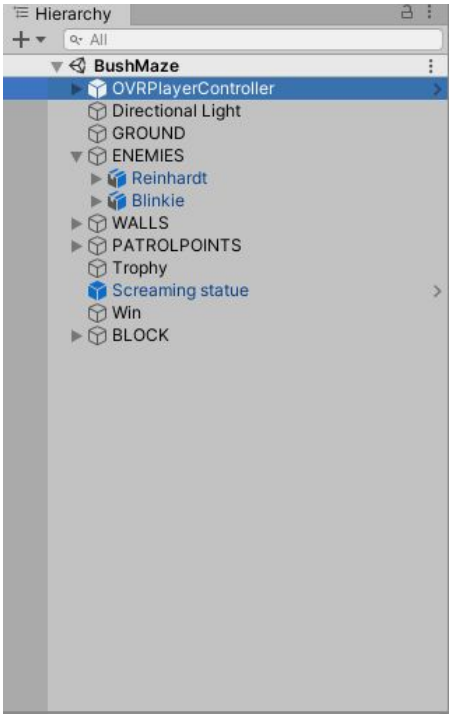
Logbogen har været behjælpelig under hele processen, men særligt under fjernarbejdet grundet Covid-19. Her har gruppen bedre kunne holde styr på hvem der har arbejdet på hvad, og hvilke informationer der har været til stor hjælp for projektet, fra vejleder, Martin Kirkegaard.

Teknikken bag

Unity

Unity er et program der ligger frit tilgængeligt for alle med en internetforbindelse. Dette program kan bruges til at lave 2D, 3D, VR og AR spil, uden at man behøver den største viden indenfor programmeringssprog, da mange eksisterende funktioner allerede eksisterer inde i programmet. Der er selvfølgelig stadig mulighed for at man kan skrive sin egen kode, for at lave et unikt program, og i disse tilfælde skal scriptet skrives i C# (Unity Documentation, 2019).

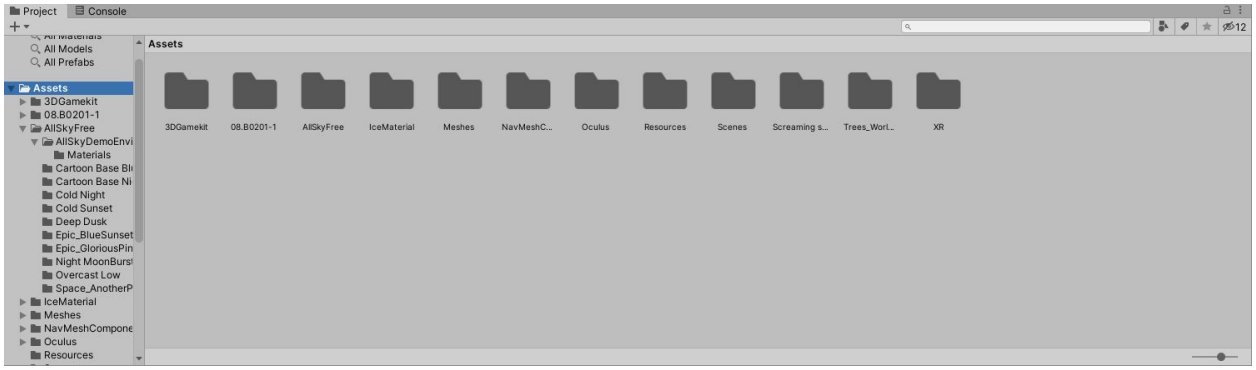
Unity er et visuelt 'programmeringssprog' - det skal ikke forstås som, at Unity i sig selv er et programmeringssprog, men i stedet et værktøj, der med visuelle cues 'skriver' koden, når man interagerer med programmet. I Unity har hvert vindue en funktion - Hierarki, projekt assets, inspector og scene er blandt de mest vigtige, idet det er her du laver dit program.



Hierarki for The Maze Chase, 16/5-2020.

I venstre side af Unity-vinduet findes hierarkiet, hvor de individuelle dele af din scene befinder sig. Hvert objekt kan enten stå alene eller lægges under et andet objekt; dette vil sige at Unity er objekt-orienteret, idet hvert basis-objekt kan genbruges flere steder, uden at skulle laves på ny.

Desuden kan et objekt A, der har objekt B lagt under sig i hierarkiet, bestemme hvad objekt B skal gøre, når objekt A interageres med i programmet. I ovenstående figur ses hierarkiet fra gruppens spil; *OVRPlayerController* er spilleren, og hvert objekt i hierarkiet er samlet i en gruppe for lettere at kunne overskue, hvilke objekter der bruges til hvad. Alle objekter kan navngives, hvilket også hjælper til at finde det specifikke objekt, der ønskes ændret.

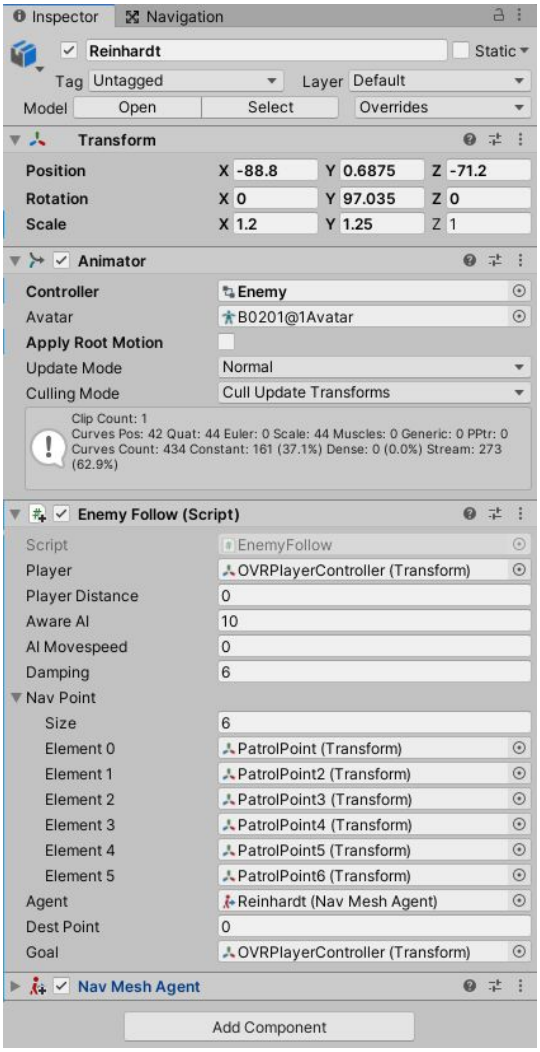


Projekt assets for The Maze Chase, 16/5-2020

I bunden af Unity-vinduet findes projekt assets, som er alle de objekter, der kan bruges i programmet. Når et nyt projekt startes, vil der oftest ikke være særligt mange assets i dette vindue, men nye assets kan enten hentes fra Unity's tilhørende asset store, hentes fra pålidelige hjemmesider, eller laves på egen hånd gennem andre programmer. Assets kan være alt fra 3D-modeller til færdige stykker kode, der kan kombineres med individuelle objekter til at lave en bevægelse, en funktion eller en effekt. Dette kan for eksempel være, at når der trykkes på en knap, bevæger et objekt sig, eller hvis et objekt rammer et andet objekt, kommer der en eksplosion.

I The Maze Chase findes både assets hentet fra asset store, og assets som gruppen selv har lavet. Blandt disse er blandt andet teksturerne på labyrintens vægge udarbejdet af gruppen, og scriptet der får fjenden til at patruljere og jagte spilleren er kodet med inspiration fra diverse YouTube-kilder (Octo Beard, 2018). For at bruge en asset i sit projekt, kan dette trækkes ind i hierarkiet, eller i tilfælde af scripts kan disse trækkes direkte til et objekt.

Dette kan ses på fjende-objektet, der med tilhørende script ser således ud:

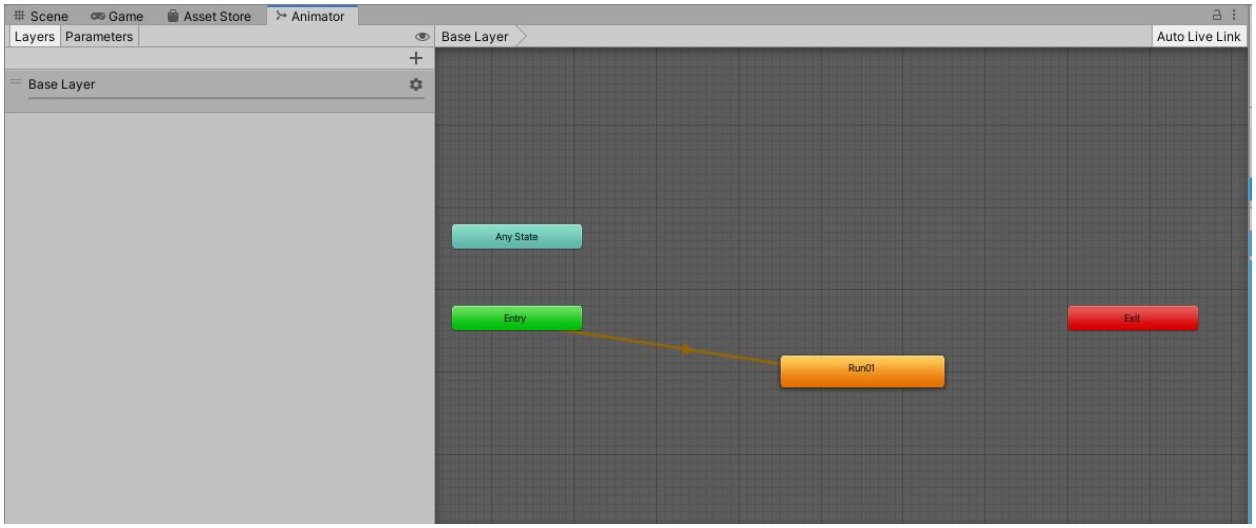


Fjende-objekt ‘Reinhardt’, 16/5-2020

En nærmere forklaring af hvordan Enemy Follow-scriptet fungerer vil følge i næste kapitel. Da scriptet er synligt i Inspector-vinduet, er det dog relevant at forklare dette først.

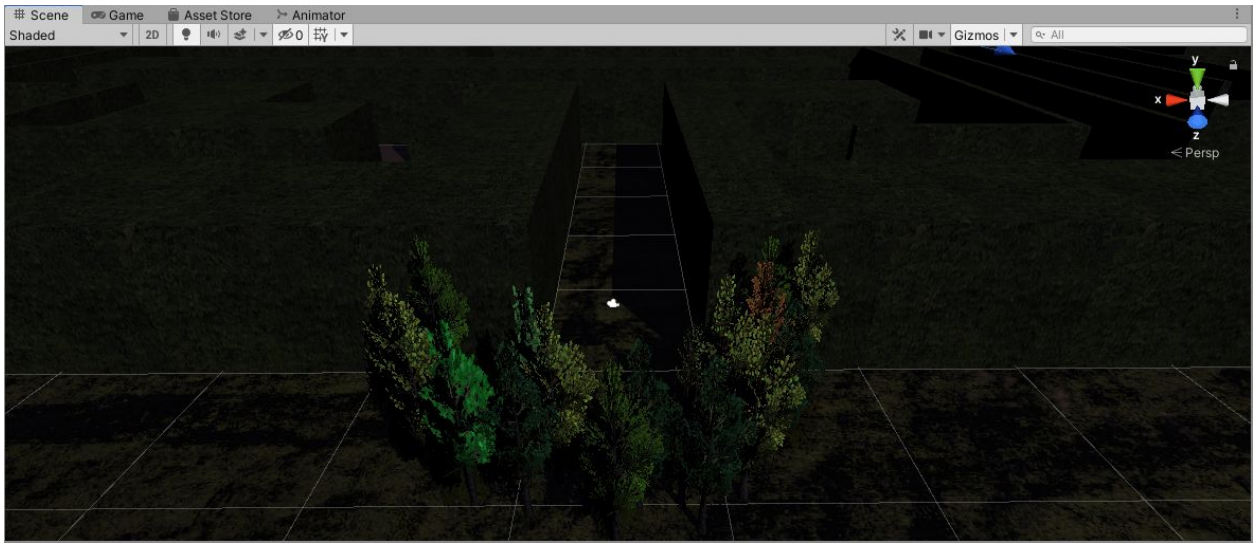
Inspector-vinduet findes i højre side af Unity, når et objekt er valgt i hierarkiet eller i scenen. Dette vindue viser alle de variabler, som har en påvirkning på objektet; dette kan være størrelse, position, hældning, tekstur, hastighed, effekt og mange flere; i dette vindue kan også tilføjes scripts til det dertilhørende objekt. Scripts er objekter i sig selv af skrevne stykker kode, som enten kan køre individuelt i hierarkiet, eller tilføjes til et objekt for kun at påvirke eller knytte sig til dette.

I ovenstående figur kan man desuden se ‘Animator’ komponenten. Dette er endnu en del af Unitys indbyggede kode, der kan bruges til at animere objekter. Animator-vinduet ser således ud:



Animator-vindue til fjende-objekt, 16/05-2020.

I The Maze Chase er fjende-objektet blevet illustreret med en model hentet fra Asset Store (PRESS START, Unity Asset Store 2020); denne model er pakket med to animationer, hvoraf gruppen valgte at bruge den ene: 'run01'. I Animator-vinduet kan der trækkes pile mellem hver bjælke, og derfra vælge hvilken animation der skal spille hvornår. Ønskes der kun at bruge én animation, behøves det ikke at loope eller trække den samme animation ind i vinduet flere gange, så længe der ikke trækkes en pil til 'Exit' bjælken.



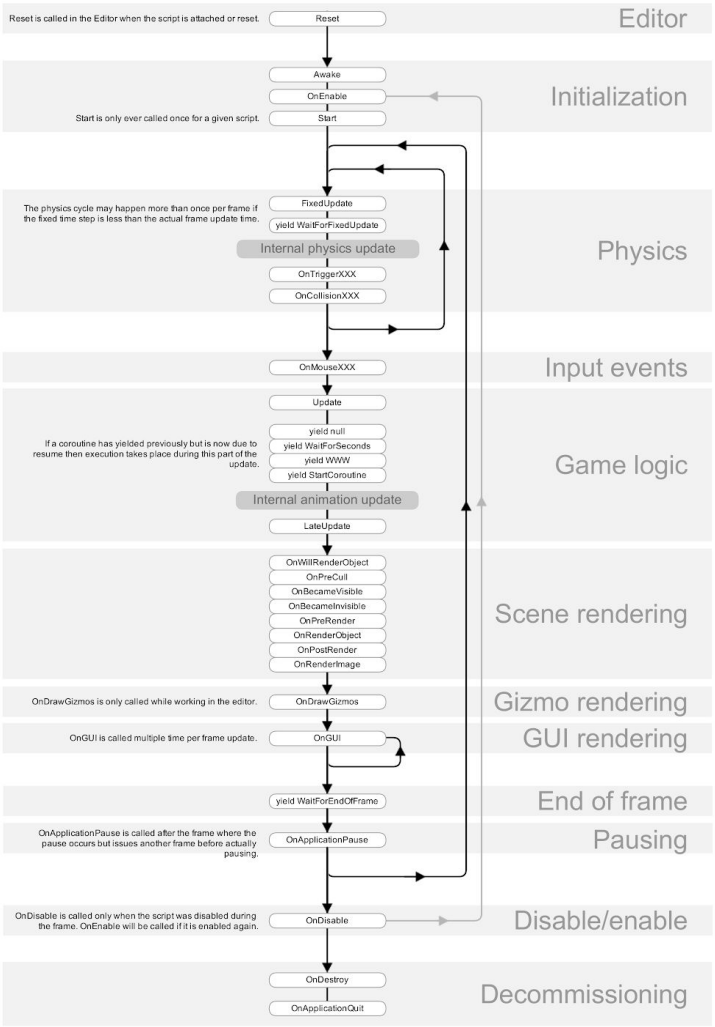
Scene fra The Maze Chase, 16/05-2020

Scene-vinduet findes i midten af Unity, og det er her, man kan se hvordan programmet udformer sig. I sig selv har scene-vinduet kun få funktioner: et 'kamera', der styrer hvad man kan se i selve vinduet, og nogle værktøjer til at flytte og dreje på objekter i scenen. Disse samme funktioner kan findes i inspector-vinduet, men forskellen ligger i om ændringen

foretages med bevægelse af musen versus indtastning af tal. I ovenstående figur kan ses indgangen til labyrinten hvori spillet foregår; desuden kan kameraet også ses, og lysretningen der er blevet valgt.

Monobehaviour

Monobehaviour er en base class, som alle C# scripts skal bruge, for at kunne fungere med Unity. MonoBehaviour indeholder en gruppe af methods der kan kaldes i egne scripts, som gør tilgængeligheden og kodningen af et program lettere for brugeren. Brugen af MonoBehaviour er nødvendig, og derfor en stor del af Unitys objekt-orienterede programmering, idet man kan skrive flere forskellige scripts, som alle arver fra hinanden, eller arver fra samme class og har hver sin funktion.



Flowchart for MonoBehaviour, Unity3D.com

I ovenstående figur ses et udsnit af de metoder, der kommer med *MonoBehaviour*, og i hvilken rækkefølge disse kaldes, og hvilke kategorier de falder under. Blandt andet ses metoderne *Start()* og *Update()* disse bruges til at instantiere en kode når scriptet kaldes, og derefter hver gang scriptet opdateres, hvilket gøres hvert frame. Der findes også *FixedUpdate()* og *LateUpdate()*, som kan bruges til at kalde updates hurtigere, langsommere eller efter spil logikken er blevet kørt, som for eksempel hvis et kamera skal flytte sin position efter et spil-objekt har flyttet sig.

En af de oftest brugte metoder fra *MonoBehaviour*, der bliver benyttet i The Maze Chase, er *OnTriggerEnter()* og *OnCollission Enter()*. *OnTriggerEnter()* bruges når to spil-objekter rammer

hinanden, og det ene spil-objekt har en *IsTrigger* hæftet på sig, hvilket gøres i Inspektor-vinduet. Desuden er det vigtigt at det ene eller begge spil-objekter har komponenten *RigidBody* hæftet, da programmet ellers ikke kan læse, om spil-objekterne rammer hinanden.
Det vigtigste programmøren skal huske, når denne benytter metoder fra *MonoBehaviour* i sine egne scripts, er at tjekke om alle faktorer for udførelsen af koden er blevet opfyldt. For eksempel vil *OnTriggerEnter()* ikke fungere, selvom metoden er kaldt korrekt i det skrevne script, hvis de nødvendige komponenter ikke er hæftet på de spil-objekter der skal benytte scriptet.

Oculus Quest

Oculus Quest

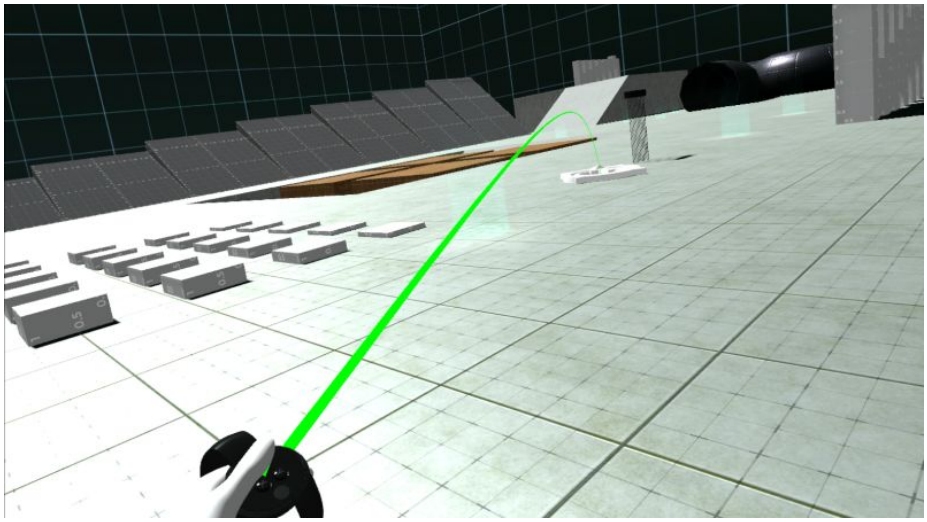
Oculus Quest er en udvidelse af Oculus-linjen, der er ejet af Facebook; Quest er udviklet til at være det første All-in-One headset fra Oculus, hvilket vil sige at der ikke behøves PC, telefon eller spillekonsol for at hente, installere eller køre VR spil, film og programmer på headsettet. Det kommer med to controllere og et ladekabel, og tager kun et par minutter at sætte op. Ved opstart vil headsettet bede brugeren om at lave et ‘guardian space’, som et det rum, man kan bevæge sig i, uden at headsettet giver en advarsel.
Oculus Quest kan derudover parres med en telefon for at aktivere *Oculus Developer mode*, som er nødvendigt for at man kan pushe spil fra Unity til Quest. Dette gøres ved at føje et kabel fra computer til headset, og gennem de rigtige indstillinger i Unity kan en app compiles og pushes til headsettet, der derefter vil køre det. Dette program kan desuden findes senere i Quests indbyggede program-bibliotek under fanen ‘unknown source’.

Oculus Integration

Oculus integration er en asset-pakke som downloades fra Unity Store. Denne asset er en stor hjælp at have i ens projekt for at kunne udvikle VR-spil til Oculus, idet den indeholder en generel hjælpepakke med assets og eksempler på Unity scener. Dette kan derfor bruges til udviklingen af VR-spil for både entusiaster og professionelle spiludviklere. En af de vigtigste dele af *Oculus Integration* for The Maze Chase er de prefabs der følger med; prefabs er præ-definerede spil-objekter, ofte med tilhørende, underkastede spil-objekter (i Unity kaldet 'child objects'), der også har specifikke scripts eller andre formål. Et eksempel på et vigtigt prefab for denne gruppe er *OVRPlayerController*, som både giver et script til at bevæge spilleren ved hjælp af Oculus Quests medfølgende controllere, men også har blandt andet *OVRCameraRig* som child object, hvilket gør det muligt at bruge *Oculus Quest* headsettet som kamera, og se alting i VR. Oculus Integration giver også en masse nye materialer, modeller og skyboxes ('himlen' i et 3D spil, der også giver ambient lys til en given scene), hvilket giver rig mulighed for at prøve forskellige ting af uden at importere en masse forskellige pakker.

Bevægelse af spilleren

Når det kommer til VR spil findes der to måder at manuelt bevæge sig på: Med en controller, eller ved teleport. Med en controller behøver spilleren ikke pege et sted hen for at flytte sig, og kan lettere flytte sig uden at se i den retning de vil bevæge sig. Snarere tværtimod kan spilleren se alle andre steder hen mens de flytter på sig, og derved få en større indlevelse i spillet, idet der ikke er en lysende cirkel på jorden, hvor de vil flytte sig hen som med teleportation.



Eksempel på teleport, developer.oculus.com, 29/05-2020

Ved teleport har spilleren større mulighed for at flytte sig lange distancer på kort tid, men til gengæld står denne stille mellem hvert 'hop'. Teleport kan desuden være meget desorienterende da denne form for bevægelse er instantiøs og netop flytter spilleren forholdsvis lange distancer uden at vise afstanden der bliver flyttet. Derudover er den lysende cirkel for landing, og den farvede bue for hvor man peger hen også med til at fjerne indlevelsen i spillet, da det er en meget klinisk måde at vise bevægelsesmulighederne.
Ud fra disse observationer har gruppen valgt at bruge controller-baseret bevægelse i The Maze Chase, idet vi ønsker en dybere indlevelse og mere realistisk bevægelse, således at spilleren lettere kan lade sig bilde ind, at de er i situationen og ikke blot spiller et spil.

AI adfærd

Artificial Intelligence, oftest forkortet til AI, kan blive anvendt på adskillige måder inden for spilverdenen, men AI teknikken er oftest brugt til udvikling af adfærden af non-playable characters, forkortet til NPC, i spillet. En NPC er en karakter i spillet som spilleren ikke kan styre men derimod er NPC'en kontrolleret af computeren. Denne karakter besidder en forudbestemt adfærd som kan deles ind i tre grupper: *fjendtlig, venlig* og *neutral*. Selve begrebet NPC bruges som regel til at betegne karakterer som enten er under den venlige eller neutrale kategori, men den fjendtlige adfærd er noget som er tydeligt i mange spil. Denne adfærd tilhører de fjendtlige karakterer som spilleren støder på i løbet af spillet. Disse karakterer udgøre de forskellige udfordringer som spilleren bliver udsat for og har derfor en stor rolle i spillet. I computerspil er NPC'ens adfærd i de fleste tilfælde scriptet og bliver aktiveret af specifikke handlinger eller dialoger forbundet til den unikke NPC. Dette betyder at NPC'en adfærd derfor er pre programmeret og ikke kan ændres i løbet af spillet, medmindre man vælger at omskrive koden. Dog med hjælpen af AI er det blevet mere fleksibelt at kontrollere, samt implementere NPC'en i spillet (Zhadan, 2018).

AI bruges til kontrollere NPC'ens adfærd, herunder alt fra simple patruljer til beslutningstagning. Der findes tre forskellige måde at opdele kontrollen af NPC: *Bevægelseskontrol, beslutningstagning* og *strateg*i. Den første, *bevægelseskontrol*, beskriver hvordan ens NPC generelt bevæger sig rundt i spillet, altså hvilke patruljer den foretager sig. Denne form for AI kontrol kan dog også indebærer simplere ting som at aktivere en animation fra ens NPC.

Beslutningstagning benyttes til at ens AI kan foretage beslutninger som typisk stammer fra en database af mulige handlinger. Den sidste form af AI kontrol, *strategi*, bliver anvendt til flere NPC's i en gruppe, hvor hver karakter i gruppen kan besidde sine egne beslutninger, men selve beslutningstagningen påvirkes af gruppe strategien.
I de fleste tilfælde bliver AI brugt i spil til at lave stifinding, da faste patruljer er lette at implementere inde i spillet. Af denne grund er stifinding et af de vigtigste komponenter af spiludvikling, idet at det er essentielt for et spilsystem at være i stand til at finde den meste effektive patrulje til ens NPC, så de kan bevæge sig fra et startpunkt til en destination, undgå forhindringer og have realistiske bevægelser (Zhadan, 2018).

AI adfærd i vores spil

I 'The Maze Chase' har gruppen valgt at udvikle en NPC, monster-AI med fjendtlig adfærd, som jager spilleren igennem labyrinterne. Til dette spil har gruppen gjort brug af stifinding til monsterets faste patrulje i labyrinterne. Dette indebærer også brugen af *Bevægelseskontrol,* til hvordan vores AI bevæger sig rundt i spillet, samt generelt dens adfærd.
Monsterets generelle adfærd i spillet er opsat således, at den kan gå, se og jage og er bygget på *NavMesh*. Den vil gå mellem dens forskellige patruljepunkter - usynlige punkter der dikterer den rute som monsteret skal gå på. Dens synsfelt er i fronten af den, og den vil registrere spilleren, hvis denne kommer inden for en hvis afstand af monsteret. Hvis dette sker vil monsteret begynde at at jage spilleren, hvilket vil fortsætte lige så længe som monsteret har spilleren inde for sit synsfelt. Hvis spilleren ikke længere kan ses af monsteret, vil det bevæge sig tilbage til sin patrulje.
Der forklares mere detaljeret om monsteret senere i rapporten, under afsnittet *Beskrivelse af programmet*.

Tekniske Udfordringer

Da gruppen først fik udleveret deres *Oculus Quest*, blev dette gjort i forbindelse med en åben workshop ført af Ebbe Vang på RUC, så alle grupper der arbejdede med VR eller Unity kunne få en fælles introduktion til disse. Under workshoppen blev gruppen gjort bekendt med Unitys funktion og arbejdsmetode, og sat ind i hvordan *Oculus Quest* kunne sættes til computer for at pushe app fra denne til headsettet. Her stødte gruppen imidlertid på vores første problem, da

Unity meldte fejl ved hvert forsøg på at pushe appen. Efter meget troubleshooting blev der gjort et sidste, desperat forsøg på at klare problemet: Et nyt kabel blev brugt til at forbinde Quest til computer.
Dette viste sig at løse problemet, da det nye kabel var i stand til at lede både nok strøm til at køre headsettet, og transmittere data fra enhed til anden.
Det næste problem der opstod var, at appen der blev pushet fra Unity til Quest ville blive låst i en sort loading screen, som efter op til ti minutter ville crashe til hovedmenuen i Quest. Med hjælp fra IT-vejleder Ebbe Vang fik gruppen kørt en *adb logcat *:e*, der resulterede i fejlmeddelsen *"vulkan: dequeueBuffer returned unrecognised buffer"*. Ved nærmere undersøgelser viste denne fejl at være grundet en manglende *XR Plugin Manager* i Unity, der straks blev installeret hvorefter Unity fejlfrit kunne pushe, og appen ville køre i Quest.

Problematikker under programmering

Under projektet er der opstået nogle problemer i udvikling af spillet, som vi mener er vigtige at dokumentere. Disse vil blive beskrevet og gennemgået, for at give et indblik i hvordan vi har løst disse problemer og hvad der var årsagen til disse.

Se gennem vægge

Under udviklingen af monsterets AI, opstod det problem, at monsteret registrerede spiller-karakteren igennem labyrintens vægge. Dette resulterede i, at monsteret ville se spilleren på den anden side af en væg, hvorefter den ville blive ved med at løbe ind i væggen, medmindre spilleren bevægede sig i en retning væk fra den givne væg.

```
//gameObject.GetComponent<NavMeshAgent>().SetDestination(ThePlayer.transform.position);
playerDistance = Vector3.Distance(player.position, transform.position);
```

Kodeudsnit 1 af EnemyFollow. 25/05-2020

Grunden til dette problem viste sig i scriptet til monsteret. I Kodeudsnit 1 ses et stykke kode som er blevet udkommenteret. Denne kode sætter monsterets destination til spillerens position, hvilket resulterede i, at monsteret ville se spilleren, selv igennem vægge, da den kunne registrere spilleren, så længe de var inde for monsterets *AwareAI radius* (Se Kodeudsnit 3).
Ved at fjerne dette kodestykke, har monsterets AI kun kendskab til det, der er inde for dens synsfelt, og begynder kun at jage, når den kan se spilleren.

Fastlåst på patrulje

Et andet problem der opstod under monsterets udvikling, var at den ikke gad at forlade sin rute for at jage spilleren. Dette resulterede i at monsteret kun var effektivt inde for dens patrulje, hvilket gjorde at den nemt kunne undgås og derfor gjorde monsteret ligegyldigt.
Mange forsøg på at ordne dette virkede ikke, da mange ændringer i koden enten gjorde at monsteret ophørte med at fungere. Derfor valgte vi at øge antallet af patruljepunkter, således at monsteret ville kunne finde rundt i hele labyrinten, og samtidig jage spilleren heri. Vi valgte senere at tilføje flere monster-karakterer. Dette gjorde at monstrene havde kortere patruljer, og derfor var hurtigere til at gennemfører disse, og samtidig gøre det sværere for spilleren.

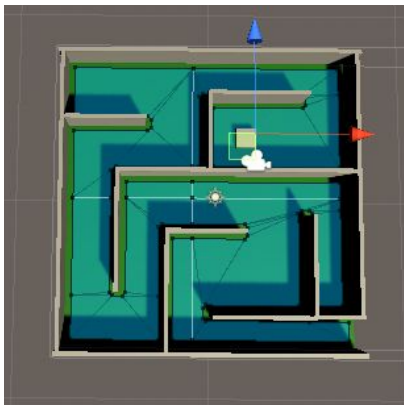
Beskrivelse af programmet

Kodning af Monster AI

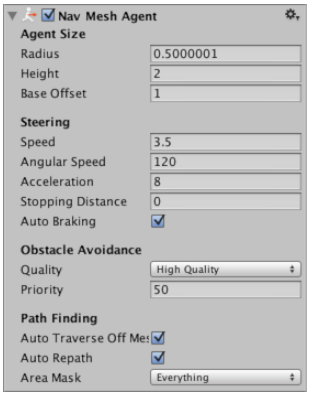
Kodningen af den AI som bruges i vores spil, er primært blevet gjort på baggrund af Unity's *NavMesh* og *NavMeshAgent* funktioner, samt et script skrevet i C#. *NavMeshAgent* er en række komponenter der hjælper forskellige karakterer i et spil, med at bevæge sig mod deres mål, og enten jage eller undgå andre karakterer. Takket være *NavMesh*, som bruges til at identificere forskellige forhindringer, er det muligt for karaktererne at differentiere mellem begrænsninger i spilverden, i form af vægge og andre objekter. Med *NavMeshAgent* kommer der også en API (*Application Programming Interface*), hvor det er muligt at ændre de forskellige parametre i forhold til at finde vej og rumlig opmærksomhed. Hvordan *NavMesh* og *NavMesh Agent* er blevet brugt vil blive beskrevet nedenfor, hvor der også vil blive gået i dybden med det script, som understøtter AI'en.

NavMesh & NavMeshAgent

NavMesh, eller *Navigation Mesh*, er et 3D objekt, der dækker alle gangbare områder i spilverdenen. Det kan bruges til at udføre rumlige opgaver, som stifindingstest og gangbarhedstest, ændre i indstillinger for stifinding for specifikke områdetyper og til at justere den globale opførsel af stifinding og undgåelse af andre karakterer. For at benytte sig af rumlige opgaver, skal man *"bake"* NavMesh for sin scene (Unity 2019, *Building a NavMesh*).



NavMesh baking er en proces der laver et *NavMesh* ud fra level geometrien. Processen opsamler *"render meshes"*, som er en opsamling af samtlige spilobjekter, og terræn af alle spillets objekter, der er markeret som *"Navigation Static"*, hvilket betyder at de objekter ikke vil ændre sig. Derefter behandles de til at oprette et navigationsnet der tilfører levellet dets gangbare overflader. Det endelige *NavMesh* vil derefter blive vist i scenen som et blåt overlay (Illustration NavMesh Overlay) på det underliggende levels geometri når 'Navigation' vinduet er åbent. Det vil derudover være muligt at finde en *NavMesh* fil under Assets mappen (Unity 2019, *Building a NavMesh*).



NavMeshAgent-komponenter hjælper med at oprette karakterer, som undgår eller jager hinanden, mens de bevæger sig mod deres mål. Agenterne analyserer spilverdenen ved hjælp af *NavMesh* og ved derfor hvordan de undgår hinanden såvel som andre forhindringer. Stifinding og rumlig analyse håndteres ved hjælp af scripting-API'en til *NavMeshAgent* (Unity 2019, *NavMeshAgent*).

Agenten defineres af en lodret cylinder, hvis størrelse er specificeret af radius og højde parametre. Cylinderen bevæger sig med et objekt eller karakter, men forbliver lodret, selv hvis selve objektet roterer. Formen på cylinderen bruges til at opfange og reagere på sammenstød mellem andre agenter og forhindringer. Højden og radiusen på cylinderen er specificeret to steder, i *NavMesh bake* indstillingerne og i parametrene for de individuelle agenter (Unity 2019, *NavMeshAgent*).

NavMesh og *NavMeshAgent* er vi vores projekt blevet brugt til at give monsteret bevidsthed om dens omgivelser, således at den kan navigere i labyrinten, uden at gå igennem vægge og med mulighed for at navigere.

Script til EnemyFollow

I dette afsnit vil koden til monsteret blive gennemgået. Der vil som udgangspunkt blive kigget på de vigtigste dele af koden.
Det første koden gør, er at inkorporere *NavMeshAgent* (Se Kodeudsnit 2), beskrevet tidligere i rapporten, således at dens egenskaber kan bruges i sammenhæng med koden.

```
void Start()
{
    //Incorporates the NavMeshAgent
    UnityEngine.AI.NavMeshAgent agent = GetComponent<UnityEngine.AI.NavMeshAgent>();
    agent.destination = goal.position;

    agent.autoBraking = false;
}
```

Kodeudsnit 2 af EnemyFollow. 25/05-2020

Ved at benytte *NavMeshAgent* er det muligt at styre en del parametre for monsterets AI, såsom hastighed, uden at skulle programmere det manuelt.

En vigtig del af monsterets AI er dens evne til at navigere i labyrinten. Dette bliver gjort ved at bruge navigationspunkter, som monsteret vil finde vej til i en specifik rækkefølge.

```
void GotoNextPoint()
{
    if (navPoint.Length == 0)
        return;
    agent.destination = navPoint[destPoint].position;
    destPoint = (destPoint + 1) % navPoint.Length;
}
```

Kodeudsnit 3 af EnemyFollow. 25/05-2020

For at sikre at monsteret går videre til de næste navigationspunkter, har vi gjort brug af kodestykket nederst i Kodeudsnit 3, som fortæller monsteret, at når det er ankommet til et navigationspunkt, for eksempel punkt 1, vil den automatisk begynde at bevæge sig mod punkt 2. Takket være opsætningen, er det muligt at tilføje yderligere navigationspunkter uden at skulle ændre i koden.

```
if (playerDistance < awareAI)
{
    if (playerDistance < 10f)
        Chase();
    else GotoNextPoint();
}

{
    if (agent.remainingDistance < 0.5f)
        GotoNextPoint();
}
```

Kodeudsnit 4 af EnemyFollow 25/05-2020

For at få monstret til være en udfordring, skal den kunne jage spilleren. Dette gøres ved at give den en *Chase* funktion (Se Kodeudsnit 4), som kaldes, hvis spilleren kommer indenfor en bestemt afstand af monsteret (Kodeudsnit 4). *Chase* funktionen er i stand til at holde fast i spillerens position, så længe denne er indenfor monsterets synsfelt, og det kan derfor jage spilleren, indtil det ikke længere kan se spilleren. Takket være *NavMeshAgent* er dette kun gældende for hvad monsteret kan se, således at spilleren ikke kan ses igennem vægge. Hvis spilleren lykkedes med at undslippe monsteret, vil det returnere til sin planlagte rute, og gå hen til det navigationspunkt, der var det næste på listen over sin rute.

```
void Chase ()
{
    transform.Translate(Vector3.forward * AIMovespeed * Time.deltaTime);
    AIMovespeed = 6;
}
```

Kodeudsnit 5 af EnemyFollow. 25/05-2020

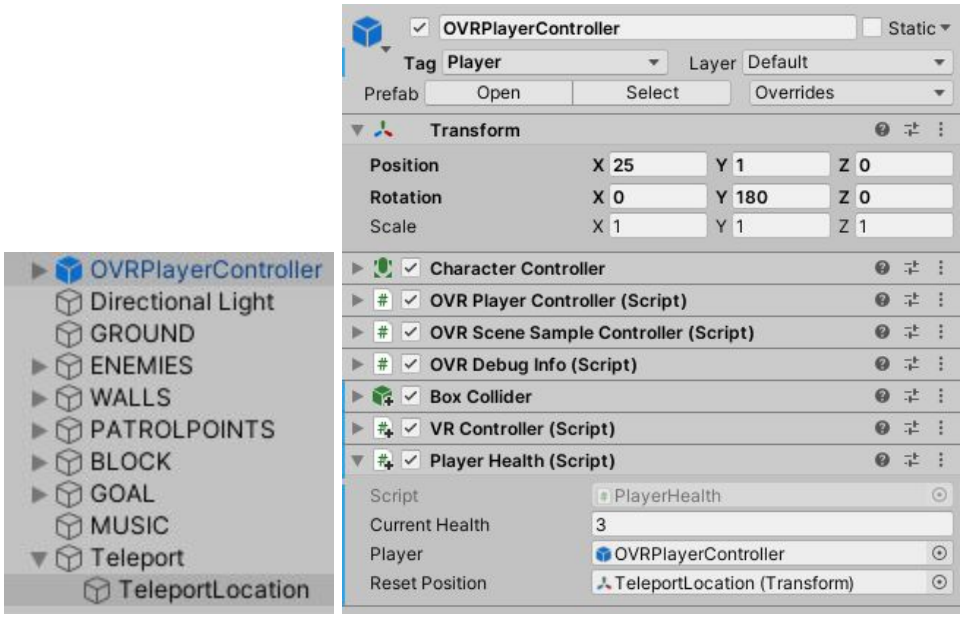
Script til PlayerHealth

```
public class PlayerHealth : MonoBehaviour
{
    public int CurrentHealth = 3; //Player starts out with 3 lifepoints
    public GameObject Player;
    public Transform ResetPosition;

    void Start(){

    }

    void OnCollisionEnter(Collision col) //OnCollisionEnter is a method used for when two colliders/rigidbodies touch each other
    {
        if (col.gameObject.tag == "Enemy") //Player collides with the enemy
        {
            CurrentHealth -= 1;  // And loses 1 lifepoint

            if (CurrentHealth <= 0) //If CurrentHealth is zero (or less) the player is dead
            {
                Death(); //Calling the Death class down below
            }
            //Player is reloaded on a new position in the level when losing a lifepoint
            Player.transform.localPosition = ResetPosition.transform.position;
        }

    }
```

Kodeudsnit 6 af PlayerHealth. 26/05-2020

Ovenstående kode danner rammen for hele interaktionen mellem spilleren og monster AI'en. Spillerens lifepoints sættes til at starte med 3, og der defineres et spil-objekt og en transform lokation til senere brug, hvorefter der følger en metode som definerer hvad der sker ved en kollision mellem spilleren og monsteret. *OnCollissionEnter()* er en metode der bruges til at registrerer når to rigidbodies støder på hinanden.

Der startes et *if-statement*, hvori der beskrives hvordan programmet skal agerer, hvis spillere kolliderer med et af monstrene. Ved hjælp af *col.gameObject.tag* ved Unity at dette if-statement kun skal køre, hvis det objekt spilleren støder ind i har fået tagget 'Enemy' i inspektor-vinduet. Ved kollision fratrækkes et lifepoint fra spilleren og de genstarter på en ny position i labyrinten. Dette gøres ved hjælp af *transform.localPosition = ResetPosition.transform.position. transform.localPosition* gør at spillerens lokation bliver transporteret til et specifikt startpunkt, som i dette tilfælde er *ResetPosition*.



TeleportLocation i hierarki og inspektor-vindue, 26/05-2020

Inde i inspektor-vinduet for vores Player objekt kan det ses, at der bruges et spil-objekt til at definere hvor spilleren skal flyttes hen til. Dette gøres i stedet for at bruge en Vector3, da der i stedet blot kan flyttes på et spil-objekt i Unity hvis der ønskes et nyt landingssted, i stedet for at manuelt skulle taste tallene ind i scriptet.

Inde i *if-statementet* findes der desuden endnu et *if-statement*, der determinerer hvad der skal ske hvis spilleren mister alle sine lifepoints. Hvis spilleren kommer ned på 0 lifepoints kaldes *Death()* som defineres i skærmbilledet nedenfor.

```
void Death() //Making a class that defines what "death" is
{
    SceneManager.LoadScene(5); //Loads the GameOver Scene. In our case it said "5"
}
```

Kodeudsnit 7 af PlayerHealth. 26/05-2020

I denne metode, genindlæser koden det der kaldes for scene 5, hvilket er den scene der dukker op som 'Game Over' når spilleren er død.

Script til Løb og Hop

I scriptet *VRController* udvides der på *OVRPlayerController* scriptet, som fulgte med *Oculus Integration*. Dette gøres for at give spilleren mulighed for at hoppe, hvis dette ønskes, idet *Jump()* funktionen i *OVRPlayerScript* ikke kaldes til en knap på controlleren.

Desuden udvides scriptet også med en funktion, der gør det muligt at ændre spillerens hastighed i inspektor-vinduet, og gøre det muligt at lave en sprint-funktion når der holdes en knap nede.

```
[RequireComponent(typeof(OVRPlayerController))]
public class VRController : MonoBehaviour
{
    private OVRPlayerController controller;

    [SerializeField]
    private float moveSpeedMultiplier = 3.0f;

    [SerializeField]
    private bool allowDoubleXSpeed = false;
```

Kodeudsnit 8 af VRController. 27/05-2020

Ovenstående kode-udsnit klargør, at det er nødvendigt at have en komponent af typen *'OVRPlayerController'*, for at dette script kan køre.
Derudover laves to fields af typen float og bool, som gør det muligt at ændre værdierne af disse i inspektor-vinduet.

```
void Start()
{
    controller = GetComponent<OVRPlayerController>();
    controller.SetMoveScaleMultiplier(moveSpeedMultiplier);
}

void Update()
{
    if(OVRInput.GetDown(OVRInput.Button.PrimaryIndexTrigger))
    {
        controller.Jump();
    }

    if(OVRInput.Get(OVRInput.Button.SecondaryIndexTrigger) && allowDoubleXSpeed)
    {
        controller.SetMoveScaleMultiplier(moveSpeedMultiplier * 2.0f);
    }
    else
    {
        controller.SetMoveScaleMultiplier(moveSpeedMultiplier);
    }
}
```

Kodeudsnit 9 af VRController. 27/05-2020

Når dette script starter, vil det først sætte en variabel *'controller'* af typen *OVRPlayerController* til at finde komponenten *OVRPlayerController*. Herefter sættes hastigheden for spilleren.
I *Update()* kan det ses, at der kontinuerligt vil tjekkes om spilleren trykker på den primære eller sekundære frontale trigger, henholdsvis venstre og højre hånd. Hvis der trykkes på den primære frontal-trigger, vil spilleren hoppe én gang. *GetDown()* er en metode, der tjekker hvorvidt en knap bliver trykket på, men ikke om den holdes nede. Dette betyder, at hvis spilleren

holder knappen nede, vil de kun hoppe én gang, hvor de kun kan hoppe flere gange hvis de trykker på knappen igen.
Derimod bruges *Get()* til at tjekke status på en given knap hele tiden; hvis den sekundære frontal-trigger holdes i bund, vil spilleren blive ved med at sprinte.

Script til Baneskift og (Gen)Start af spillet

I The Maze Chase bruges to forskellige scripts til at ændre, hvilken scene spilleren befinder sig i; *ChangeScene* og *ReStartGame*. *ChangeScene* bruges med en trigger til at skifte level, når spilleren når i mål i labyrinten.
For begge scripts er det vigtigt at bruge *UnityEngine.SceneManagement;*, da scriptet ellers ikke har nogen viden om, hvad metoderne omhandlende scener gør.

```
using UnityEngine;
using UnityEngine.SceneManagement;
```

using UnityEngine.SceneManagent;, 27/05-2020

```
public class ChangeScene : MonoBehaviour
{

    public float chosenScene = 0.0f;

    void OnTriggerEnter(Collider other){

        SceneManager.LoadScene((int)chosenScene);
    }
}
```

Kodeudsnit 10 af VRController. 27/05-2020

Der bruges en float kaldet *chosenScene* til at vælge hvilken scene spilleren skal skifte til, som samtidig er mulig at ændre i inspektor-vinduet i Unity. Dette gør det muligt at genbruge koden til at loade flere forskellige scener, uden at skulle hardcode levelnummer eller navn ind i scriptet.
Siden *chosenScene* er en float, men scene-numre kun kan være integers, er det vigtigt at caste *chosenScene* til integer i *SceneManager.LoadScene()*, da denne kode ellers ikke vil kunne køre.
OnTriggerEnter() bruges som beskrevet i *MonoBehaviour* til at tjekke om to spil-objekter kolliderer, hvormed det også er vigtigt at begge spil-objekter har en collider, og det ene objekts collider er sat til at være en trigger.

ReStartGame fungerer meget på samme måde, idet det også er et script til at skifte scene. Hvor *ReStartGame* har en betydelig forskel er dog i den måde som sceneskiftet kaldes;

```
public class ReStartGame : MonoBehaviour
{
    void Start()  {}

    void Update()
    {
        if (SceneManager.GetActiveScene().name == "OpenGame"){
            if (OVRInput.GetDown(OVRInput.Button.One))
            {
                //Start the game
                SceneManager.LoadScene(1);
            }
        }
        if (SceneManager.GetActiveScene().name == "GameOver" ){
            if (OVRInput.GetDown(OVRInput.Button.One))
            {
                //Restart the game
                SceneManager.LoadScene(0);
            }
        }
    }
}
```

Kodeudsnit 11 af ReStartGame, 27/05-2020

Med et *if-statement* tjekkes om den aktive scene er enten *OpenGame* eller *GameOver*, og hvis en af disse er sande, vil der blive tjekket om spilleren trykker på en knap og starter eller genstarter spillet. I koden er denne knap '*OVRInput.Button.One'*, hvilket svarer til 'A' på spillerens controller. *SceneManager.LoadScene()* er desuden hardcodet for begge disse instanser, idet det er vigtigt at der ikke ændres på hvilken scene spilleren skal føres til, da dette kan ødelægge oplevelsen af spillet.

Testing

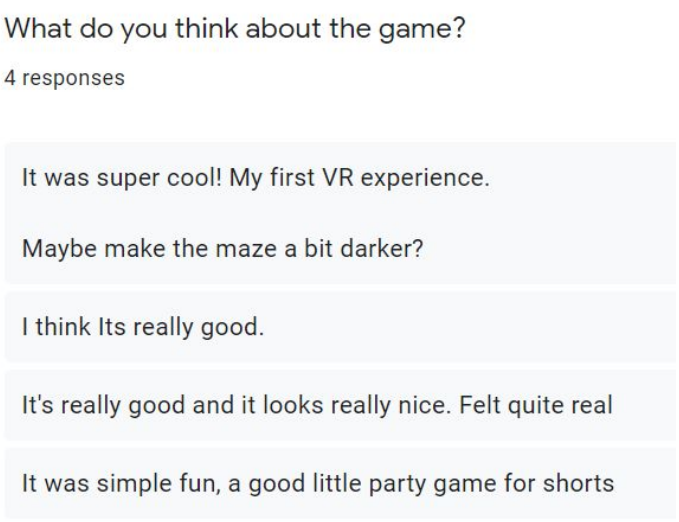
Den 12/5-20 testede vi for første gang vores spil på fire testpersoner som var i aldersgruppen 18-25+ år. Testen foregik ved at testpersonerne fik en introduktion til spillet og dets koncept. Dernæst blev spillet sat op og hver enkelt testperson blev bedt om at gennemføre spillet én ad gangen efter at de havde fået en forklaring af controls, men ellers fik de lov til at gøre hvad de havde lyst til i labyrinten, så længe at de nåede i mål.
Tre ud af fire valgte at prøve spillet stående, mens en person valgte at afprøve spillet siddende. Ydermere, brugte to af testpersonerne den højre controller til at dreje rundt i spillet, mens at de resterende to valgte at benytte sig af VR-headsettet til at vende sig rundt.

Gruppen havde før testen opstillet et spørgeskema, som testpersonerne skulle besvare efter afprøvningen. En af testpersonerne er ikke dansktalende og derfor har gruppen valgt at udarbejde spørgeskemaet på engelsk. Spørgeskemaet bestod af følgende spørgsmål:

1. How old are you?
2. What do you think about the game?
3. How was the movement in the game?
4. Was the goal of the game easy to understand?
5. Elaborate if needed on above question.
6. How do you feel about the enemy? 1 is worst, 5 is best.
7. Elaborate if desired!
8. Was it hard to find your way in the maze? Please elaborate.
9. What would you like to see added to the game?
10. Other comments? Critique, errors, anything goes.

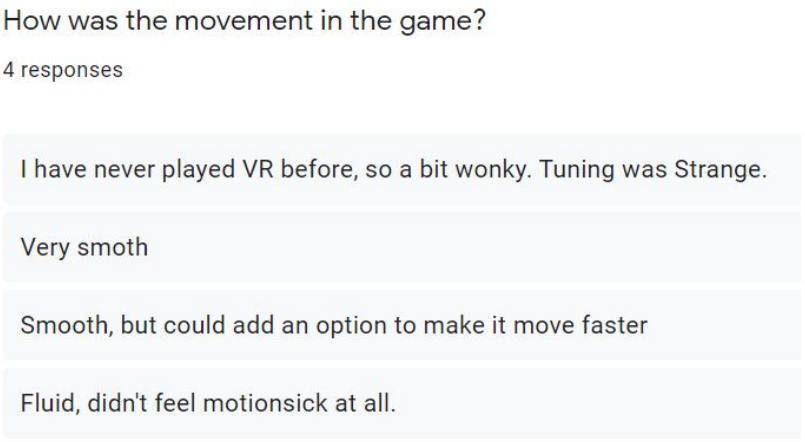
Resultater fra første test

Ud fra resultaterne fra spørgeskemaet kan vi se at spillet generelt har fået positiv feedback, hvor testpersonerne overordnet har haft en god oplevelse, som går fra at det var en af testpersonernes første gang de prøvede VR, til konstruktiv feedback som giver gruppen noget at arbejde med for at give den bedst mulige spiloplevelse for slutbrugeren.



Svar fra spørgeskema, første spørgsmål, 12/5-2020.

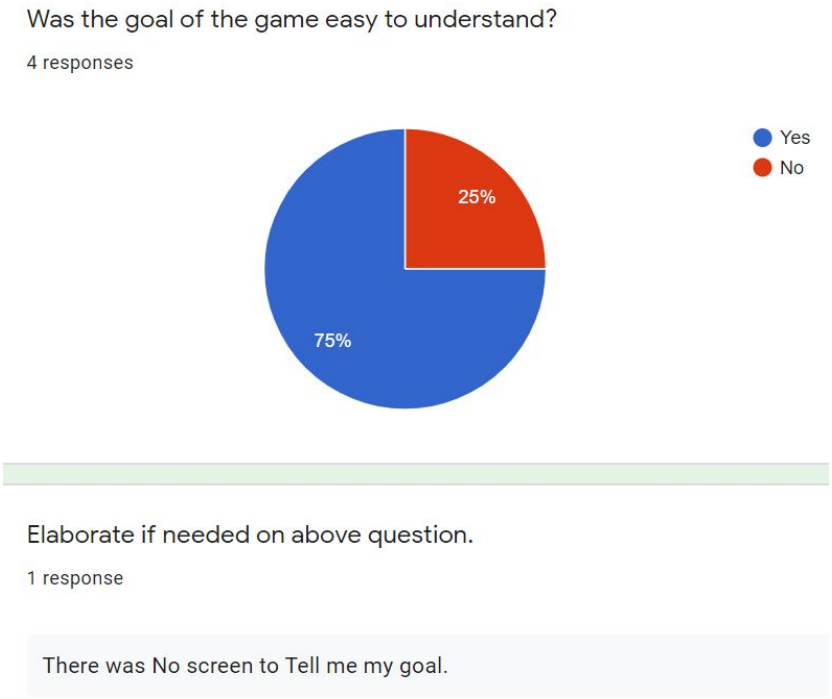
Testpersonerne blev også spurgt om hvad de synes om bevægelsen i spillet, som herunder indebærer det at få spilleren til at bevæge sig rundt i labyrinten ved hjælp af controllerne, men også hvordan det føltes for dem når de anvendte VR-headsettet til at bevæge blikket rundt. Billedet nedenfor viser deres svar:



Svar fra spørgeskema, andet spørgsmål, 12/5-2020.

Ud fra disse svar kan man se at testpersonerne har delte meninger i forhold til bevægelsen i spillet. Nogle synes at det var meget jævnt, og at de ikke oplevede cybersickness, hvilket er en tilstand med køresyge-lignede symptomer, som ofte opstår når man bruger et VR-headset. En anden testperson, som ikke havde prøvet VR før, synes at det var mærkeligt at skulle dreje sig fysisk for at dreje sig i spillet, mens et andet svar pointerede at det kunne være en god idé hvis

man som spiller kunne bevæge sig hurtigere rundt i labyrinten og dermed flygte hurtigere fra fjenderne.



Svar fra spørgeskema, tredje og fjerde spørgsmål, 12/5-2020.

Gruppen var også interesseret i at finde ud af om formålet med spillet var klart for brugerne. Billedet ovenfor viser at målet for spillet primært var klart, dog var det stadigvæk uklart for nogle af testpersonerne idet, at der på daværende tidspunkt ikke var en startskærm som gav en instruktion til spillet og en forklaring på hvad målet er. Dette er noget som gruppen senere vil implementerer så der ikke vil opstå tvivl for brugerne når de får VR-headsettet på og skal starte spillet.

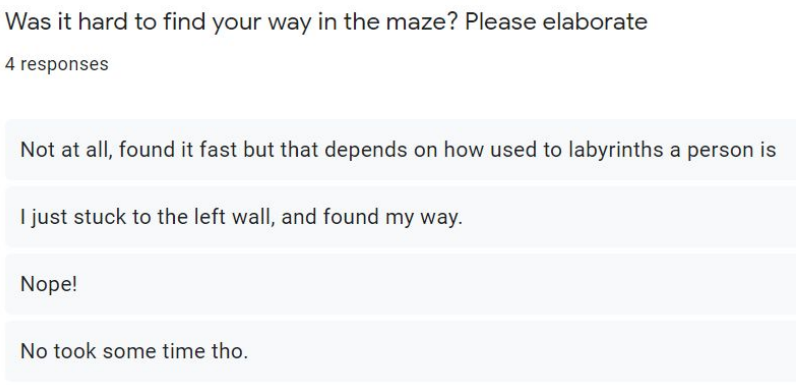
Ydermere var det også interessant for gruppen at høre hvad testpersonerne synes om AI monstrene i spillet idet disse fjender er en stor del af spillet, ved at være udfordringen for spilleren.

Billede af monsteret fra spillet

Svar fra spørgeskema, sjette og syvende spørgsmål, 12/5-2020.

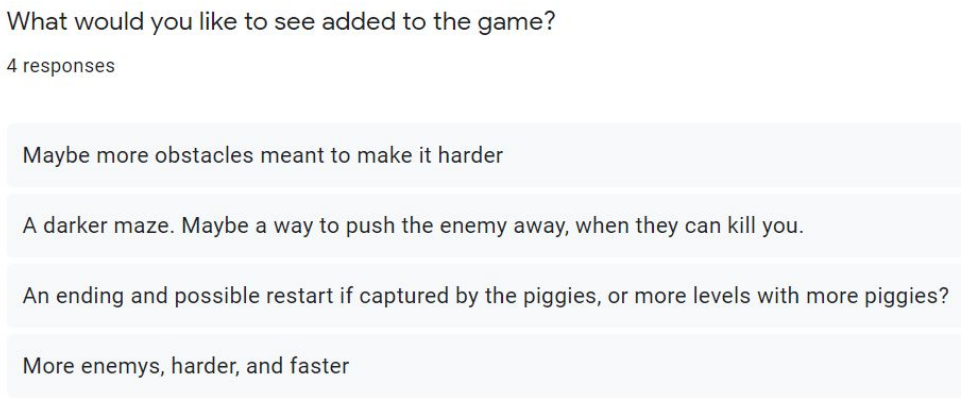
Ud fra disse svar kan man se at monstrene generelt fik positiv feedback, hvor at de blev betragtet som 'nuttede' og som nogle sjove fjender at støde på imens man prøver at finde sin vej rundt i labyrinten. Dog pointerer en af testpersonerne at disse fjender ikke var så svære at løbe fra, hvilket ikke er optimalt når intentionen med fjenderne er at det skal være en udfordring for spillerne at blive jagtet og forsøge at stikke af, uden at blive spist af monstrene.

Et andet vigtigt spørgsmål som gruppen ønskede svar på, var om det var svært at finde rundt i labyrinten da dette også er en essentiel del af spillet. Testpersonerne svarede dette:



Svar fra spørgeskema, niende spørgsmål, 12/5-2020.

Dette viser at generelt var det ikke svært at finde rundt i labyrinten - hvor det tog noget tid for en af testpersonerne mens de andre hurtig kunne finde rundt. Dette er både godt og skidt. Det er positivt at spillerne ikke sidder fast og derfor mister motivationen for at gennemføre spillet, men det er negativt hvis udfordringen i at finde vej ikke er stor nok og derved gør spillet mindre underholdende og udfordrende for spilleren.



Svar fra spørgeskema, sidste spørgsmål, 12/5-2020.

Billedet ovenfor viser hvad testpersonerne forslag til hvad der kunne forbedres ved spillet, hvilket opsummeret er at de ønsker flere udfordringer implementeret i labyrinten, samt slutning på spillet, mulig genstart hvis spilleren bliver fanget af monstrene og mulighed for at skubbe fjenderne væk når de prøver at fange spilleren.

Opsummering af første test

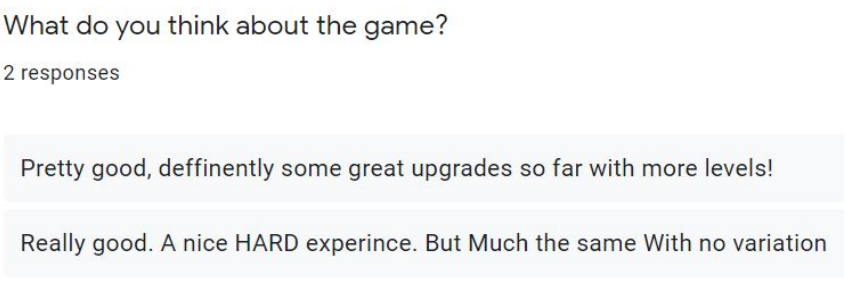
Testpersonerne har generelt haft en positiv oplevelse med at afprøve spillet og har krediteret med konstruktiv feedback om hvordan spillet kunne forbedres - alt fra et ønske om at gøre spillet mørkere, mulighed for at kunne løbe hurtigere og flere enemies ind i labyrinten for at gøre udfordringen større.
Feedbacken fra spørgeskemaet er alt sammen noget, der kan bruges til at forbedre spillet og gøre det endelige produkt, til en bedre oplevelse for slutbrugerne.

Resultater fra anden test

Efter feedback fra den første test, ændrede gruppen i nogle ting i spillet så det stemte overens med gruppens ønsker i forhold til spillet og projektet, og testpersonernes ønsker om forbedringer i spillet. Der blev tilføjet 'flere' labyrinter, som en variation af den oprindelige hvor målet var flyttet til en ny position og væggene er anderledes placeret. Herudover blev der tilføjet flere monstre så der kommer flere til, for hver gang et level er gennemført og spilleren sendes videre til næste level, for at intensiverer sværhedsgraden af spillet.
Ydermere er der blevet tilføjet uhyggelige lydeffekter og baggrundsmusik til spillet, for at tilføre et ekstra skræmmende element til oplevelsen.
Der blev udarbejdet et nyt spørgeskema til anden test, for at få feedback på spillet efter de mange ændringer der blev foretaget siden sidste test - og eventuelle nye idéer og kommentarer. Efter anden test blev testpersonerne stillet følgende spørgsmål:

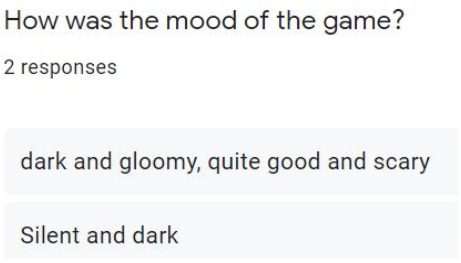
1. What do you think about the game?
2. How do you like the changes of the game from last time?
3. What did you think of the introduction screen?
4. How was the mood of the game?
5. What did you think of the level change?
6. What did you think of the change in difficulty?
7. What would you like to see added to the game?
8. Other comments? Critique, errors, anything goes.

Til anden test deltog to personer i testningen af spillet, da det kun var muligt for to af de oprindeligt fire testpersoner at deltage anden omgang.
Begge testpersoner udtrykte stor begejstring for de foretagede ændringer i spillet og spillet som helhed, og var glade for de tilføjede levels og flere fjender der jagtede dem gennem labyrinterne - dog synes der stadig at være en smule utilfredshed med sværhedsgraden af spillet.

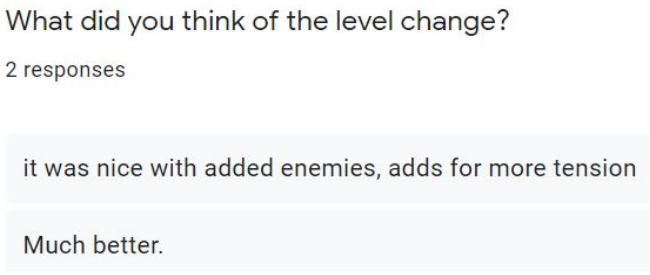


Svar fra andet spørgeskema, første spørgsmål, 27/5-2020

Første test viste at testpersonerne havde et ønske om at stemningen i spillet skulle være mørkere og mere uhyggelig, og at sværhedsgraden skulle øges gradvist for hvert gennemført level. Af denne grund tilføjede gruppen de tidligere nævnte lydeffekter, flere monstre og belysningen blev gjort mørkere. De foretagede ændringer viste sig at give god respons fra testpersonerne, som synes at spillet forekommer mere uhyggeligt end tidligere - hvilket kommer til udtryk i svarene fra spørgsmål 4 og 5 i spørgeskemaet.



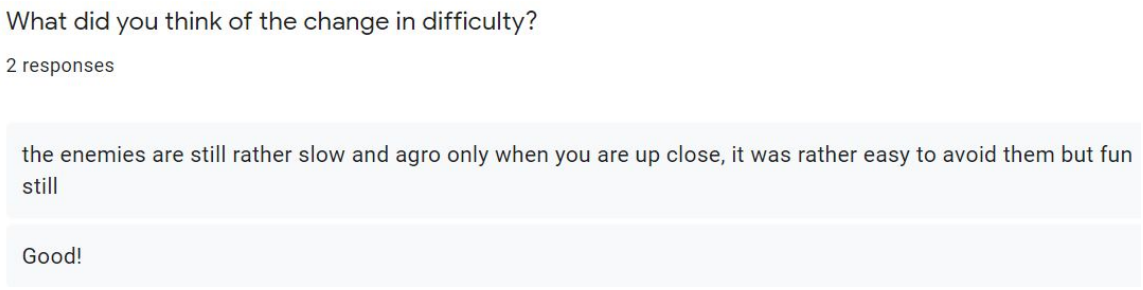
Svar fra andet spørgeskema, fjerde spørgsmål, 27/5-2020



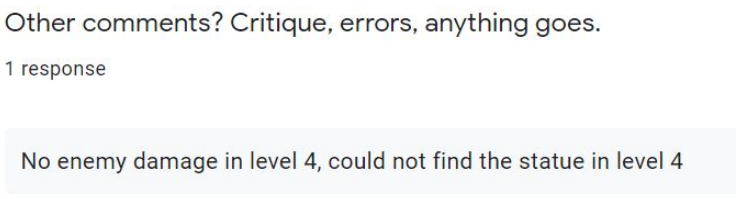
Svar fra andet spørgeskema, femte spørgsmål, 27/5-2020

På trods af de mange ændringer i spillet er der stadig en smule utilfredshed med sværhedsgraden af spillet. En af testpersonerne udtrykte at monstrene er for nemme at

undvige, fordi de er for langsomme i forhold til spilleren. Den anden testperson syntes dog at ændringerne var fine og havde ikke problemer med hastigheden på monstrene.



Svar fra andet spørgeskema, sjette spørgsmål, 27/5-2020



Svar fra andet spørgeskema, sidste spørgsmål, 27/5-2020

Denne fejl blev efterfølgende testet af den ansvarlige for headsettet, og det blev fundet at Unitys rammer for kollision var blevet sat forkert i forhold til, om monster-objektet overhovedet kunne ramme spilleren. Dette var selvfølgelig ikke meningen, og blev hurtigt udbedret, dog desværre først efter test.

Opsummering af anden test

Generelt gav testpersonerne positiv feedback til de ændringer gruppen havde foretaget i spillet. Der viste sig stadig at være en del der kunne arbejdes på for at kunne tilfredsstille begge testpersonerne, og gruppens vision om projektet. Dette kan bruges til videreudviklingen af spillet i fremtiden. For at kunne komme alles forventninger og krav til spillet i møde, kunne der eksempelvis udvikles 3 sværhedsniveauer, ‘easy’, ‘medium’ og ‘hard’, så spilleren kan vælge sværhedsgraden i startmenuen - så spillet imødekommer alle spilleres behov for udfordringer.

Process og refleksion under fjernarbejde

Gruppen har været stærkt udfordret undervejs i skriveprocessen, i form af udbruddet af Covid-19 virusset, som har gjort det vanskeligt at mødes og arbejde på projektet. Det har medbragt en hel del tvivl og uvished om hele projektet, hvordan samarbejdet nu skulle kunne fortsætte med et forbud mod at samles i grupper, hvis ikke det er højst nødvendigt, og hvordan en eksamen nu vil komme til at foregå.

Trods udfordringen, har gruppen dog fundet frem til en fornuftig måde at vedligeholde det gode samarbejde og skabe fremgang i projekt-processen, ved at 'mødes' online på *Microsoft Teams*, føre en logbog for hver dag der udvikles på projektet, og hyppigt benyttet sig af Kanban Boardet, som nævnes tidligere i rapporten. Kanban Boardet bruges som redskab til at uddelegerer opgaver og holde styr på hvilke opgaver der er i udvikling og hvilke der er fuldført.
Ydermere har gruppen oprettet et *'Google doc'* dokument, hvori rapporten udformes så alle gruppemedlemmer kan skrive på rapporten og samtidig kunne hjælpe og følge med i hinandens skriftlige arbejde, på én og samme tid.
Gruppen har sørget for at mødes minimum én gang ugentligt på *Teams*, for at hjælpe og sparre med hinanden, følge op på igangværende opgaver og uddelegerer nye hvis opgaverne er udført - alt for at sikre at der hele tiden har været fremgang i projektet og at alle gruppemedlemmer har været med til alle faser i udviklingen af projektet.

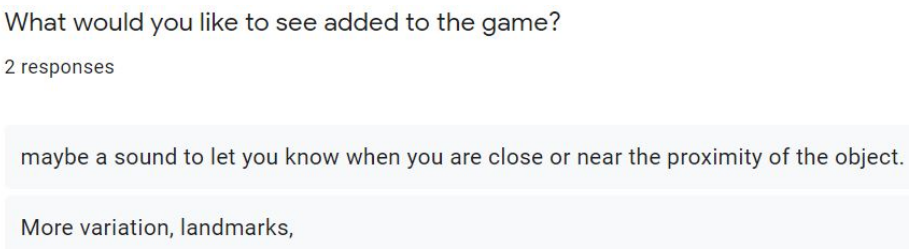
Udover de ugentlige møder online tages der jævnligt kontakt til vejleder, Martin Kirkegaard, og som udgangspunkt planlægges møder, en gang hver 14. dag, for at kunne få hjælp og vejledning når der opstår udfordringer og særligt i starten af online-forløbet.
Ydermere har det været muligt at live-code online, så alle gruppemedlemmer har kunne se koden og være med under udviklingen af spillets og monsterets kode.

Fra den 14. maj og frem til aflevering af projektet, har gruppen aftalt at mødes online hver dag kl. 10 eller 11 for at uddele dagens opgaver, briefe hinanden om gårsdagens arbejde og lægge planer for hvornår hvad skal være færdigt. Dette har været med til at sikre at gruppen har kunne nå at finpudse de sidste detaljer i spillet og rapporten, inden aflevering.

Refleksion

Der har fra starten af projektet været mange idéer som vi gerne ville have haft mulighed for at implementere. Desværre, grundet tidspres og Coronakrisen, har det ikke været muligt at få implementeret alle vores idéer. Disse idéer inkluderer bl.a. flere scripts til monstrene for at give variation i dem, udvide på antallet af levels og verdener og indføre yderligere mulighed for at snøre monstrene i form af gemmesteder og gåder.
Hvis der i fremtiden skulle videreudvikles på The Maze Chase spillet, ville en af de vigtigste prioriteringer være at udvide på de scripts der bliver brugt af monstrene. Vores originale plan med disse var at skabe forskellige monstre, med forskellige opførsel. Blandt de originale idéer var monstre som kunne accelerere hurtigere, som gav op efter at have jaget spilleren i et stykke tid og som kunne gøre brug af unikke evner. Idéen bag forskellige scripts var at gøre monstrene mere unikke, samtidig med at give spilleren forskellige udfordringer at overkomme. Hvert script ville give en anderledes måde at undgå monstrene på, f.eks. ville et monster med acceleration måske ikke være en stor trussel i starten, men hvis den fik lov til at jage spilleren længe nok, ville den ende med at være for hurtig til at kunne undgås. Det samme kunne gøres med et monster der jager spilleren i et stykke tid, før den giver op. Her skulle spilleren måske bevæge sig hurtigt i en given tid, da det ikke ville være muligt at undvige monstret før det stoppede op.
Vi havde også diskuteret om at gøre brug af randomization til monstrene. Dette ville gøre at vi kunne gøre deres ruter, hastighed, acceleration og andre variabler tilfældige, og derved gøre at der kunne være et enormt antal af forskellige monstre, med forskellig adfærd.
Vi havde fra start af udtænkt at der skulle være et væld af forskellige baner/levels, som hver havde deres eget unikke tema og udseende. Vi havde lavet planer om at indføre verdener med temaer som hække labyrint, is, ild/lava, borgruin, kloak/undergrund, rum station/ydre rum, ørken landsby og fængsel. Hver verden ville komme med sit eget sæt af udfordringer og sine egne unikke monstre. Is verdenen ville f.eks. have platforme hvor man glider rundt, og gøre det svært at gå normalt. Ild levellet kunne have skadende overflader, så ikke kun monsteret var en trussel. Det var dog ikke muligt at få lavet alle disse baner, og vi har derfor lavet forskelliger variationer af det ene level vi har lavet.
Vi havde også diskuteret muligheden for at implementere yderligere genstande og udfordringer. I forhold til genstande havde vi originalt planlagt at spilleren skulle finde en nøgle eller anden genstand, som de skulle bruge for at kunne gennemføre et level og komme videre til det næste.

Disse ville dog ikke være de eneste ting, som skulle kunne samles op. Genstande som skulle kunne bruges til at klare forskellige udfordringer, eller give ekstra evner til spilleren var også oppe og blive diskuteret.
Udover dette, var det også originalt planen at spilleren skulle kunne gemme sig fra monstrene, derved gøre det nemmere at undslippe dem. Det var tanken, at der skulle være skjulte rum, rundt omkring i labyrinten, som skulle kunne agere som disse skjulesteder. Ved at gemme sig i dem, kunne spilleren undgå monstre på andre måder end bare at flygte fra dem.



Svar fra andet spørgeskema, syvende spørgsmål, 27/5-2020

Vi har yderligere fået feedback fra vores testpersoner, om hvad de mener der kunne forbedres og udvikles på. Den ene ting de bringer op er en mangel på variation i de forskellige leveller. Dette kunne undgås ved at inkorporere de forskellige verdener som nævnt tidligere i refleksionen. Der blev i denne sammenhæng også foreslået at gøre brug af milepæle, så spilleren kan have et bedre overblik over labyrinten og deres egen position. Dette kunne også inkludere specifikke lyd signaler, f.eks. ved spillets mål. Alt dette kunne gøre spillet engagerende for spilleren.

Konklusion

I løbet af dette semester har gruppen fået erfaring med Unity og brugen af dette til at udvikle spil; der har været mulighed for alle at arbejde med både kodning i form af C# scripts og sammenhæng mellem spil-objekter i editoren, og samarbejde omkring teksturer, modeller og funktioner i spillet har været en stor del af processen. The Maze Chase er endt som et simpelt spil der kan gennemføres indenfor en time, men som samtidig har genspillelighed i forbindelse med sværhedsgrad og rig mulighed for at blive arbejdet videre på efter projektets afslutning. Trods problematikker undervejs i forhold til tekniske udfordringer og fjernarbejde føler gruppen at de kan stå inde for produktet; med hjælp fra udefrakommende er der blevet testet og arbejdet frem mod hvad en spiller ser som værende et godt spil, frem for de ideer medlemmerne i gruppen fra starten havde tænkt at implementere. Fjernarbejde har desuden hjulpet til at

konkretisere hvilke muligheder der var reelle at bruge i udviklingen, idet kun én person havde adgang til Oculus Quest headsettet, så der var ikke mulighed for alle medlemmer at teste forskellige ideer, eller fordele arbejdsbyrden på samme måde når det kom til test af den konkrete bane eller kode.
Gruppen har fundet at meget kode, som på skrift og i Unity editor fungerer perfekt, ikke fungerer når spillet portes til Oculus Quest - dette gør det svært at teste om en kode er færdig, før medlemmet med headsettet kunne få det implementeret i spillet.

Litteraturliste

Artikler

Dempsey, T. (2019). *38+ Powerful Virtual Reality Statistics to Know in 2019.* Fundet d. 12/05-20 på: https://learn.g2.com/virtual-reality-statistics

Bøger

Anderson, D.J & Carmichael, A (2016). *Essential Kanban Condensed* (1. Udgave). *https://resources.kanban.university/guide/*

Sherman, W. R. (2018) *Understanding Virtual Reality-Interface, application and Design (*2. Udgave). fundet d. 18/05-20 på: https://www-sciencedirect-com.ep.fjernadgang.kb.dk/book/9780128009659/understanding-virtual-reality

Hjemmesider

Barnard, D. (2019). *History of VR - Timeline of Events and Tech Development.* Fundet d. 18/05-2020 på: https://virtualspeech.com/blog/history-of-vr

Unity (2019). Unity (2019). *Unity Documentation.* Fundet d. 10/5-2020 på:

https://docs.unity3d.com/Manual/index.html

Unity (2019). *Unity Documentation- Building a NavMesh.* Fundet d. 10/5-2020 på: https://docs.unity3d.com/Manual/nav-BuildingNavMesh.html

Unity (2019). *Unity Documentation- NavMesh.* Fundet d. 10/5-2020 på: https://docs.unity3d.com/ScriptReference/AI.NavMesh.html

Unity (2019). *Unity Documentation- NavMesh Agent.* Fundet d. 10/5-2020 på: https://docs.unity3d.com/Manual/class-NavMeshAgent.html

Assets brugt fra Unity Asset Store

Oculus, Oculus Integration. Downloaded d.21/02-2020 fra: https://assetstore.unity.com/packages/tools/integration/oculus-integration-82022
Press START, PBR-Orc-Pig. Downloaded d. 12/05-2020 fra: https://assetstore.unity.com/packages/3d/characters/creatures/pbr-orc-pig-109248
Unity Technologies, 3D Game Kit. Downloaded d.8/03-2020 fra: https://assetstore.unity.com/packages/templates/tutorials/3d-game-kit-115747
RRFreelance/PiXelBurner, World Space Trees. Downloaded 12/05-2020 fra: https://assetstore.unity.com/packages/vfx/shaders/world-space-trees-free-shader-117088
Vsify, Screaming Statue. Downloaded 12/05-2020 fra: https://assetstore.unity.com/packages/3d/environments/dungeons/screaming-statue-60499
neocrey, Horror Sound Atmospheres and FX. Downloaded 20/05-2020 fra: https://assetstore.unity.com/packages/audio/ambient/horror-sound-atmospheres-and-fx-96731
rpgwhitelock, AllSky Free - 10 Sky / Skybox Set. Downloaded 12/05-2020 fra: https://assetstore.unity.com/packages/2d/textures-materials/sky/allsky-free-10-sky-skybox-set-146014

Journaler

Zhadan, A. (2018). Master Thesis Project: *Artificial Intelligence Adaptation in Video Games.* Fundet d. 8/05-20 på: http://lnu.diva-portal.org/smash/get/diva2:1269116/FULLTEXT01.pdf

Videoer

Octo Beard (2018). *Enemy Patrol Path in Unity [5 Minute Unity Tutorial]* | *Octo Beard.* Fundet d. 10/04-20 på: https://www.youtube.com/watch?v=KqkW3FsuPoM&list=PLMHmRB0Gh0RiZ8CMBXJfakwq5EMRPwAh7&index=11