Denne rapport er udviklet som en del af datamatikeruddannelsens 5. semester på UCN. Rapporten samt tilhørende produkt udgør afgangsprojektet og er udarbejdet i perioden 1. april til 12. juni.

Zen Rabbit Studios

Afgangsprojekt
Datamatikeruddannelsen
UCN 2014

Af
Ronnie Hemmingsen
Toke Olsen

Indhold

Indledning

Formålet med denne rapport er at undersøge, hvad der indgår i processen at udvikle et softwareprodukt og udgive det. I rapporten forsøges der at tage udgangspunkt i et virksomhedsperspektiv, hvor udviklingen og produktet skal danne grundlaget for en sund virksomhed.

Rapporten indeholder derfor en række emner, som er vurderet relevant for projektet. Det første, som rapporten dykker ned i, er en teknologianalyse, der redegør for de tekniske værktøjer mv., som er nødvendigt i dette projekt.

Det næste er en markedsanalyse, hvis formål er at undersøge, hvordan man succesfuldt driver en virksomhed. Analysen indeholder statistikker omkring salgsplatforme, demografi mv. Efter markedsanalysen kommer en virksomhedsanalyse, hvor der bliver redegjort for de omkostninger en nystartet virksomhed kan forvente. Derudover forsøges der i afsnittet af beskrive de styrker og svagheder, virksomheden besidder, for dermed at give et indblik i hvilke tiltag, der skal udføres for at øge chancen for succes.

Den næste del af rapporten beskriver den proces produktet har gennemgået. I denne del bliver der gjort rede for designfasen, hvor produktidéen mv. blev til. Endvidere beskrives det, hvordan produktet under og efter udviklingen skal testes. Herefter kommer den tekniske dokumentation af det endelige spil. Her gives der kodeeksempler for at give et indblik i funktionaliteten i spillet.

Rapporten indeholder ydermere en gennemgang af den arbejdsudvikling, som projektet er gennemgået. Sidst afsluttes der med en perspektivering samt konklusion, hvor der samles op på produktet og projektet som hele.

Hvad er et spil

Spil er et ganske vidt begreb, som kun bliver bredere dag for dag. I det store hele kan man argumentere, at såfremt et spil kræver brugerens input for at nå sit potentiale, så er det et spil. Hvis det ikke var tilfældet kunne der lige så vel være tale om en bog, en film eller en anden slags fortælling. Det interaktive element er det som adskiller et spil fra disse andre ting.
Samtidigt har spil også elementer fra snart sagt alle de mere traditionelle kunstformer: film, bøger, musik, billedkunst mm. Spiludvikling kræver en kombination af alle disse kunstformer, og dertil at brugeren aktivt involveres.

Det betyder altså, at for at kunne producere et spil skal der samtidigt produceres en lang række grafiske og lydmæssige kunstværker. Dem skal vi så søge at knytte sammen via kode, til et produkt som er underholdende og attraktivt for så mange mennesker som muligt.

For effektivt at kunne producere vores første spil, skal vi bruge forskellige typer af værktøj og mandskab. Eksempelvis skal det undersøges, hvilke billedredigerings- og modelleringsværktøj vi kan gøre brug af. Der skal findes et passende udviklingsværktøj, som kan bruges til at samle de forskellige komponenter. Vi skal finde en måde at producere de lyd-assets, som vi har behov for, og sidst skal det undersøges, hvilke andre typer teknologier og værktøj vi har behov for.

Spiludvikling kan overordnet set deles op i to kategorier, den ene side er den kunstneriske, som just beskrevet, mens den anden handler om godt købmandsskab. Det er irrelevant om vi producerer verdens bedste spil, hvis vi ikke samtidigt kan formå at få det solgt med profit, hvorfor skal der laves en markedsanalyse. Formålet er med rimelig sikkerhed at kunne konstatere om der faktisk findes en køber til vores produkt.

I løbet af denne proces er det også væsentligt at få undersøgt, hvilke salgskanaler det er muligt at gøre brug af, samt hvilken, eller hvilke, monetiseringsstrategier, som kunne være relevante at anvende.

For at være i stand til at arbejde effektivt og målrettet skal der vælges en passende udviklingsproces, og der skal udarbejdes designdokumenter således, at alle de projektinvolverede trækker i den samme retning.

Efter gennemgangen af det ovennævnte bliver det nødvendigt at tage et skridt tilbage og vurdere, om virksomheden besidder de nødvendige ressourcer for at kunne påbegynde spilproduktion.

Problemformulering

I dette afsnit præsenteres projektets problemformulering. Den består af et hovedspørgsmål, som er virksomhedens overordnede problem samt nogle uddybende underspørgsmål.

Spørgsmålene er udledt af de problemer og behov, som forundersøgelsen har afsløret i forbindelse med opstarten af Zen Rabbit Studios.

Hovedspørgsmål

- Hvordan opstartes et spilfirma.

Delspørgsmål

- Hvilke forretningsmæssige initiativer skal virksomheden sætte i værk for at kunne eksistere.
- Hvilke tekniske forudsætninger er der bundet op på spilproduktion.
- Hvordan udvikles et spil.

Teknologianalyse

For at kunne bygge et spil skal vi i princippet bruge tre forskellige typer assets: Grafik, lyd og kode, samt den mest passende teknologi til at samle delene til et spil. Dertil kommer så, hvad der er behov for i forbindelse med versionsstyring og testhardware.

For at kunne identificere de teknologier, der matcher vores behov, gennemgås der i dette kapitel en række alternativer på hvert enkelt felt.

Valg af spilmotor

Det er af afgørende betydning for virksomhedens fremtid, at vi vælger det rigtige udviklingsværktøj fra starten. At skifte fra en spilmotor til en anden er ensbetydende med et dyk i firmaets akkumulerede knowhow, og en periode med indlæring af nye systemer, som ellers kunne være brugt på udvikling.

Der findes naturligvis en lang række andre [1], både proprietære og kommercielle spilmotorer, men for at bevare overskueligheden fokuseres der udelukkende på de følgende, da de repræsenterer mest udbredte og veldokumenterede.

CryEngine

CryEngine 4 [2]	
Versioner:	Gratis til uddannelse og non-kommerciel brug. Ellers individuelt aftalt licens. Abonnementsordning på vej: 50kr/md
Unikke Features:	Bliver ofte fremhævet for dens håndtering af tung grafik.
Scripting sprog:	LUA
Understøttede platforme:	PC, Xbox 360, Xbox One, PlayStation®3, PlayStation®4 or Wii U.
Fokus:	Tung grafik, store teams, store projekter, PC og konsol

TABEL 1 - CRYENGINE FEATURES

CryEngine udvikles af det tyske CryTek, og har som udgangspunkt kun været brugt af store og veletablerede firmaer. Grunden dertil skal formentlig findes i, at et spilfirma skal have en vis størrelse for at kunne retfærdigøre udgiften til en licens. Derudover har CryEngine bygget sit ry på at levere de subjektivt flotteste spil i branchen. Det har værdi for AAA-udviklere, men er nok mindre væsentligt for et lille spilhus.

Det er usikkert, hvad en pro-licens præcist koster, da priserne varierer efter størrelsen på det firma, der henvender sig. Der er dog tegn, som peger i retningen af, at CryTek skal have 20% af bruttofortjenesten ved licenser givet til indie-udviklere [3].

Dog har CryTek for nyligt annonceret [4], at de vil implementere en abonnementsplan til ca. 50kr. om måneden og uden at kræve royalties for udgivne titler. Det er endnu ikke officielt annonceret om der bliver adgang til kildekoden, eller om der er andre restriktioner, der skal tages højde for. Umiddelbart gør det dog, at det med ét slag er blevet den billigste af de undersøgte spilmotorer.

Der er et aktivt fællesskab omkring CryEngine, og der er adgang til forskelligt undervisningsmateriale. Men af de spilmotorer, som der her tages udgangspunkt i, er det den der virker til at have det mindste

fællesskab. Konkret betyder det, at udviklingen af spil vil tage længere tid og medføre flere frustrationer, end det ellers havde været tilfældet.

Unity3D

Unity3D	
Versioner:	Gratis udgave (inkl. Gratis iOS og Android moduler) Pro-version til 8.000kr. (abb. 400kr/md) Pro-iOS: +8.000kr. (abb. +400kr/md) Pro-Android: +8.000kr. (abb. +300/md)
Unikke Features:	Massiv brugerbase giver svar på alt
Scripting sprog:	C#, UnityScript (JS mod), Boo
Understøttede platforme:	PC, Mac, Linux, Xbox 360, Xbox One, PlayStation®3, PlayStation®4 or Wii U, WebGL, iOS, Android, Windows Phone, BlackBerry.
Fokus:	Generel spiludvikling, mindre teams, og mindre projekter. All-round devices

TABEL 2 - UNITY3D FEATURES

Indtil tidligere I år var der reelt kun én spiller på markedet for virksomheder af vores slags, og det var Unity. Derfor er der også et massivt community omkring Unity-motoren, og da der er rigtigt mange nybegyndere, betyder det at mængden besvarede spørgsmål omkring brugen af spilmotoren, og spiludvikling i det hele taget er massiv. Dertil kommer, at Unity selv er meget aktive med udviklingen af læringsmateriale. Det er også et absolut plus, at det er relativt let at portere et spil fra en platform til en anden, og Unity har pt. klassebiblioteker til alle de platforme, som er relevante for virksomheden.

Med en prisskilt på 0 kroner for gratis-versionen, der indeholder langt de fleste af systemets komponenter, er der intet at sige til, at mange nybegyndere har taget Unity til sig. Paradoksalt nok betyder det til gengæld, at det kan være svært at finde decideret læringsmateriale om mere avancerede emner.

Der er ikke mulighed for at redigere direkte i Unity's kildekode, men man kan tilføje sine egne komponenter, og af den vej bøje systemet til sin vilje.

Prisen på pro-versionen er 8.100 kr. for en licens til den cyklus Unity er i, og den ejer man så permanent. Hvis man vil opgradere til den efterfølgende cyklus, koster det det halve. Nu, og indtil Unity 5 lanceres, kan man forudbestille version 5, og få adgang til Unity 4 Pro indtil da.

Hvis man vil udvikle til smart devices, koster det yderligere 8.100 kr. for iOS pro og igen 8.100 kr. for Android Pro.

Man får altså ikke adgang til f.eks. iOS pro features selvom man har Unity pro.
Det er et omdiskuteret emne, men det betyder, det hurtigt kan eskalere i prisen. Især fordi der her er tale om kun én licens. Hvis et team er på 4-5 mand, hvilket er typisk, skal man altså gange op.
Unity har også en abonnementsordning, hvor man kan leje hver enkelt del for 400 kr./md.
Benytter man sig af denne løsning, så er det værd at bemærke, at man binder man sig for et år af gangen.

GameMaker

GameMaker Studio	
Versioner:	Flere versioner, med forskellig pris efter features Fra gratis til 4.300kr.
Unikke Features:	Lav et spil helt uden programmering
Scripting sprog:	GML (Game Maker Language)
Understøttede platforme:	PC, Mac, Android, HTML5, iOS, Linux, Windows Phone, Tizen
Fokus:	For den absolutte nybegynder, Små projekter, PC, Web, mobil.

TABEL 3 - GAMEMAKER FEATURES

GameMaker, som udvikles af YoYo Games, er mest at betragte som nybegynderens første værktøj. Som bruger kan man udvikle et spil uden at skulle skrive en eneste linie kode. Af den grund er det vidt udbredt blandt hobbyister. Alligevel er der eksempler på succesfulde spil der er udviklet med GameMaker, senest Hotline Miami, Spelunky og Risk of Rain. Omvendt er der ingen større titler på deres showcase liste, hvilket indikerer, der er nogle begrænsninger i den pipeline, som systemet er bygget op over.

Selvom det er muligt helt at undgå kode, så er der mulighed for selv at scripte ved brug af det javascript-lignende GML-sprog. Selvom GameMaker kan fås fra 0kr, så kræver det Master udgaven til 4.300 kr., for at kunne eksportere til iOS og Android.

I lighed med Unity3D er der et stort fællesskab omkring systemet, hvilket væsentligt bidrager til at stabilisere læringskurven.

Unreal Engine

Unreal Engine 4	
Versioner:	Individuel prissætning. Abonnementsordning 100kr/md + 5% af brutto
Unikke Features:	Fuld adgang til kildekoden.
Scripting sprog:	C++
Understøttede platforme:	PC, Mac, iOS, Android. Konsoller er ikke omfattet af abonnementsordningen.
Fokus:	Store teams, store projekter, PC og konsol

TABEL 4 - UNREAL ENGINE FEATURES

Unreal Engine udvikles af Epic Games, og har navn efter det klassiske FPS Unreal fra 1998. UE har været anvendt i en meget lang række spil siden da bla. Mass Effect og Gears of War-spillene. Selv om den oprindeligt primært var til brug for FPS-spil, viser listen, at det har været muligt at udvide den og lave tilpasninger, så den har kunnet bruges til spil i alle genrer.

Indtil nu har spilstudier kunnet forhandle sig til en licens hos Epic. Der er ikke angivet faste priser, så man kan kun spekulere i, hvad det har kostet.
Siden 2009 har man kunnet hente en gratis "light"-udgave af Unreal Engine 3, kaldet UDK (Unreal Developer Kit). Hvis man udviklede og udgav noget ved brug af den, kostede det et engangsgebyr på 535kr. samt 25% af bruttofortjenesten.

Ligesom tilfældet var med CryEngine, er det tidligere på året blevet annonceret, at Unreal Engine 4 bliver tilknyttet en abonnementsordning, hvor man for godt 100kr/md kan få en fuldt opdateret version af spilmotoren. Dog skal man også aflevere 5% af bruttofortjenesten på ens udgivelser. Hvis man benytter sig af denne model kan man udgive til Mac, PC, iOS og Android, men ikke konsoller. I givet fald skal man igen tage kontakt til Epic for en handel.

Epic stiller dokumentation og læringsvideoer til rådighed, og der er også et dedikeret fællesskab, hvor man kan finde løsninger. Og i modsætning til de andre systemer er der her mulighed for at redigere direkte i motorens kildekode.

Byg eget framework

Den sidste mulighed er naturligvis, at firmaet går i gang med at udvikle sit eget framework. Det ville medføre 100% fleksibilitet i forhold til firmaets fremtidige udvikling. Samtidigt kan en god spilmotor generere massiv indtægt til virksomheden, hvis den kan sælges eller lejes ud til andre udviklere, men det er en massiv opgave at give sig i kast med. Det kræver et højt teknisk niveau, og det kræver kapital der kan understøtte virksomheden indtil den er klar til brug. Det kan tage flere år at nå til det punkt.

Opsummering

Selvom både CryEngine og Unreal Engine 4 er billigere i anskaffelse end Unity (For den fulde version), så er konklusionen at firmaet bør anvende netop Unity. Det skyldes, at fælleskabet omkring dette system er enormt, og beviseligt gerne deler ud af dets erfaringer. Der er ikke det spørgsmål, som ikke er blevet stillet og besvaret, og det alene er nærmest nok til at vælge Unity frem for nogen andre.

Dertil kommer, at alle de involverede i virksomheden allerede har stiftet et indgående bekendtskab med Unity. Det er ikke nogen triviel opgave at blive omskolet til en ny spilmotor, så det trækker også væsentligt i retning af Unity.

De sidste par måneders røre, som er forårsaget af at både Epic og CryTek har annonceret at deres systemer bliver tilgængeligt i et prisleje, hvor selv de mindste kan være med, betyder dog, at det ikke er sikkert, at virksomheden skal binde sig for meget til én platform.
Netop den type overvejelser er en vægtig grund for at udvikle sit eget system, om det så er et framework, eller en decideret Game Engine. Hvis man har muligheden for det, så har man også selv al kontrollen.

Det er dog ikke en realistisk mulighed for dette firma, da ingen af de involverede har erfaring med den type udvikling. Selv i bedste fald ville der gå årevis før en egenudviklet spilmotor ville kunne bruges til seriøs udvikling, og i den tid ville der ikke være nogen indtægter, og ej heller nogen mulighed for virksomheden for at positionere sig på markedet.

Angående GameMaker, så er det måske nok et system, som vi bør afprøve, da det muligvis kan være tidssparende i forhold til udvikling af prototyper. Som primær spilmotor virker det dog afskrækkende, at der ikke findes eksempler på store produktioner i deres showroom. Det er ikke i virksomhedens bedste interesse at binde sig op på en motor, hvorfra vi ikke kan skalere til større produktioner.

Altså bør virksomheden udvikle det første spil med Unity3D.

Valg af udviklingsmiljø

Da vi udvikler i Unity3D, har vi mulighed for at gøre brug af tre forskellige scripting sprog: C#, JavaScript/UnityScript og Boo. Af de tre er vi bedst påklædt til at bruge C#, da det er et sprog, vi har modtaget undervisning i. Samtidigt er det også et sprog, som benyttes vidt og bredt både i forbindelse med spiludvikling, og mange andre sammenhænge. UnityScript benyttes udelukkende i forbindelse med udvikling i Unity, hvilket gør det til et mindre attraktivt sprog at specialisere sig i. Samlet set konkluderer vi, at vi skal anvende det IDE, som bedst understøtter C#. Det betyder enten Visual Studio, som er

Microsofts proprietære system, og som vi skal bruge en licens for lovligt at måtte anvende. Alternativt kan vi benytte os af deres gratis version Visual Express, som er en "lite"-version af den samlede pakke.

Som en tredje løsning kan vi benytte os af MonoDevelop, som er en open source implementering af .Net. MonoDevelop kommer med Unity3D installationen.

Selv om Visual Studio har den bredeste featureliste af de to IDE´er, så er det ikke afgørende for os, da vi faktisk kun har behov for at benytte os af en autocompletion-funktionalitet, samt en syntax-highlighter. Disse features findes i alle de nævnte IDE'er, hvorfor vi uden videre kan benyttes os af MonoDevelop eller Visual Express.

Grunden til at det forholder sig sådan er, at det er Unity, der skal kompile koden. Der er ingen executable som Visual Studio kan benyttes sig af, og derfor kan vi heller ikke benytte os af de indbyggede debug-værktøjer, som ellers ville gøre Visual Studio til den oplagte kandidat.

Valg af billedredigeringsprogram

For at kunne bygge spil, skal vi bruge en billededitor. Vores grafiker er erfaren i at arbejde med Blender til 3D-modeller og animationer og Gimp til billedredigering, og da det samtidigt er gratis-produkter, så det er oplagt at fortsætte med at benytte disse værktøjer.

Ikke desto mindre er vi opmærksomme på, at der er alternativer, som vi kan få brug for på længere sigt. Den alternative kandidat i forhold til billedredigering er Photoshop, som er en standard i industrien. Ud over Photoshop producerer Adobe også en række andre interessante produkter, som f.eks. Illustrator, som på længere sigt kunne blive interessante.

Hvad angår 3D-modellering, så er det den samme situation, Blender er tilstrækkeligt til vores umiddelbare behov. Men der findes mere professionelle produkter som f.eks. Maya og 3D Studio Max som det på længere bane være interessant at lave en mere konkret analyse.

Produktionen af produktions grafikassets er alene i hænderne på firmaets grafiker, og vi benytter os af de produkter, som han vurderer er passende fra projekt til projekt. Derfor nøjes vi med at konstatere, der findes alternativer, men at vi som udgangspunkt benytter Blender og Gimp.

Lydeffekter og musik

Der findes en lang række betalte og gratis systemer, som kan benyttes i forbindelse med lydoptagelser til både effekter og musikkompositioner. Der er dog ingen kompetencer i brugen af dem at finde blandt firmaets ansatte.

Indtil den situation kan ændres vil vi ikke lave analyse af nogen af disse værktøjer. I stedet agter vi at benytte os af gratis, og betalte, lydeffekter og musikstykker som er produceret af professionelle. Lyd-assets kan købes igennem Unity's webshop eller igennem andre specialiserede webbutikker.

Versionsstyring

For at sikre, at vi har fuld kontrol over vores produktion, og ikke er i farezonen for at miste data, er det afgørende at vi har versionskontrol på vores projekt. Der findes en lang række muligheder inden for området, betalte såvel som gratismodeller. I det følgende afsnit gennemgår vi et udsnit af dem.

Unity Team Server

Unity producerer selv et "Team-License"-produkt [5], der kan integreres direkte i Unity. Det koster 2.700 kr. pr person, og der er ikke nogen Sky-funktion inkluderet, hvilket betyder, at vi selv skal have en server

sat op. Til gengæld er Unity produktet skræddersyet til at håndtere både den type af filer som Unity producerer, og den størrelse de kan forvente at komme i.

GitHub

Alternativt kan vi betale os fra en privat konto hos f.eks. GitHub. De kan fås fra 40 kr. om måneden. Til gengæld begrænser vi os til projekter, der er mindre en 1GB, og filer, som ikke overstiger 100MB. Ligeledes anbefaler GitHub [6] [7], at man bruger et andet redskab til "mellemregninger" af tungere typer filer, som f.eks. 3D-modeller.
Strengt taget behøves der ikke betales for en GitHub-konto, men gratis-kontoerne er offentligt tilgængelige. Det er ikke noget problem i første omgang, men det er en ulempe, hvis vi skal arbejde med en tredjepart, eller generelt vil holde et projekt lidt tættere til kroppen.

Bitbucket

Bitbucket er et alternativ til GitHub. I modsætning til GitHub er det muligt at få private repositories gratis for op til 5 brugere. Samtidigt kan vi have et ubegrænset antal repositories, hvilket vil sige, at så længe firmaet har maksimum 5 personer tilknyttet med adgang, så kan vi få gratis opbevaring. I øvrigt er der integration med Jira indbygget, hvilket er relevant at tage i betragtning, når vi kommer til et tidspunkt, hvor vi får brug for et reelt bugtracker-værktøj. I forhold til projekt- og filstørrelser, så er det i hovedtræk de samme regler der gør sig gældende her, som hos GitHub.

Dropbox

Vi kan også benytte et værktøj som Dropbox. Dels er det let at dele materiale imellem projektdeltagere, men det er også muligt at sætte et versionsstyringssystem op direkte i en mappe. Igen kan man benytte sig af gratisløsningen, hvor hver bruger har 2GB (I udgangspunktet), men det er ikke stort bedre end med GitHub. Den optimale Dropbox løsning ville koste 400 kr./md.

Opsummering

Fra starten udelukker vi selv at sætte et system op. Det valg begrundes med, at de ressourcer, som det kræver, sammenlagt med opstart af virksomhed ville tage overhånd.

Hvis vi selv satte en server op, så ville det også medføre, at vi selv skulle stå for vedligehold af den, håndtere back-ups og opgraderinger og meget mere. Vi er et spilfirma og ikke en hostingservice. Eller sagt på en anden måde, det er ikke et område, vi vil bruge vores tid og energi på.

Givet det ovenstående har vi besluttet at bruge en kombination af Dropbox og GitHub til vores første projekt. Alle midlertidige assets gemmer vi i vores Dropbox-mapper, og de bliver først importeret til vort Unity projekt når vi er klar til at implementere dem i spillet. Selve Unity-projektet er decideret versionsstyret og ligger på en gratis-konto på GitHub. Det er ikke specielt relevant lige nu, at det er offentligt tilgængeligt, for det er svært at forestille sig, at nogen skulle være interesserede i det. Hvis det bliver vurderet som værende nødvendigt, kan vi slette det Git Repository, som projektet ligger i, når spillet er klar til udgivelse.

Af selvsamme grund kunne Bitbucket være et relevant produkt til det næste projekt. Når vi har valgt det fra i denne omgang, så skyldes det at vi har erfaring med at bruge GitHub, og det er ikke et område, vi er interesserede i at bruge energi på i denne ombæring.
Reglerne angående fil- og projektstørrelse behøver vi i første omgang ikke bekymre os om, da det er er urimeligt at forestille sig vores projekter overstige 1GB. Når vi kommer til lidt større projekter, så er det et emne, der kan tages op igen.

Markedsanalyse

I dette afsnit vil vi analysere og danne os et overblik over underholdningsindustrien som helhed. Dette skal danne grundlag for resten af markedsanalysen, da det er relevant at kende til den overordnede udvikling af industrien. I analysen bliver der gennemgået statistikker omkring spilplatforme, salgsplatforme, segmenter samt målgruppeanalyse, hvor hver enkelt leder op til den næste og har fokus på det den retning, som projektet skal tage.

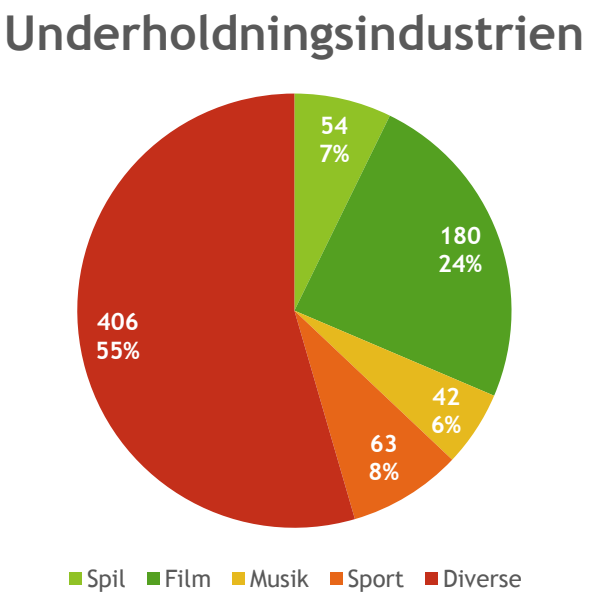
Dette afsnit afsluttes med en konklusion, hvor der bliver gjort overvejelser over, hvordan virksomheden bedst udnytter disse informationer.

Analyse af spilmarkedet kontra Anden Underholdning

Inden for underholdningsbranchen findes der et væld af forskellige underbrancher og nicher, som er i indirekte konkurrence med spilindustrien. Det er vigtigt at redegøre for, hvilke industrier, som har fremgang, samt hvilke industrier, som svinder ind, da dette teoretisk set kan have indflydelse på virksomhedens fremtid. Dog betyder størrelsen af industrien, at projektets succes ikke afhænger af industriens umiddelbare fremtid.

Underholdningsindustrien er i stor fremgang og er vokset med over 50% det sidste årti og er vokset fra $449 mia. i 1998 til $745 mia. i 2010 [8].

"The movie industry cleared 180 billion US dollars in 2009. Sports earned 63 billion, computer games 54 billion, live performances 35 billion, and music 7 billion. Global entertainment from all sources is expected to reach 1.4 trillion dollars by 2015."
- prweb.com



FIGUR 1 - OMSÆTNING FOR DEN SAMLEDE UNDERHOLDNINGSINDUSTRI, FORDELT PÅ OMRÅDER.

Som det ses i Figur 1 er spilindustrien en relativ lille del af underholdningsindustrien, men tager man spilindustriens alder i betragtning, så har den haft enorm fremgang i forhold til de andres, på ganske kort tid.

"No other sector has experienced the same explosive growth as the computer and video game industry. Our creative publishers and talented workforce continue to accelerate advancement and pioneer new products that push boundaries and unlock entertainment experiences. These innovations in turn drive enhanced player connectivity, fuel demand for products, and encourage the progression of an expanding and diversified consumer base. "
- Michael D. Gallagher, president and CEO, Entertainment Software Association

FIGUR 2 - UDVIKLINGEN I OMSÆTNINGEN I SPILINDUSTRIEN [9].

Som det ses af grafen i Figur 2 har spilindustrien haft stor positiv udvikling, specielt de seneste år, og vi har ingen grund til at forvente at den aftager i fremtiden. På dette grundlag er det derfor naturligt at konkludere, at computerspilsbranchen klart bør overvejes at føre virksomhed i.

Analyse af spilplatforme

Spilindustriens platforme har enorm betydning for udviklingen og produktionen af nye spiltitler af flere grunde. Først og fremmest er det en teknisk udfordring, da de forskellige platforme er udviklet forskelligt, og det kræver derfor ressourcer fra udviklernes side at tilpasse en spiltitel til en ny platform. Dette gælder og ressourcestyring i form af computerkraft, som spænder vidt fra små håndholdte enheder til optimerede Pc'er. Derudover har platformene vidt forskellige interaktionsmuligheder fra touch skærm til joystick til mus og tastatur.

Spilindustrien kan opdeles i 5 forskellige hovedplatforme, som ses i Tabel 5 herunder [10].

Spilindustriens indtægt på verdensplan (i mio. USD)

Segment	2012	2013	2014	2015
Spilkonsoller	37.400	44.288	49.375	55.049
Håndholdte konsoller	17.756	18.064	15.079	12.399
Mobile spil	9.280	13.208	17.146	22.009
PC-spil	14.437	17.722	20.015	21.601
Total	78.872	93.282	101.615	111.057

TABEL 5 - SPILINDTÆGT PÅ VERDENSPLAN FORDELT PÅ PLATFORM

Det der kan læses ud fra Tabel 5 er, at næsten alle områder har stabil fremgang, kun med undtagelse af håndholdte konsoller (Nintendo DS, PS VITA mv.) som forudsiges at miste markedsandele. Det skyldes sandsynligvis den enorme fremgang af mobile spil, som i væsentlig grad opfylder det samme behov som de håndholdte konsoller. Dette hænger tæt sammen med den enorme udbredelse af smartphones i verdenen, som er steget fra 1,13 mia. telefoner i 2012 til 1,75 mia. telefoner i 2014 – næsten en fordobling på få år [11].

Skulle man derfor vælge en branche ud fra disse tal står valget imellem spilkonsoller og mobile spil, hvor begge har haft stor fremgang de seneste år. Af de to er spilkonsolmarkedet det største, hvorimod den mobile platform er mere udbredt i verden. Samtidigt kan der være store omkostninger forbundet med at udvikle til konsollerne, da de respektive platformsudviklere har omfattende procedurer der skal overholdes ved udgivelse til deres produkter.

Det er også værd at notere sig, at stigningen i PC-spil er kraftigt stigende som slutningen af den forgående konsol-generation kommer nærmere. Tabel 5 herover viser tydeligt denne tendens. For at kunne udgive spil til PC, kræves der i princippet intet ud over et produkt.

Som nystartet udvikler er det derfor lettere at udgive til mobile platforme eller PC.

Kvalitetsforventninger

Hvis vi ud fra det ovenstående holder fast i, at vores marked er enten PC eller mobil, så er det vigtigt at forstå forskellen blandt de to målgrupper. Overordnet vil vi benyttede de alment brugte termer "Hardcore" og "Casual" [12].

En "hardcore" spiller er en person (m/k) som virkelig holder af spil, en som har brugt en masse tid på at spille spil og diskutere dem med andre, og i et større eller mindre omfang følger med i hvad der sker i branchen. Derfor har han eller hun helt naturligt opbygget nogle højere krav til kvaliteten af det spil som

personen bruger sin tid på. Spil der henvender sig til denne type spiller kan være af en næsten uanet kompleksitet og scope [13].

En ”casual” spiller derimod, er en som godt kan bruge lidt tid i ny og næ på at spille, men som ikke er interesseret i at bruge energi på at sætte sig ind i komplekse spil [14]. Spil der produceres til casual-spillere er derfor typisk lette at genkende på deres simple design, der gerne er modelleret omkring en enkelt, eller kun ganske få, game mechanics. Oplagte eksempler er Angry Birds eller FarmVille.

Både i forhold til hardcore- og casual-arketyperne, gælder det at der er mange gyldne middelveje; altså brugere som ligger et sted imellem de to yderpunkter. For at kunne bruge opdelingen til noget må vi derfor se på hvilke spil der henvender sig til hvilken gruppe, og hvor de spil befinder sig.

Candy Crush Saga, som fås på mobile devices, og som Facebook-app på PC, er det mest udbredte casual spil for øjeblikket. På Facebook har det små 60 mio. daglige brugere [15], og til de mobile enheder er det downloadet over 500 mio. gange [16]. Men det findes ikke på Steam, GOG.com eller andre dedikerede PC salgskanaler. Det samme scenarie gør sig gældende for mange andre spil af samme type. Det er en kraftig indikator for at den type brugere som de spil har, er meget lettere tilgængelige på de nævnte platforme.

Omvendt findes de mere ”hardcore” spil ikke i samme omfang på mobile enheder (eller Facebook), hvilket delvis skyldes, at maskinkraften på smartphones ikke er den samme som på en PC, men også at markedet for den type spil ikke er der i samme grad som på de dedikerede PC og konsol salgskanaler.
Der findes naturligvis spil som gør sig gældende på alle platforme, i casual-kategorien kan man f.eks nævne Plants vs. Zombies, som startede på mobilplatformene, men nu også findes på PC og konsol som standalone klient. Mens Minecraft der startede som et PC-spil nu findes i en meget populær mobiludgave også [17].

Det er bydende nødvendigt, at vores spil opfylder de forventningskrav som slutbrugeren har, og fordi casual-spillere stiller lavere krav til spillets kompleksitet, og kvaliteten af assets, så vil vi som et lille firma have bedre muligheder for at opfylde de krav, end hvis vi valgte at udvikle til et mere kræsent publikum. For at kunne indfri de forventningskrav, må vi se på eksempler på spil som ligner det vi laver og som minimum sikre at vores produkt er af samme kvalitet.

Det er også et faktum, at mange af de spil som udgives til begge platforme, ikke lever op til selv de mest basale krav [18]. Problemet er især stort på Android, da der ikke foregår nogen godkendelsesproces før et spil bliver tilgængeligt for offentligheden. Apple udfører en form for kontrol, som forhindrer de værste udgivelser i at blive solgt [19]. Der er ganske givet flere grunde til, at netop mobilspil er så plaget af lavkvalitetsprodukter. Men man kunne gisne om, at en del af dem stammer fra amatørudviklere, der benytter de samme teknologier, som vi har gennemgået tidligere.

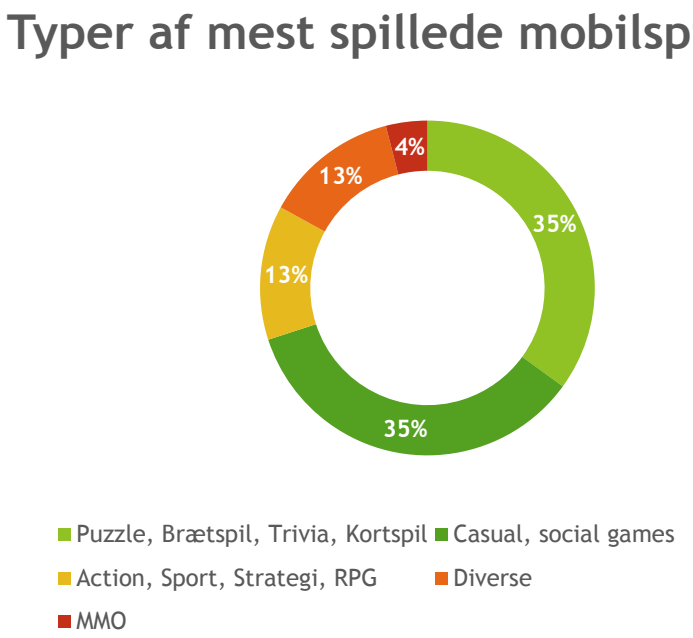
Opsummering

Baseret på det ovenstående, og en vurdering af virksomhedens ressourcer, finder vi, at det vil være mest formålstjenstligt at udvikle vores første spil til en mobilplatform.

Derfor fokuseres der udelukkende på det mobile marked i resten af afsnittet for at undgå irrelevante analyser på brancher, som ikke er i virksomhedens umiddelbare fokus.

Segmentanalyse

Når der er er fundet en platform at udgive til, er der behov for at segmentere branchen i typer af apps, da der i dette tilfælde er fokus på mobilbranchen. I dette projekt er der valgt at lave et spil, og der kigges derfor på, hvilke genrer af spil, som dominerer.



FIGUR 3 - PROCENTOPDELING AF SPILS POPULARITET FORDELT PÅ GENRE

Som ses ud af Figur 3, er de dominerende genrer sociale spil samt "hjernevrider"-spil, som ligger dominerer med 70% af markedet. Derefter er det Action-spil mv., som tager tredjepladsen af populære genrer med 13%, hvor MMO har små 4% af markedet. Ud fra dette burde fokus lægge på sociale spil eller puzzles mv. Kigger man dog på tal fra 2012 har sociale spil haft fremgang, hvorimod puzzles mv. er faldet fra 47% til 35% [20] [21].

Hvis man udelukkende baserede sit valg af spil på tallene i Figur 3, så burde man altså udvikle et Trivia-spil med sociale elementer for at ramme det bredest mulige segment.

Mobile salgsplatforme

Da markedet for mobile spil er i stærk fremgang er det relevant at undersøge, hvilke muligheder, der er for, at udgive sit produkt og samtidigt optimere salget. Først og fremmest er det relevant, at vide hvilke platforme som er populære og som har det største marked. Dernæst skal der kigges nærmere på værdien af disse platforme samt hvilke genrer af apps, som bliver købt.

Smartphone App Statistics	iPhone	Android	Blackberry	Windows
Total app downloads	27 mia.	29 mia.	2,4 mia.	4,1 mia.
Percent of app users who have never paid more than $1 for an app	45 %	62 %	63 %	58 %
Average number of downloaded apps per phone	88	68	49	57
Total number of apps in store	905,000	850,000	130,000	220,000
Total app store revenue in 2013	$6,400 mio.	$1,200 mio.	$550 mio.	$950 mio.

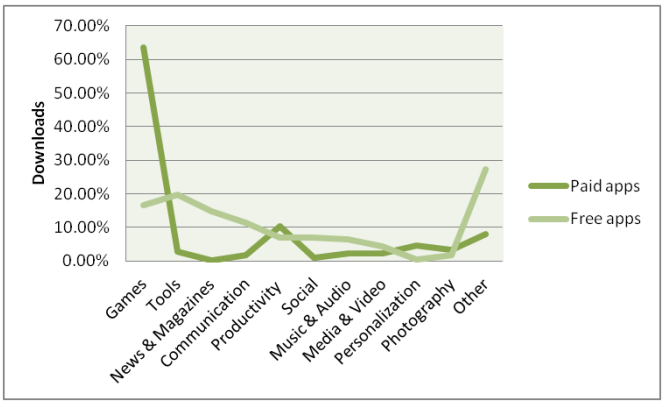
TABEL 6 - ANTAL HENTEDE APPS, FORDELT PÅ SALGSPLATFORME.

App Category Market Share for All Devices	Percent of Downloads
Games	23%
Entertainment	11%

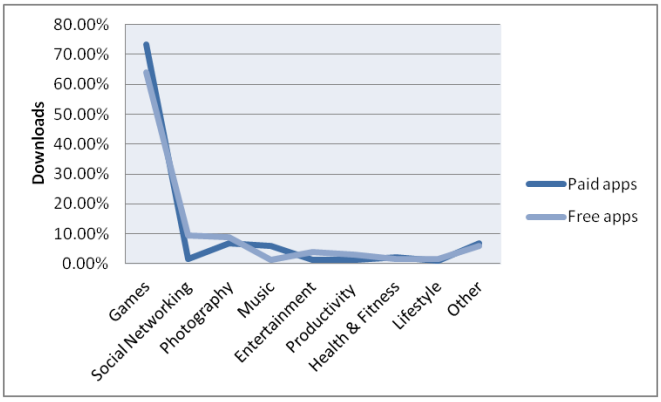
TABEL 7 - PROCENT HENTEDE SPIL OG UNDERHOLDNINGS APPS, UD AF DET SAMLEDE

Jf. Tabel 6 er iPhone- og Android-markedet stærkt dominerende i antal downloadede apps [22] og det vil derfor være naturligt at fokusere på disse som det primære mål for udgivelsen af produktet, da det medfører størst chance for succes.

Dog skal omsætningen af Windows apps bemærkes, da det relative lave antal downloads har givet en omsætning næsten tilsvarende til Androids omsætning. Det kan skyldes, at der er flere gratis apps til Android, end på de andre markeder. Figuren herover ikke tager højde for om en app er gratis, eller benytter sig af IAP, men kun direkte salg. En indikator på, at det forholder sig på denne måde er, at totalen af hentede apps er højere på Android, end på Iphone, mens omsætningen markant højere på iPhone.



TABEL 8 -HENTNINGER FRA GOOGLE PLAY I PROCENT PR. TYPE [79]



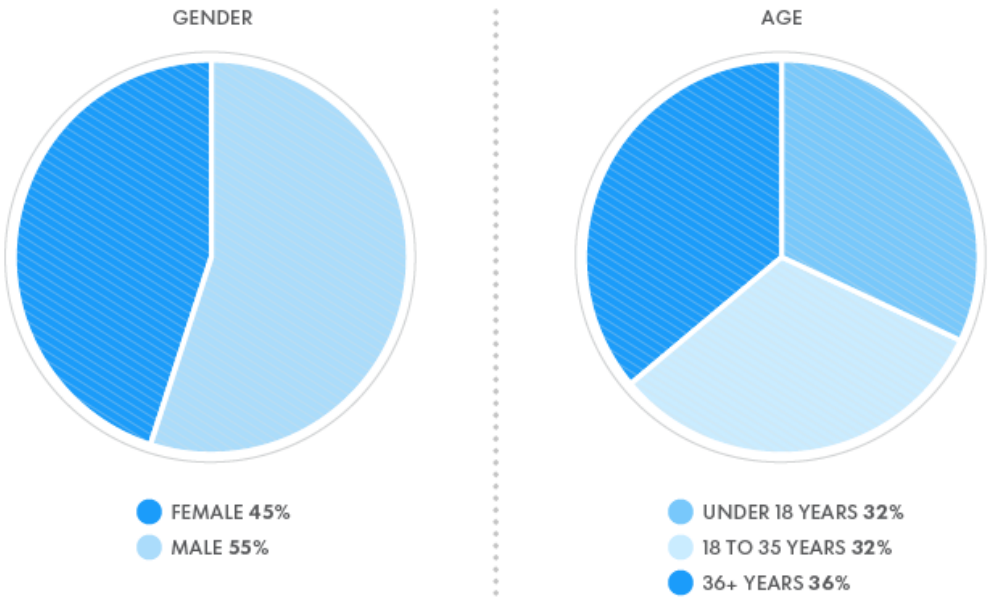
TABEL 9 - HENTNINGER FRA APP STORE I PROCENT PR. TYPE [79]

Af de fire markeder som er repræsenteret herover, er det umiddelbart Apple og Googles markedspladser som er mest valide. Det fremgår desuden tydeligt af Tabel 7, Tabel 8 og Tabel 9 at spil udgør en betragtelig del af det samlede antal hentede apps.

Derfor bør vores primære valg af salgskanal være enten App Store eller Google Play, men den endelige beslutning kommer også til at bero på andre faktorer, eksempelvis test, hardware mv.

Målgruppeanalyse

Efter at have udvalgt en platform er det vigtigt at redegøre for den målgruppe, man håber at nå for at være i stand til at optimere markedsføringen.



FIGUR 4 - KØNS- OG ALDERSFORDELING IMELLEM SPILLERE [23].

Figur 4 viser kønsfordelingen iblandt amerikanere der spiller spil. Som man kan se, er der næsten lige mange mandlige som kvindelige mennesker i vores målgruppe. Ud fra aldersfordelingen kan vi konkludere, at der generelt er flest mennesker under 36, der identificerer sig selv som spillere. Med tanke på, at mediet, bortset fra enkelte tidligere udgivelser, kun er godt 30 år gammelt, virker det rimeligt at antage, at der kommer flere "ældre" kunder til efterhånden, som de der er vokset op med mediet også bliver ældre. Tallene i denne figur dækker også andre platforme end de mobile.

Demografi – Apple og Android

Da det er besluttet, hvori vores fokus er rettet på Apple og Google, er det interessant at se, hvilken forskel, der er i demografien blandt deres kunder.

Demographic differences in iPhone and Android ownership

% of cell owners in each group who own an iPhone or Android

		% who say their phone is an iPhone	% who say their phone is an Android
All cell owners (n=2,076)		**25%**	**28%**
Gender			
a	Men (n=967)	24	31[b]
b	Women (n=1,109)	26	26
Age			
a	18-24 (n=238)	31[ef]	43[cdef]
b	25-34 (n=279)	34[def]	40[def]
c	35-44 (n=283)	29[ef]	33[ef]
d	45-54 (n=354)	25[f]	27[ef]
e	55-64 (n=392)	19[f]	17[f]
f	65+ (n=478)	11	7
Race/ethnicity			
a	White, Non-Hispanic (n=1,440)	27[b]	26
b	Black, Non-Hispanic (n=238)	16	42[ac]
c	Hispanic (n=235)	26[b]	27
Education attainment			
a	Less than high school (n=144)	11	25
b	High school grad (n=565)	17[a]	27
c	Some College (n=545)	27[ab]	31
d	College + (n=799)	38[abc]	29
Household income			
a	Less than $30,000/yr (n=504)	13	28
b	$30,000-$49,999 (n=345)	23[a]	27
c	$50,000-$74,999 (n=289)	25[a]	31
d	$75,000+ (n=570)	40[abc]	31

Source: Pew Research Center's Internet & American Life Project, April 17-May 19, 2013 Tracking Survey. Interviews were conducted in English and Spanish and on landline and cell phones. Margin of error is +/-2.4 percentage points based on cell phone owners (n=2,076).

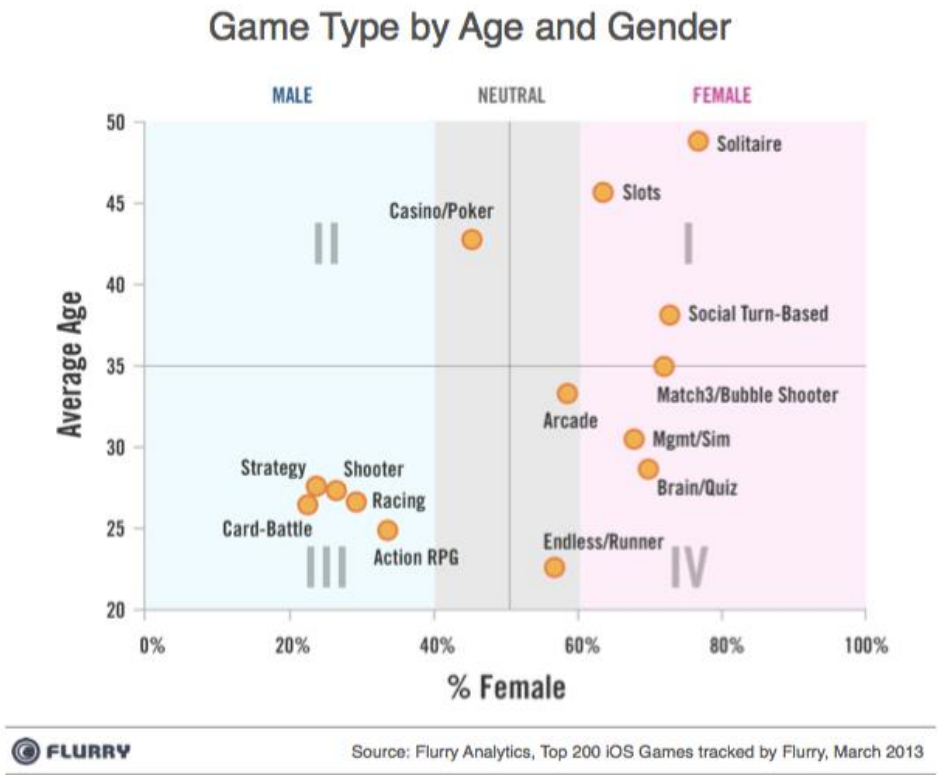
Note: Columns marked with a superscript letter ([a]) or another letter indicate a statistically significant difference between that row and the row designated by that superscript letter. Statistical significance is determined inside the specific section covering each demographic trait.

FIGUR 5 - DEMOGRAFIOPDELING MELLEM ANDROID OG IPHONE BRUGERE.

Det første, og måske vigtigste tal man skal bide mærke i Figur 5, er, at der er flere Android, end iPhone brugere. Det lå også i kortene i de tal, som fremgår af Tabel 6, hvor der er er flere hentede apps på Android. For os betyder det umiddelbart ikke så meget, vi benytter os af et udviklingsværktøj, der gør os i stand til at udvikle til begge markeder. Men givet tallene i Tabel 6 - Antal hentede apps, fordelt på salgsplatforme.så er der kunder uanset hvilken mobilplatform, der udgives på.

Figur 5 viser os dernæst, at der marginalt flere kvinder end mænd, som har en iPhone. Fordelt på alder, er der især mange yngre personer, der ejer en Android telefon, mens tallene er mere jævnt fordelt, efterhånden som brugerne bliver ældre. Det er også værd at notere sig, at den samlede procentdel af de adspurgte, som ejer en af de to typer smartphone, falder drastisk, som alderen stiger.

Hvis man ser på sammenhængen imellem højt/lavt uddannede og ditto lønnede, så er der tilsyneladende en klar sammenhæng i det, at jo bedre ens levevilkår må formodes at være, jo større er chancen for at brugeren køber en iPhone. Det kan være en af grundene til at omsætningen i direkte salg af apps på iPhone er højere end modpartens.



FIGUR 6 - POPULÆRE GENRER FORDELT PÅ KØN OG ALDER [24]

Figur 6 giver et indtryk af, hvilken type spil man kan forvente, at en given aldersgruppe og køn spiller. Genrerne er placeret fra 0 til 100% kvinde, hvilket altså vil sige, at jo længere man bevæger sig ud af x-aksen, jo større er chancen for, at ens bruger er kvinde. Som det fremgik af Figur 5, så er kønssammensætningen i Amerika således, at der 52% kvinder, som ejer en smartphone, og 55% mænd. Mens Figur 4 viser at 45% af de, som identificerer sig som spillere, er kvinder.

Konklusionen må nødvendigvis være at rette markedsføringen mv. mod begge køn.

Piratkopiering

I lighed med PC og konsoller er der en risiko for at få piratkopieret sit produkt, når man udgiver til mobile platforme. Der er delte meninger om, hvor stort problemet er, og alle tilgængelige tal er baseret på enkelte udvikleres anekdoter. Det virker dog til, at problemet er størst på Android platformen, hvilket nok også er med til at forklare, hvorfor mange spil udkommer først til iOS, og i mange tilfælde forbliver der [25] [26].

Grunden til forskellen på piratfrekvensen imellem de to platforme kan være, at det er langt nemmere at installere en piratkopieret kopi af et spil på en Android enhed. Det kan man nemlig gøre uden videre, mens det på iOS platformen kræver, at man "jailbreaker" sin enhed. Det vil sige, at man frigører sin iPhone/iPad fra Apples system, og det er et ekstra skridt som ligger imellem brugeren, og muligheden for at piratkopiere, og ved samme lejlighed betyder det at enhedens garanti ophører [18] [27].

Opsummering

Når der efterhånden er blevet redegjort for de tal og statistikker, som man føler er relevante for ens virksomhed / produkt, så bør det være muligt at danne sig et billede af det marked, man bevæger sig ind på.

I det forgående afsnit har vi redegjort for spilmarkedet som hele. Vi har forsøgt at redegøre for markedets udvikling og om hvorvidt industrien er profitabel også i fremtiden. Baseret på den analyse konkluderer vi, at industrien er velvoksende med en god fremtid og på det grundlag er det oplagt at starte et spilfirma.

Dernæst blev der kigget på underbrancherne, hvor de forskellige tekniske platforme dikterer opdelingen. Her konkluderes det, at udvikling til den mobile platform er et naturligt valg for virksomheden.

Vi kan også se, at det er mest oplagt at udvikle til Android og/eller Apple, og at, hvis vi gør det, så kunne det være en mulighed at lave differentieret betaling. For eksempel kunne vi tage en fast pris for spillet på iOS, mens Android versionen skulle have freemium-elementer.

Sidst bliver der redegjort for det mobile marked i form af segmentanalyse af de forskellige spilgenrer, målgruppeanalyse samt hvilke distributionsplatforme, som er relevante.

Baseret på de oplysninger vi har fået om vores potentielle målgrupper, så konkluderer vi, at det er muligt at lave spil til mænd, kvinder, unge og voksne samt i nærmest alle typer genrer, netop fordi der er massive segmenter i alle målgrupperne. Der hvor vi skal sætte ind er, at genre, målgruppe og type gerne skal passe til det enkelte segment.

Virksomhedsopstart

I det følgende kapitel vil vi kigge på virksomhedens interne ressourcer, de eksterne markedsvilkår og vores position. Formålet er at afdække virksomhedens styrker og svagheder, de eksterne trusler og vores muligheder for at træde ind på markedet.

Virksomhedens interne ressourcer

I dette afsnit beskrives de interne ressourcer, som virksomheden har til rådighed. Det er værd at bemærke, at der i visse sammenhænge, og her tænkes der på de uhåndgribelige ressourcer og virksomhedens evner mere er tale om hensigtserklæringer end reelt tilgængelige ressourcer.

Håndgribelige ressourcer:

- Almindelige windows PC´er
- 1 Samsung Galaxy Tab 10.1 2 (4.4.2)
- 1 Samsung S4 telefon (4.4.2)
- 1 HTC One (4.4.2)
- Unity3D 4.3.4 (gratis version)
- Visual Studio 2013
- Gimp 2
- Blender 2.69
- Trello.com
- Github.com
- SourceTree

Uhåndgribelige ressourcer

Virksomhedens kreative kraft er uden tvivl vores vigtigste kort, vi er, hvem vi er, og da spiludvikling har mange fællestræk med andre kunstformer, kan vores personlighed bruges som et direkte kerneaktivt. Ydermere har er der den fordel ved det, at det er en ressource som ikke kan kopieres af konkurrerende firmaer. Vores personlighed og vores holdning skal skinne klart igennem i alt, hvad vi laver.
Virksomhedens brand er en anden af de uhåndgribelige ressourcer som vi fra starten vil søge at styrke. Et velrenommeret brand kan hjælpe os til mersalg i forhold til almindelige kunder. Men det kan også åbne døre for os i forbindelse med virksomhedens muligheder for at netværke, og dermed direkte øge vores mulighed for vækst.

Virksomhedens evner

Vi har et godt branchekendskab – Vi kender de forskellige kundesegmenter, som vi kan komme i berøring med. Derfor har vi et godt udgangspunkt for at levere et produkt, som matcher det segment som et givent projekt specifikt henvender sig til. Ligeledes kan vi også med stor sikkerhed vælge den rette prisstrategi til spillet. Vi ved, hvad der er kvalitetsstandarden for hver type platform og for de kundegrupper der anvender disse platforme. Dermed kan vi kan placere vores produkter på et passende niveau på en mere effektiv måde. Ydermere besidder vi et solidt kendskab til spil af alle slags, og i kraft af den erfaring er vi i stand til at træffe fornuftige designvalg igennem alle faser af udviklingen.
Virksomhedens personale består af uddannede programmører og grafiker, hvilket dækker en bred vifte af de kompetencer som er påkrævet i forbindelse med spiludvikling.

Vi er fleksible i vores tankegang og vores processer, således at vi hele tiden kan udvikle os som fagpersonel, og som firma. Vi er ikke bange for at op- eller nedskalere firmaet efter behov. På samme måde er vi heller ikke bange for at tage projekter af en type, som ikke umiddelbart følger den fra starten lagte plan, hvis vi vurderer at det giver mening for virksomheden. Det kunne for eksempel være, at udvikle undervisningsmateriale el. lign.

Kernekompetence

Spiludvikling med Unity3D spilmotoren.
Stram projektplanlægning
Virksomhedens kreativitet

Kerneydelser

Spiludvikling – Egen udvikling
Spiludvikling – Eksterne kontrakter

Virksomhedens eksterne miljø

Virksomhedens umiddelbare konkurrenter er alle andre mobil udviklere i hele verden. Det gælder både de, som har penge bag sig, og dem der, som os, starter fra den absolutte bund.
I Danmark er det oplagt at nævne Tactile Entertainment (Chasing Yello), som en direkte konkurrent, men de markedspladser, som vi skal sælge på er globale. I det lys er konkurrencen langt skarpere fra firmaer som Rovio (Angry Birds), ZeptoLab UK (Cut the Rope) eller PopCap (Plants Vs. Zombies).
Derudover vil der være specifikke konkurrenter i forhold til det enkelte projekt, forstået på den måde at vores direkte konkurrent ikke er den samme, hvis vi udvikler et PC-spil, som hvis det er et mobil-spil.

Den teknologiske udvikling er foregået på en måde, hvor udbredelsen af mobile enheder, der kan afspille spil, samt den modning spilmotorer som Unity3D har gennemgået i de senere år, gør det muligt for små virksomheder at udgive spil af høj kvalitet for et meget lavt budget.

Den næste ting, der kommer, bliver formentlig udbredelsen af Virtual Reality enheder. At Facebook for ganske nyligt betalte 2 mia. USD for startup-virksomheden Occulus Rift [28], og Sony´s nyligt annoncerede Project Morpheus signalerer klart at det er et område som tages alvorligt. Pt. Er det kun muligt at købe en prototype af Occulus VR, og Morpheus er slet ikke tilgængeligt. Men det er afgjort et område, som vi som nystartet virksomhed vil holde meget nøje øje med.
Det samlede billede af industrien generelt er noget mudret, fordi spil kommer i mange forskellige størrelser, og dermed prisklasser. I den tunge ende finder man firmaer som Ubisoft og Blizzard, og generelt er der kun en begrænset mængde udviklere, der kan håndtere udviklingen af såkaldte AAA-spil. Jo mere man nedskalerer et spil, jo flere virksomheder findes, der som kan håndtere opgaven, og det er ikke unormalt, at virksomheder i størrelsen 1-5 mand producerer mobil eller pc-spil.

Samlet set er kundegrundlaget massivt. Der er solgt mere end 700 mio Apple devices (tlf og pads), og på Android siden er tallet formentligt omkring en milliard [29]. Selv om disse tal dækker over udgåede modeller, ure, ipods o.lign. typer udstyr der ikke er direkte relevante for salg af apps, så er det dog rimeligt at antage, at der er et tilstrækkeligt kundegrundlag til at starte en virksomhed der udvikler mobilspil.
Dertil kommer at vi også kan udvikle til platforme som: PC, Playstation, Xbox, WiiU eller Webbrowsere. På PC findes der en række markedspladser hvoraf de mest oplagt for os er Steam og GoG. For de andre gælder det at hver platform har sin egen markedsplads, og sit eget kundesegment.

Der er ingen tvivl om at konkurrencen på spilmarkedet er knivskarp. De teknologiske fremskridt, der gør det muligt for os at starte op, har gjort det samme for tusindvis af andre udviklere, og de fleste har langt

mere erfaring med udvikling, end vi har.
På den anden side er kundegrundlaget massivt, og konstant stigende. Det er også et marked, hvor de, der kan udskille sig fra mængden og/eller levere konstante kvalitetsprodukter, har alle tiders mulighed for at slå igennem.

Opstartsbudget

Selv om vi forsøger at holde vores budget på så lavt et niveau som muligt, så er der stadig nogle omkostninger, som vi er nødt til at forholde os til. Når vi vil udvikle vores spil til iOS, så kræver det, at vi indkøber nogle Apple produkter til test, samt en Mac til at kompilere spillet på. Det koster desuden 535kr. at blive registreret som iOS developer.

Etableringsbudget	
UDGIFTER	**Kr.**
Kontorinventar:	30.350,00
IT (computer, printer, netværk mv.)	14.197,00
Unity Pro + iOS	16.153,00
Vareindkøb:	4.000,00
Assets, grafik	1.000,00
Assets, lyd	2.000,00
Andet: Unity assets	1.000,00
Rådgivere:	0,00
Advokat	0,00
Revisor	0,00
Andet:	0,00
Markedsføring:	100,00
Hjemmeside	100,00
Webshop	0,00
Andet:	0,00
Andre udgifter:	535,00
Andet: iOS developer program licens	535,00
Udgifter i alt:	**34.985,00**

TABEL 10 - ETABLERINGSBUDGET

Dertil kommer Unity Pro og Unity iOS Pro, samt eventuelle assets som vi ikke kan eller vil producere selv til vores spil. For eksempel kan det blive nødvendigt at købe os til et soundtrack til spillet. Det er også vores hensigt at indkøbe et plugin, som kan håndtere det tunge arbejde i forhold til vores IAP.

Det skal også påregnes at et mindre beløb kommer til at gå til at oprette en firmahjemmeside, men selve opsætningen af denne, samt hvad vi skal bruge af markedsføringsmateriale, har vi til hensigt selv at stå for. Vi laver ikke markedsføring for virksomheden som sådan, men i stedet på et givent projekt.

Da vi agter at gå til Nordjysk Iværksætter Netværk (NIN) for advokat og revisor rådgivning, har vi i første omgang sat udgifterne til disse til nul. Således bliver eventuel fremtidig rådgivning på de felter en del af virksomhedens drift budget. På denne måde ender vi med et relativt lavt beløb som vi selv kan finansiere. Drift budgettet vil vi ikke yderligere beskæftige os med i forbindelse med denne rapport, da posterne i høj grad kommer til at bero på informationerne som gives af NIN.

Finansiering

På grund af den lave indgangsbarriere i forhold til opstartsbudgettet, vil vi i første omgang forsøge at klare os ved at bruge vores egne penge. Vi kan lave en prototype af vores første spil for omkring nul kroner, da de værktøjer vi skal bruge findes i gratis versioner. Faktisk behøver vi ikke starte virksomheden overhovedet før vores prototype er ved at være klar til fremvisning.
Vi kan her bruge vores egne penge, men af åbenlyse årsager ville vi hellere undgå det. Da vores økonomiske udgangspunkt er lidt svagt, har vi ikke den store tiltro til at en bankforbindelse ville være en mulighed i første omgang. Det samme gør sig gældende for venture kapital. I øvrigt er der altid en pris forbundet med det sidstnævnte, og som udgangspunkt vil vi gerne bevare vores frihed. Det lægger også bedre i tråd med vores strategi om at vækste firmaet på en organisk måde.

Det vi har til hensigt at gøre, er at søge forskellige fonde og offentlig støtte. I forhold til opstart af virksomheden vil vi f.eks. gøre god brug af den gratis rådgivning, vi kan få af Nordjysk Iværksætter Netværk [30]. Derudover er der en meget lang række forskellige tilskud man kan søge, hvilket er et område, som vi vil undersøgte ganske nøje [31] på et senere tidspunkt.

Virksomhedsform

Da virksomheden skal startes op med så lille en startkapital som muligt, udelukkes virksomhedsformerne Aktieselskab (A/S) og Anpartsselskab(Aps), der ville kræve indskudsværdi for hhv. 500.000 og 50.000.

Mulighederne er herefter Enkeltmandsvirksomhed, Interessentskab(I/S) og Iværksætterselskab(IVS). Enkeltmandsselskaber og I/S´er kan startes uden videre, men ejeren (eller ejerne, hvis der er tale om et I/S) hæfter personligt for virksomhedens eventuelle gæld, og kan på den måde give de involverede massive kvaler i det tilfælde, at virksomheden går konkurs.

Tilbage er der IVS [32], der er en ny forretningsform etableret i januar 2014. Et IVS har den samme struktur som et anpartsselskab. Blot kan det startes op med et indskud på 1 kr. hvilket er et beløb, som vi i fællesskab er i stand til mønstre. Kravet er så at man opsparer penge i virksomheden indtil man runder de 50.000 kr., som det koster at konvertere til et Aps.

Et IVS er et selvstændigt retsobjekt, hvilket vil sige, at ejerne ikke hæfter personligt for eventuel gæld. Argumentet imod opstart af et IVS er, at det ikke nødvendigvis vil inspirere en finansiel institution til at investere i virksomheden, når ejerne ikke har noget på spil. Men på den anden side kan vi heller ikke på anden vis fremskaffe nogen form for sikkerhed, hvorfor argumentet bliver irrelevant.
Altså er det hensigten at starte virksomheden som et IVS.

Forretningsgrundlag
Der er en række forskellige måder et spilfirma kan tjene penge på sine produkter på, i kapitlet herunder gennemgås de forskellige, samt under hvilke omstændigheder de enkelte metoder er velegnede.

Direkte Salg
Direkte salg er den klassiske metode, hvor man sælger produktet direkte til forbrugeren for et fast beløb. Det direkte salg gør sig godt på PC og konsol-markedet, hvor forbrugeren altid har været vant til, at det er sådan tingene gøres.
Denne metode kan kombineres med de andre, men det kan være svært at forklare køberen, at han skal betale mere efterfølgende. Det er dog ikke uhørt at det forekommer, MMORPG´er som World of Warcraft har f.eks. en abonnementsmodel tilknyttet, mens f.eks Dead Space 3 har IAP tilknyttet.

Reklamer
I stedet for at tage penge direkte fra produktet, kan man indsætte reklamer, hvorfra man kan generere en kontinuerlig indtægt. For at denne metode kan være rentabel kræves det, at der er mange mennesker, der bruger ens spil, da summen, man får for enten vist reklame eller klikket-på reklame, er meget lille. Derfor bruges reklame-metoden ofte på mobil- og webbaserede platforme, da spillet så er tilgængeligt for et meget stort marked. Samtidigt er reklamefinansierede spil meget ofte gratis i anskaffelse, hvilket gør det let for kunden at anskaffe sig spillet.
Reklame-metoden bruges ofte i ”demo”-udgaver af spil. I den forbindelse er håbet så, at spilleren skal blive så glad for spillet, at han køber den fulde version. Eksempler på dette er f.eks. Angry Birds Lite og Wordfeud.

Free-To-Play / Mikrotransaktioner
Efter direkte salg er Free-To-Play den måske mest udbredte indtjeningsstrategi for spil i øjeblikket. Ved Free-To-Play forståes det, at selve spillet er gratis for brugeren at anskaffe sig. I stedet genereres virksomhedens indtjening ved at spilleren kan købe sig til visse goder via mikrotransaktioner. Ofte er der tale om en virtuel møntfod, som spilleren så kan bruge i spillets butikker.
Metoden blev først anvendt i tidlige onlinespil, og har siden bredt sig til alle typer, og størrelser, af spil. Et af de mest kendte eksempler er vel Facebookspillet Farmville, hvor spillet er gratis, men hvor der så er rig mulighed for at spendere efterfølgende.
Denne metode har givet anledning til en del debat, da der er mange måder at organisere et Free-to-Play-spil på, og nogle af dem er decideret forbrugerfjendske [33].
Der er en række negative eksempler, men omvendt kan det også være en måde hvorpå virksomheden kan knytte endnu tættere bånd til sine kunder, hvis det gøres rigtigt. Se f.eks. Path of Exile, Loadout, League of Legends, World of Tanks

Abonnementsordning
En ordning hvor forbrugeren betaler et månedlig beløb for adgang til spillet. Det har tidligere været den optimale måde at holde de store multiplayer-spil i live på, men tiden er tilsyneladende ved at løbe fra den model. I hvert fald ses det igen og igen hvordan nye MMO´er starter med en abonnementsmodel, men kort tid efter opgiver ævred og går Free-To-Play. Se f.eks. Star Wars – The Old Republic og Age of Conan.

Kontrakt
En helt almindelig kilde til indtægt for mange studier, er at indgå en kontrakt med en kunde om at producere et spil. Under de omstændigheder aftales prisen med kunden fra starten, samt eventuelle bods-

eller bonus-ordninger, hvorefter spiludvikleren går i gang med at løse den aftalte opgave. Under de omstændigheder er det ikke udelukkende spiludvikleren der har opgaven med at promovere spillet, og ofte er det netop promoveringen der er udgiverens opgave i samspillet. Der findes en lang række både store og mindre udgivere, der enten ejer, eller laver enkeltstående kontrakter med spilfirmaer. Blandt de mest kendte er: Electronic Arts, Activision og 2K, som alle udgiver til alle tænkelige platforme.
Mindre samarbejdspartnere end dem kunne dog også gøre det. Der findes eksempler på små studier som laver små spil for firmaer til deres markedsføring [34] [35].

Merchandise

Selvom salget af spil formentligt altid vil være den primære indtægtskilde for et spilfirma, så skal man ikke underkende af salg af merchandise. Værdien er tofoldig i og med, at det både er en potentiel stor indtægtskilde til virksomheden, og en måde at styrke virksomheden og/eller det enkelte spils brand.

Opsummering

Som virksomhed kan man godt vælge at specialisere sig i en specifik indtjeningsmetode, og det kan også godt vise sig at være fornuftigt. Især i forbindelse med Free-To-Play, som dækker over en myriade af muligheder.

Et sidste argument imod direkte salg, og for Free-To-Play og reklamebaserede modeller, er at visse platforme har store problemer med piratkopiering. Et kopieret eksemplar af et produkt der kun har direkte salg som strategi, er mistet indtjening for udvikleren. Men hvis indtjeningsmodellen på produktet er af en karakter hvor det er lige meget om produktet i sig selv er gratis, så er piratkopiering udelukkende en fordel for udvikleren.

Det er ikke relevant for os at spekulere i abonnementsmodeller på nuværende tidspunkt. Den generelle tendens i industrien er, at gå væk fra den model og i retning af modeller med mikrotransaktioner. Hvis det en dag vender, så forbrugerne hellere vil betale premium priser for det fulde produkt, så er det en diskussion der kan tages op igen.

Overordnet set er over de seneste par år blevet skabt et konsensus omkring hvilke metoder der matcher hvilke typer spil, og det er vores holdning, at vi er bedst tjent med at følge med strømmen på det område. Det vil altså sige, at vi ikke fra starten vil fastlægge os på en bestemt type forretningsmodel, men i stedet vælge en model der passer til projektet.
Til det konkrete projekt vil det sige enten direkte salg, gratis med reklamer og/eller gratis med mikrotransaktioner.

Udgangspunktet for virksomheden er, at der er to programmører og en grafiker til rådighed på deltid. Alle er i princippet ulønnede, da evt. løn hænger direkte sammen med den omsætning som kan genereres. Ingen af de tre iværksættere har før udgivet et spil, eller noget andet produkt. Ligeledes er der heller ingen erfaring med iværksætteri blandt opstarterne.
Alle involverede har arbejdet på spilprojekter på fritidsbasis i et par år, og har derfor en grundviden om hvordan et spil skal skrues sammen. Desuden er alle involverede vokset op med at spille spil af alle mulige slags, på alle tænkelige platforme, hvilket give en solid platform hvorpå vi kan diskutere forskellige designløsninger, indtjeningsstrategier osv.
Der er ingen reel organisationsstruktur i virksomheden, da udgangspunktet er at grundlæggerne er ligestillede, men med en grovopdeling af opgaverne, hvor vores grafiker er hovedansvarlig for at tegne vores projekters vision, mens det er programmørernes primære opgave at sikre at den vision kommer til live.

I forhold til opgaver som idegenerering, design, historiefortælling mv. Altså de aspekter, som ikke direkte omhandler produktionen af assets, så betragter vi det som en fælles opgave.
Det økonomiske udgangspunkt for virksomheden er, at vi ikke har egentlig kapital at starte op for.

Derfor er vi for nuværende tvunget til at arbejde deltid, arbejde hjemmefra, og arbejde med gratis-værktøj. Der er heller intet markedsføringsbudget, eller mulighed for at deltage i messer af nogen art.

SWOT Analyse

Hvis man tager de ovenstående faktorer, og vurderinger i betragtning kan samle det i en SWOT-model som set i Tabel 11. Vores styrker består primært i de mennesker, vi er, og den erfaring med spilbranchen, som vi besidder. Dertil kommer, at vi er uddannede til at kunne håndtere de tekniske aspekter af udviklingsarbejdet. Hvis man kombinerer det med mulighederne, som kort opsummeret er, at der er et kæmpe marked at sætte kløerne i for dem der ved hvad kvalitet er, og hvad brugernes forventningsniveau ligger på. Så mener vi, at vi har alle tiders mulighed for at starte et sundt firma op.
Vores svage sider er primært bundet op på en svag opstartsøkonomi, men det kan vi udligne al den stund, at indgangsbarrieren på markedet er så lav. Se evt. Tabel 10.

Vi starter firmaet som en sidebeskæftigelse, derfor har vi ingen trusler i forhold til at gå fallit som følge af lønudgifter mv. De eneste reelle trusler fra markedet, er piratkopiering, som er en allestedsnærværende trussel for industrien i det hele taget. Og så det, at vores produkter risikerer at forsvinde i mængden, og aldrig bliver set af nogen.

Da vi ikke har en kapital som kan bruges på markedsføring, trækker vi igen veksler på vores personligheder, den kreative kraft. Viral markedsføring er altid en mulighed for at blive set af mange, og om den slags er succesfuldt eller ej beror altid på de personligheder der designer kampagnen. Dog mener vi, at vores bedste chance for at undgå glemslen, er konsekvent at levere gode spil af høj kvalitet.

Styrker	**Svagheder**
- Fleksibilitet - Virksomhedens kreative kraft - Teknisk knowhow - God indsigt i spilbranchen generelt	- Mangel på kapital - Manglende erfaring med virksomhedsdrift - Svagt netværk i branchen
Muligheder	**Trusler**
- Stort marked med mange selvstændige segmenter - Lav indgangsbarriere - Mange konkurrenter leverer lavkvalitetsprodukter	- Vores produkter kan meget let drukne i mængden - Piratkopiering

TABEL 11 – SWOT

Strategiplan

I første omgang er det virksomhedens mål, at nå en størrelse hvor det kan fuldtidsbeskæftige ejerne. Vi vil ikke lægge os fast på om vi er et firma, der kun udvikler til App-markedet, eller PC, eller konsol. Vores fleksibilitet er vigtig. Derfor har vi også valgt Unity3d som udviklingsplatform, da den netop understøtter mange formater, uden at vi i virksomheden skal starte forfra med nye værktøjer for hvert projekt.

Det er ikke vores holdning, at der er et reelt loft over hvor meget en spilvirksomhed kan vækste, hvis den først kommer ud over hobbyniveau.

Virksomheden om et år:

Om et år har virksomheden udgivet mindst et spil og beskæftiger en mand på fuld tid. De(t) udgivne spil udgør virksomhedens portefølje, og med dem i hånden vil vi forsøge at finde investorer, således at fremtidige projekter kan udvikles med fuldt fokus. Derudover har vi fået alle dele af håndværket ind under huden fra præ-produktionen og til udgivelse.

Virksomheden om 2 år:

Om to år beskæftiger virksomheden minimum ejerne på fuld tid. Meningen er, at virksomheden skal skaleres på en organisk måde, hvor vi altid er på et niveau, hvor vi ikke tager projekter, som er for store. Vi bygger os selv op, og vi tager projekter i den størrelse, der passer for konstant udvikling.

Målsætning & Handlingsplan

- Om 3 mdr. Skal der være en prototype af platforms-delen klar. (12/6/2014)
- Om 4 mdr. Skal virksomheden have firmahjemmeside og pressemateriale klar. (11/7/2014)
- Om 5 mdr. Skal der være en prototype af showdown-delen klar (25/7/2014)
- Om 7 mdr. Skal spillet være i beta-tilstand. Dvs. færdigt, men med plads til fejlrettelser
- Om senest 8 mdr. skal virksomheden have selvudgivet første spil.
- Om senest 8 mdr. skal virksomheden have kontakt til investorer.
- Om et år skal virksomheden være fuldtidsbeskæftigelse for mindst en af ejerne.
- Om et år skal virksomheden have udviklet og udgivet mindst to spil, eller porteret første spil til flere platforme.

Planen fra projektstart, og frem til showdown-milepælen fremgår af bilagslisten. Ligesom andre af vores design dokumenter, er projektplanen et levende dokument. Det vil sige, at de milepæls-datoer som er aftalt, og indført, ikke kan ændres. Men projekt-datoer er ikke fastlåst fra start til slut.

Projektvalg

Det spil, vi har udvalgt som vores første projekt, er en hybrid imellem en Endless Runner og et almindeligt bane-baseret platformspil. Det er to genrer, som vi kan se gør sig godt på de mobile platforme, og som, jf. vores markedsanalyse, rammer en bred målgruppe.

Af samme grund findes, der naturligvis en række direkte konkurrenter, særligt spil som Jetpack Joyride og TempleRun, som vi skal positionerer os i forhold til.

Det gør vi på fire punkter:

1. Spillet er ikke ”endless”, men derimod delt op i kortere baner, hvilket giver os mulighed for at etablere en decideret fortælling.

2. Spillet adskiller sig fra typiske platformspil ved at have en proceduralt genereret banestruktur. Altså er vores baner forskellige fra spil til spil.

3. Spillet er bygget op med en blanding af 2D og 3D (Typisk betegnet 2.5D), hvilket er unikt for genren, hvor de konkurrenter vi sammenligner os direkte med enten er helt 3D eller helt 2D.

4. Spillet indeholder bosskampe i form af et duel system, hvor vores protagonist i bedste westernstil står ansigt til ansigt med de primære antagonister. Generelt er bosskampe ikke et element, der indgår i de spil vi anser som direkte konkurrenter.

Med tanke på virksomhedens relativt begrænsede ressourcer giver det god mening at lave så meget som muligt i spillet proceduralt genereret. Når vi bygger en bane op af generiske byggeklodser, behøver vi ikke bruge store mængder tid på at lave bane-design. I stedet skal vi sikre, at vores algoritme ikke sætter spilleren i en umulig situation, og at sværhedsgraden er stigende. Begge dele er opgaver, som skal løses via kode og almindelig play test.

Play test er nemmere at udføre, hvis spillets game mechanics er let gennemskuelige. Ved at lave et spil med et featuresæt, som må anses for at være casual, gør vi testfasen lettere for os selv, samtidig med vi tilgodeser vores brugeres behov.

Vores forretningsmodel for dette projekt bliver en blanding af direkte salg, reklamer og IAP. Jvf konklusionerne i markedsanalysen vil vi sælge spillet til en fast pris på iPhone, mens en senere Android version vil være gratis for brugeren at hente, men de må til gengæld leve med reklamer. I begge versioner vil vi implementere en butik, hvorfra vi for mindre beløb kan sælge nye kampagner, og alternative skins til protagonisten.

Det passer med vores nuværende situation, hvor ingen kender os, og vi ikke har nogle tidligere produkter at bryste os af. Der er det vigtigt, at vi kan nå så mange mennesker som overhovedet muligt.

Alle de oplysninger, vi har brug for i forhold til spillets design, featuresæt, stil mv., er samlet i et såkaldt Game Design Dokument (GDD), som kan ses i dets helhed i bilagslisten sidst i rapporten. Et GDD er et levende dokument, som vi tilretter i løbet af produktionen. Det er hovedløs gerning at skrive alt ned i detaljer fra starten, da enhver feature på et givent tidspunkt kan vise sig overflødig, upraktisk, eller at brugertests viser, at spilleren hader den.

Projektbudget

Med henblik på antallet af potentielle kunder, samt vores generelle målsætning for virksomheden, nedfælder vi et mål om at nå en million i indtægt før skat for dette projekt.
I det følgende gennemgår vi, hvad der skal til for at nå det mål med et udgangspunk i en reklamebaseret app, en der kun anvender IAP, og en for direkte salg.
Da vi anvender en kombination af de tre, kan forventer vi et lidt lavere antal hentninger kan gøre det, for

at nå det samlede mål. Når der tages udgangspunkt i tallene for hver enkelt type app, så er det fordi det er umuligt at forudsige hvilken kombination af hentninger fordelt på Android og iPhone som fører os frem til målet.

Reklamebaseret

Reklameindtægter er baseret på antal visninger, og beregnes ofte via eCPM [36]. Det står for "effective cost per thousand impressions" (mille = tusind), og er en industristandard I forbindelse med reklame indtægter. eCPM beregnes ved at dividere den totale indtjening med antal visninger i tusinde.

Der findes ikke et entydigt beløb som man kan forvente at få udbetalt, og den eneste sikre måde at finde det svar på er at sende et produkt ud i verden. Det billede der tegner sig når man undersøger området er dog, at der kan være stor forskel i udbetaling alt efter hvilket netværk man er tilkoblet [37]. Derfor kan man med fordel benytte flere forskellige reklamenetværk.

Et andet argument for at være tilknyttet flere reklamenetværk er, at man på den måde øger produktets fillrate. Fillrate [38], i reklame sammenhæng refererer til resultatet af antal reklamer leveret fra reklamenetværk divideret med antallet af forespørgsler på reklamer. Altså, hvor ofte reklamenetværket kan levere en reklame når den givne app har brug for en visning. Hvis der, af den ene eller anden grund, ikke kan leveres en reklame når spillet forespørger det, så er det en tabt visning.

Der er tegn der peger på, at eCPM for mobilspil ligger mellem 5,5 og 55kr [39]. I Tabel 12 tager vi udgangspunkt i en middelpris på 27kr. Hvis vi samtidigt antager at hver bruger der henter spillet ser 3 reklamer, så skal vi have sammenlagt 13 mio. brugere for at projektet opnår målsætningen.

Antal	eCPM(DKK)	Visning pr. download	Før skat	Platform
6.500.000	27	3	526.500	App Store
6.500.000	27	3	526.500	Google Play
			1.053.000	**I alt**

TABEL 12 - 1 MIO. VIA REKLAMEVISNINGER

IAP

Hvis man i stedet tager udgangspunkt i IAP, så er indtjeningsgrundlaget baseret på antallet af brugere der rent faktisk køber noget. Igen er det naturligvis umuligt præcist at fastslå hvor mange der vil benytte sig af den mulighed, men forsigtige estimater indikerer en konversionsrate på 2,5%.

For at kunne lave en konkret beregning forudsættes det her, at vi sælger en ny "World" til vores spil til 10kr. For at vi kan omsætte for en million skal vi have 4 mio. brugere.

Antal	CR(%)	Pris(DKK)	Før skat	Platform
2.000.000	0,025	10	500.000	App Store
2.000.000	0,025	10	500.000	Google Play
			1.000.000	**I alt**

TABEL 13 - 1 MIO. VIA IAP

Direkte salg

Ved direkte salg skal vi have væsentligt færre brugere for at kunne nå vores mål. Sammenlagt 240.000, skal der til. Til gengæld er deres indgangsbarriere så meget desto højere, da der jo netop er en konkret pris forbundet ved at hente spillet. I udregningen her, er prisen sat til det lavest mulig beløb. Det er helt igennem muligt at hæve prisen, men ved samme lejlighed øger man også barrieren for brugeren. Prisen er altså sat i tråd med vores holdning om, at det handler om at blive set mere end noget andet. Både i forhold til iOS og Android gælder i øvrigt det, at butikken tager 30% af hvert salg.

Antal	Pris(DKK)	Cut	Før skat	Platform
120.000	6	0,7	504.000	App Store
120.000	6	0,7	504.000	Google Play
			1.008.000	**I alt**

TABEL 14 - 1. MIO I DIREKTE SALG

Tallene kan godt, især i forhold til de IAP- og reklamebaserede beregninger, virke voldsomme, men det er væsentligt at holde sig for øje hvor stor den samlede brugerbase er. (Tabel 6 - Antal hentede apps, fordelt på salgsplatforme.)

I det lys mener vi ikke, at det er det ikke et urimeligt mål som er opstillet.

Konklusion på forundersøgelse

I forbindelse med udarbejdningen af denne rapport kommer vi omkring en lang række teknologier og metoder. For at kunne begrænse omfanget af rapporten til det tilladte, er der nogle aspekter, som vi enten helt udelader, eller kun berører overfladisk. I løbet af forundersøgelsen har vi undersøgt hvilke elementer som vi hovedsageligt vil beskæftige os med.

Det er besluttet, at virksomheden startes som et iværksætterselskab, hvis kernekompetence er at producere spil ved brug af Unity3D. Virksomheden besidder fra starten en lang række af de kompetencer som skal til for at kunne påbegynde produktionen. Hvor der er kompetencemangler er der i stedet en plan for at fremskaffe assets via tredjepart.

Markedsanalysen godtgør, at der helt generelt findes et solidt brugergrundlag som spil kan sælges til, og at mobilplatformene er et oplagt valg til at udgive det første spil til.

Derfor er vores første projekt også netop til mobilenheder, og er designet med et scope der passer sig for den type spil. Det er yderligere bestemt, at den endelige version udvikles til iOS som lead platform, mens vores første prototype af logistikhensyn blot skal køre på PC og Android.

I henhold til virksomhedens mål for vækst, er det skønnet at projektet skal generere en million kroner i indtægt før skat for at betragtes som en succes.

Prototype fokus

Det er af flere grunde ikke muligt færdiggøre vores spil inden for projektets deadline. Det skyldes primært, at produktionen kræver en lang række forskellige assets for at kunne færdiggøres. Vi har derfor kun til hensigt at have en prototype klar til aflevering ved projektets afslutning. Det vil sige, at vi kun bruger placeholder grafik og helt udelader lyd, samt at vi kun laver en enkelt spilbar bane færdig. Desuden indeholder prototypen en funktionel hovedmenu, hvorfra spillet kan startes og afsluttes, samt placeholder undermenuer.

Selvom spillet skal have reklamer og mikrotransaktioner indbygget, så er det heller ikke elementer som kommer til at figurere i prototypen. Den uddannelsesmæssige værdi vi kunne opnå ved at implementere det er begrænset, da det er et af de områder hvor vi vurderer, at det er bedre at benytte sig af tredjeparts plugins.

Det sidste forbehold angår det faktum, at vi ikke udvikler prototypen til iOS, men Android. Det gør vi fordi vi ikke umiddelbart råder over de tekniske virkemidler der skal til for at udvikle til iOS.

I stedet fokuserer vi på at opsætte en spilbar bane, der indeholder så mange forskellige elementer som muligt. For at opfylde kravene til ”spilbar”, skal der som minimum være platforme at bevæge sig på, og en figur at bevæge. Derudover skal der være et defineret start- og slutpunkt. Ligeledes skal det være muligt for spilleren at fejle, og demoen skal kunne håndtere dette.

Designdokumentet, som findes i bilagslisten, specificerer, at platformene skal genereres i tilfældig rækkefølge for hver gang banen starter. Tillige er det et af de punkter hvor vi har mulighed for at kunne differentiere os fra andre spil af samme type. Derfor tillægger vi det særlig fokus allerede i denne første prototype.

Det andet centrale element i spillet er selve protagonisten, som vi herefter også refererer til ved hans navn: Mojo. Der er en lang række forskellige elementer forbundet med oprettelsen af Mojo, hvilket sammen med det helt essentielle i at have en protagonist, gør det til et naturligt fokuspunkt.

Medmindre et projekt udelukkende baserer sig på flerspiller interaktion, så er kunstig intelligens (herefter benævnt AI) en af hovedelementerne i et spil. Al handling der foretages af andre karakterer end protagonisten kræver AI implementering. Fjendtlige karakterer er et af de centrale elementer som kan bidrage til at øge spillets værdi. Samtidigt kan det let blive en kompliceret affære, derfor anser vi det som væsentligt at prototype fra starten.

Et andet fokuspunkt, som er mindre synlig i selve prototypen, men absolut væsentligt fra et firmaperspektiv, er vore evne til at arbejde med Unity. Derfor gennemgår vi i detaljer de overordnede elementer i Unity, samt brugergrænsefladen. Ligeledes beskrives den udviklingsmodel som Unity er bygget op omkring, og som adskiller sig fra objekt orienterede model som virksomhedens udviklere tidligere har arbejdet med.

Design

Udviklingsproces

En af de ganske få reelle fordele, der er ved at være et lille firma, frem for et stort, er den øgede mulighed for at være fleksibel. Det gælder i alle aspekter af virksomhedsdriften, men er især en force i forbindelse med udviklingsarbejdet.

Spiludvikling er i sin natur en organisk proces, hvor det hele vejen igennem et projektforløb kan give mening at ændre i såvel den teknologi som anvendes til udviklingen, samt spillets design i øvrigt. Det kan altså på et hvilket som helst tidspunkt i udviklingen blive nødvendigt at udføre prototype arbejde på dette eller hint.

Vi er også nødt til at forholde os til, at firmaet ikke fra starten besidder den nødvendige erfaring til at kunne detailplanlægge forløbet på forhånd.

Derfor er det vigtigt at vi vælger en udviklingsproces, som understøtter fleksibilitet. Derfor udelukker vi fra starten at udvikle efter traditionelle plandrevne procesmodeller. Plandrevne projekter gør sig bedst i større virksomheder med specialiserede afdelinger. Et andet karakteristika ved plandreven udvikling er, at projektet planlægges i detaljer fra starten. Hvilket altså er det diametralt modsatte hvad vi forventer at kunne, og ønsker, at gøre.

Det giver langt mere mening at benytte en agil udviklingsmetode hvor det netop er fleksibiliteten der er i højsædet. Vores udgangspunkt er derfor at vi bruger Scrum til at styre vores projekt. Scrum i sig selv foreskriver ikke noget omkring den praktiske side af softwareudvilkling, hvorfor vi anvender eXtreme Programming som rettesnor i forbindelse med den regulære udvikling [40].

Scrum

For at Scrum kan bruges til vores formål, er det blevet besluttet at lave nogle fundamentale ændringer. Medmindre andet er angivet, benytter vi Scrum og XP som de er beskrevet i ”The Scrum Guide” [41].

Den væsentligste ændring, vi har foretaget, er, at vi accepterer, en story kan oprettes og tages ind midt under et sprint. Det er ellers en af de regler som anses for fastlagte i Scrum, således at det undgås for enhver pris. Når vi alligevel har valgt at fravige den regel, så er det fordi vi ikke vil holdes tilbage af manglende erfaring. Ved oprettelsen af vores product backlog forsøger vi naturligvis at opdele alle aspekter af spillet i stories. Men vi kan ikke garantere, at der ikke vil opstå en situation hvor vi mangler en story for at kunne fortsætte produktionen. Under de omstændigheder vil vi ikke sidde på vores hænder indtil sprintet afsluttes.

Fordi vi forventer uforudsete stories, regulære såvel som spikes, udvælger vi et lavere antal story points til et sprint end det vi egentligt forventer at kunne nå. Fordelen er her, at såfremt vi får behov for at oprette en ny story for at kunne fortsætte produktionen, så har vi bedre tid til det. Hvis det behov ikke opstår, så har vi muligheden for at udfylde tiden ved at tage en anden story ind fra product backloggen.

Der er et par ulemper ved denne fremgangsmåde; først og fremmest, at det gør vores overordnede projekttidsestimering mere uprækcis. Fordi vi afsætter mere tid end vi forventer, er tanken at estimeringen gerne skulle forskubbe sig i retning af at projektet tager kortere tid end forventet. På længere sigt er det nu alligevel ikke en holdbar løsning, men præcis estimering er et spørgsmål om erfaring, og for hvert sprint, og hvert projekt, skulle vi gerne blive mere præcise, således at vi kan udfase denne regel.

Måden vi opdeler vores stories på, er ved at sige: Èn feature = Èn story. F.eks. er ”Protagonist, Løb” en story for sig selv. Den involverer at finde en model, få lavet en character controller, påhænge et kamera, scripte brugerinput mv, og tilføje en animation.

Protagonist, Hop-story består af mange af de samme trin, så en del af arbejde ville være klaret hvis den bliver løst efter ”Protagonist, Løb”-story. Ved at opdele stories på den måde vil vi et langt stykke hen af vejen kunne opnå Scrum-tesen om at stories skal kunne løses i tilfældig orden.

Det er ikke hensigtsmæssigt at sikre den uafhængighed 100%, da det blot vil betyde at visse stories ville blive urimeligt omfattende. Hvis f.eks. der blev oprettet en story for ”Gem Spil”, og vi tog den ind først, så ville det medføre at en bane og protagonisten blev oprettet først som en del af den story, for at det ville give mening. Altså anvender vi sund fornuft både ved oprettelsen af stories, og ved sprintplanlægningen hvor de stories det næste sprint byder på, skal udvælges.

Fordi vi arbejder på vores eget interne produkt har vi heller ikke en product owner som sådan. I stedet refererer vi til vores Game Design Dokument, som alle projektmedlemmer vedligeholder igennem konsensus. I forhold til Scrum master-rollen, så er det ikke en titel vi benytter som sådan, men funktionen ligger primært hos et enkelt teammedlem. Da vi ikke behøver at skulle ”forsvare” os imod en øvre ledelse, eller andre afdelinger som trækker i forskellige retninger, og samtidigt ikke har en udefrakommende kunde, består Scrum masterens rolle primært i at sikre at Daily Scrums og andre aktiviteter bliver holdt som aftalt.

Der blev valgt ikke at gøre brug af yderligere metoder, såsom Kanban, Selvom Kanbans praksisser fint ville fungere sammen med brugen af Scrum, så har vi vurderet, at der ikke er behov for yderligere strukturering i arbejdsprocessen, da udviklingsteamet er så lille, at vi godt kan se f.eks. flaskehalse, uden brug af Kanban.

Extreme Programming

I forhold til udviklingen af scripts benytter vi os af XP næsten, som det er beskrevet i manifestet [42]. En undtagelse er ved konceptet om parprogrammering, hvis funktion vi har erstattet med et almindeligt code review. Det er en beslutning, som er truffet af flere grunde. For det første har vi ikke ressourcerne til at kunne gennemføre parprogrammering på en fornuftig måde. Det kræver større skærme, og større arbejdsstationer i det hele taget, end vi har til rådighed. Den anden væsentlige årsag er, at vi mener at miste en del af vores fleksibilitet, hvis vi binder to teammedlemmer op på den samme opgave, hver gang vi skal producere et script. Det er i den sammenhæng væsentligt at erindre, at selve scripting-arbejdet kun er en del af vores samlede arbejdsbyrde.

Introduktion til Komponentbaseret udvikling

Udviklingen af vores spil beror i høj grad på komponentbaseret udvikling, da Unity er bygget op omkring komponenter.

Komponentbaseret udvikling ligesom objekt orienteret beror på objekter. Forskellen mellem de to ligger i den måde man organiserer objekterne på. I klassisk Objekt Orienteret Programmering (OOP) arbejder man med en klassestruktur, hvor man nedarver efter en X ***er en*** Y-kategorisering. Skulle man programmere vores spil fra bunden, og ud fra et OOP princip, ville man sige, at Mojo ***er et*** GameObjekt, og Mojo ***er en*** bevægelig figur etc. Fordi det giver logisk mening.

Arbejder man i stedet arbejder efter en komponentbaseret logik, så ville man sige at X ***har en*** Y-opførsel. Mojo ***har en*** Meshrenderer, og Mojo ***har en*** Rigidbody. En komponent kan således siges at være en *opførsel,* som man kan tildele et objekt.

Man kan godt designe en række underlæggende klasser der er bygget op efter traditionelle OOP principper, for så at lave et kodelag der implementerer MonoBehaviour, som er Unitys scripting API, ovenpå. Det kan være en god løsning hvis man laver et spil som skal lave tungere beregninger. F.eks. et rollespil eller et simulationsspil. Den slags er der ikke behov for i vores projekt, og derfor kan vi se bort fra objekt orienteret programmering.

Unity kommer med en lang række forskellige komponenter indbygget, hvilket fritager os fra en lang række opgaver, vi behøver for eksempel ikke spekulere på at lave kollisionssystemer, eller kameraer, fordi de allerede er lavet for os. Vores opgave er at levere de scripts, som i sig selv er komponenter, som skal udnytte den funktionalitet som ligger i de andre komponenter som benyttes i spillet.
Der er en oplagt mulighed for at sikre lav kobling i forbindelse med udførelsen af disse, da et script uden videre kan bruges i et andet projekt senere hen. Så jo mere generisk vi kan udføre den opgave, jo større er muligheden for genbrug.

Kunstig intelligens

Vores designdokument specificerer, at der skal være forskellige typer af NPC´er i spillet. Disse NPC´er skal kunne opføre sig på en fornuftig måde, både i forhold til deres interaktion med Mojo, men også i forhold til deres egen interaktion med spillet. Den opførsel refererer vi til som NPC´ens AI, dens kunstige intelligens.

For at opnå AI implementerer vi en tilstandsmaskine som også refereres til som en FSM (Finite State Machine). Ved denne metode identificerer vi hvilke forskellige tilstande den givne AI skal kunne befinde sig i, samt de(n) faktorer som skal udløse et skift fra en tilstand til en anden. Derefter er det blot et spørgsmål om at sikre, at der kan skiftes imellem tilstandene som vi ønsker det.

Grunden til, at vi vælger denne metode til AI er, at det er relativt let at implementere. En FSM er god, når der er tale om en relativt simpel AI, med et begrænset antal tilstande. Hvis AI´en derimod indeholder mange tilstande, eller fra starten er uklart defineret så man skal tilbage og rette i den, kan det gå markant ud over overskueligheden. Det samme gør sig gældende, hvis man indlejrer tilstandsmaskiner i hinanden. Et eksempel kunne være en AI, der har en tilstandsmaskine til at håndtere ”fredelige” gøremål, mens den implementerer en helt ny, når den indtræder i kamp.

Grafiske designhensyn

Når der udvikles til mobile plaforme er der adskillige hensyn der skal tages, især i forhold til de enkelte enheders hardware. På trods af de fremskridt som er gjort, og som til stadighed, gøres på mobilplatforme, så er hardwaren stadig ganske langt fra PC eller konsolniveau. Derfor er man som udvikler nødt til at imødegå disse svagheder, og designe sit produkt på de vilkår som enhederne stiller.

Det er især i forhold til den grafiske ydeevne, at der er grund til at være påpasselig, det det typisk er det mest ressourcekrævende aspekt i et spil. Hvis man laver sit spil, uden at have fokus på ydeevne, medfører det en massiv risiko for projektet. I det følgende kapitel ser vi på hvordan vi kan minimere ressourcekravene til vores grafiske assets.

Mens computere og konsoller har rig mulighed for at blive udrustet med kraftfulde komponenter, så er mobilitet og kompakthed fokuspunktet for smartphones og tablets.

Dette stiller høje krav til hardwaren, da det fysisk begrænser pladsen til ydeevnen. Samtidig er temperaturen et element som skal indberegnes, da hardwaren kræver nedkøling [43]. Enten i form af blæsere eller vandkøling. Dette betyder, at det er balancen imellem computerkraft, nedkøling og fysisk plads, som udgør en enheds arbejdskraft.

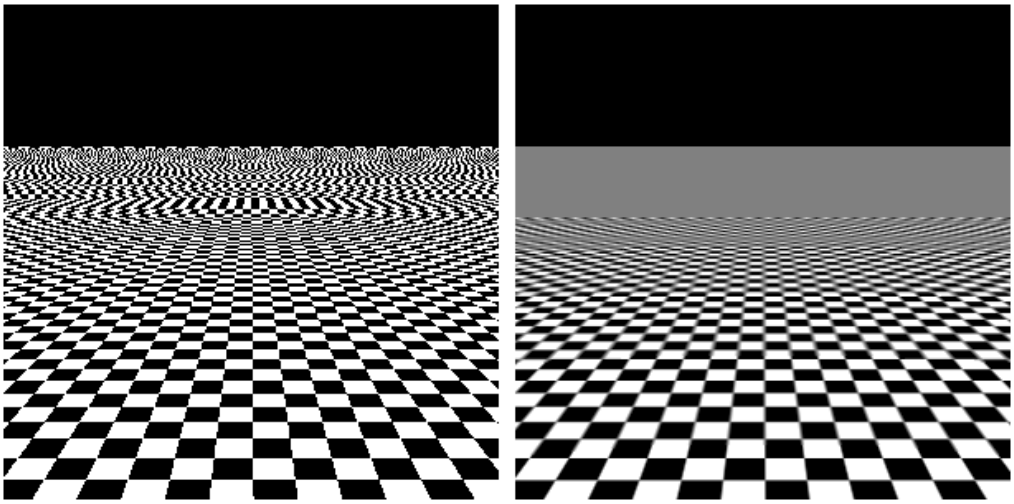
Herefter er det relevant at undersøge, hvilke teknologier og programmeringspraktikker, som kræver stor computerkraft. Specielt inden for videospil er der utrolig mange systemer og elementer, som skal spille sammen, ikke mindst det grafiske komponent.

GPU - Graphics Processing Unit

GPU'en, også kaldet grafikkort, er det stykke hardware, som står for de grafiske udregninger. Sammen med processoren er det ofte her flaskehalsen i en enheds ydeevne opstår. I mobile enheder er grafikkortet typisk begrænset af 'fillrate', som angiver, hvor meget grafikkortet kan rendere og skrive til grafikhukommelsen i sekundet. Fillraten afhænger oftest af shaderen, som er det stykke software, der udregner det visuelle resultat af interaktionen med modeller, materialer, lys, skygger mv. Brug af en kompleks shader kan resultere i at applikationen hæmmes af enhedens fillrate, og det anbefales derfor at benytte den simpleste mulige shader [44] [45]. Unity kommer med en række shadere der er optimeret til mobilenheder, som vi kan gøre brug af.

Dog findes der andre områder, som kan optimeres på. Et er tekstur-detaljegraden, som kræver hukommelse af den mobile enhed. Kræver spillets teksturer for meget hukommelse, er der et par metoder, som kan tages i brug. Det første der bør overvejes er komprimering af teksturene, hvilket gør dem lettere at indlæse i hukommelsen.

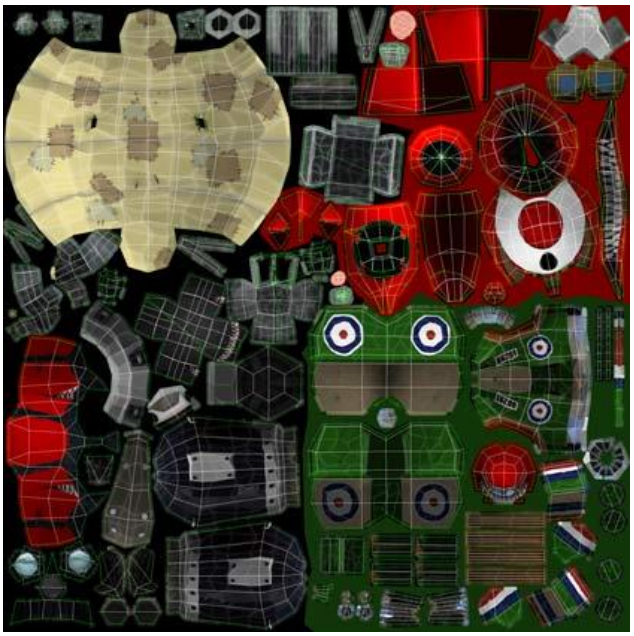
Yderligere bør der gøres brug af en teknik kaldet mipmaps. Disse er en række versioner af den samme teksture med gradvist lavere opløsning. De erstattes med den oprindelige tekstur, typisk når kameraet flyttes længere væk fra objektet med den tilhørende tekstur. Formålet med teknikken er at øge renderingshastigheden, da de mindre teksture anvender mindre hukommelse, og samtidig reducerer den også GPU-forbruget. Ydermere er det muligt at øge den visuelle kvalitet, da rendering af teksture med høj opløsning med eksempelvis høj afstand til kameraet kan resultere i moiré mønstre [46]. Her (Figur 7) ses et eksempel på ingen brug af mipmaps, imod brug af mipmaps.



FIGUR 7 - TV. INGEN MIPMAPS. TH. MED MIPMAPS.

Som det ses på Figur 7, er der stor visuel forskel på de to billeder. Billedet, som gør brug af mipmaps, er langt mere behageligt at se på, selvom billedet uden brug af mipmaps teknisk set har højere opløsning. Desuden er det muligt at se moiré-effekten i det venstre billede, da den høje opløsning ikke kan gengives i horisonten. Brug af mipmaps øger renderingshastigheden, men det skal dog bruges med omtanke, da processen i sig selv kræver omkring 33% mere hukommelse [47].

En anden måde at optimere brugen af teksturer på, er ved at benytte sig af et såkaldt ”texture-atlas”. Et teksturatlas er en billedfil, som indeholder en lang række, hvis ikke alle, tekstur i en scene. Dette gør renderingen mere effektiv, da filen bliver håndteret som en enkelt enhed i udregningerne, hvilket især er effektivt, hvis spillet indeholder mange små objekter, som bruges ofte. Et typisk atlas kunne se ud som på Figur 8.



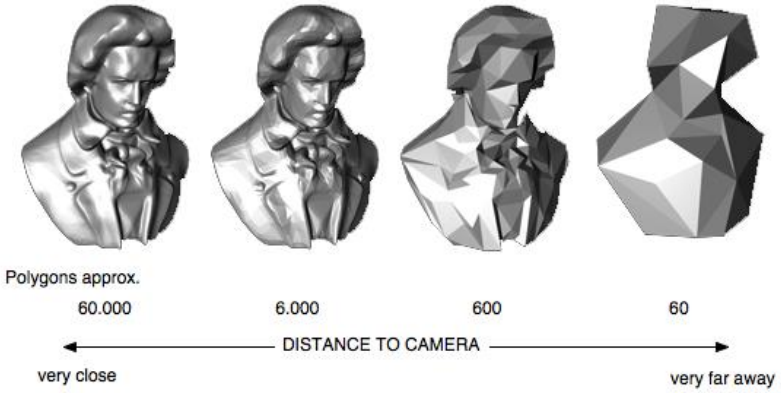
FIGUR 8 - EKSEMPEL PÅ ET TEXTURE ATLAS

Det kræver dog lidt tilpasning at anvende et tekstur atlas, da UV-mappet skal hænge sammen med den tilhørende model. UV-kortlægningen er processen hvorved et 2D billede genereres udfra en 3D model [48].

Lægger man alle teksturene fra en model i en ny fil kan det være nødvendigt at opdatere UV-mappet, så teksturerne også efterfølgende bliver pålagt korrekt.

Når man genererer et tekstur atlas er der en risiko for, at teksturene at overlapper hinanden ved brug af mipmaps, og det er derfor igen nødvendigt at sikre sig at underbillederne ikke bliver forurenet af de omkringliggende billeder. Hvilket man typisk gør ved at sikre afstand imellem de forskellige teksturer på atlasset.

Sidst er det værd at nævne LOD, som står for Level Of Detail. Med LOD forsøger man at mindske antallet af draw calls, som er antallet af materials, der tegnes på skærmen på ethvert givent tidspunkt, ved at simplificere objekterne i scenen ud fra afstanden til kameraet [49]. Teknikken minder meget mipmaps, men dækker over modellerne i scenen. Et eksempel kan ses på Figur 9.

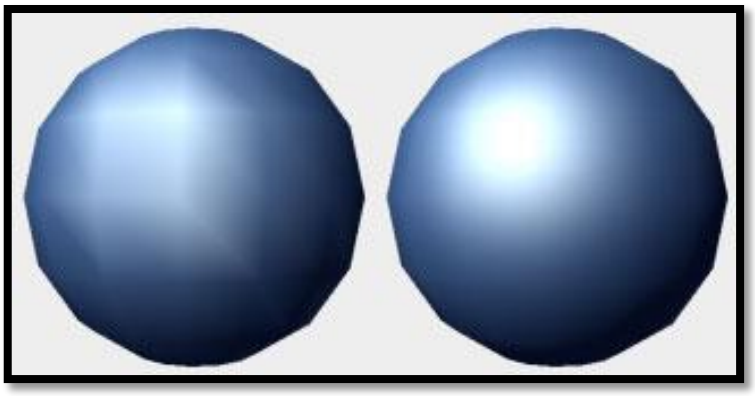


FIGUR 9 - DEMONSTRATION AF LOD [50]

Derudover er det naturligvis også en mulighed at slette objektet helt, når det opnår en given afstand fra kameraet. Samtidig med det er muligt at udnytte LOD-funktionalitet til terræn, så er det muligt at kontrollere om shaderen skal bearbejde de objekter med høj afstand til kameraet.

Optimering af lys

Et andet element, som i høj grad påvirker ydeevnen er de forskellige former for lys, som kan indbygges i et produkt [51]. Lys i Unity3D bliver renderet på én af to måder, vertex lys eller pixellys. Vertex lys er lys, som bliver udregnet per vertex i scenen og langsomt aftager fra vertex´en. Pixel lys bliver herimod udregnet for hver enkelt pixel på skærmen. Forskellen ses i Figur 10, vertex lys til venstre og pixel lys til højre.



FIGUR 10 - TV. VERTEX LYS. TH. PIXEL LYS [52]

Som det ses giver pixel lys en langt højere kvalitet, men stiller også langt større krav til den mobile enheds ydeevne. Dog giver pixellys nogle muligheder, som ikke er tilgængelige med vertex lys. Med pixellys er det nemlig muligt at benytte normal maps, light cookies og realtime skygger.

Normal maps er en teknik, hvor det er muligt at gøre overfladen på et objekt mere detaljeret uden at redigere selve 3D-modellen [53]. Light cookies er en funktion, som giver lyset et slags filter, hvor formålet

er at give udvikleren mulighed for manuelt at bestemme lysets mønster [54].
Realtime skygger er, som navnet indikerer, realtidsskygger, som opdatere løbende som objekter og lys mv. ændres.

Realtime skygger kræver mange ressourcer og er derfor et fokuspunkt i optimeringen af lyselementerne. Processen forløber således, at først renderes alle potentielle objekter, som kan kaste en skygge på et skyggekort. Herefter renderes alle objekter, som kan modtage skygger, ind i skyggekortet. Denne proces er derfor mere krævende af grafikkortet end pixel lys i sig selv [55].

3D-modeller

Tredimensionelle modeller, som ofte er brugt i videospil, kan være en krævende komponent i et produkt. Det er dermed også et element, hvor ydeevnen hele tiden skal være et fokuspunkt under produktionen. Når en 3D-model udvikles til brug for et mobilspil, er der en række punkter som bør undersøges:

Det første som bør redegøres for, er antallet af polygoner som modellen skal bestå af. Antallet af polygoner bestemmer detaljegraden på figuren og dermed kvaliteten af det visuelle element. Men jo højere antal polygoner, jo højere ydeevne kræves der af hardwaren.

På en mobil enhed er 300-1.500 et acceptabelt antal polygoner per model sammenlignet med en stationær computer, som kan håndtere et antal på 1.500-4.000 polygoner [56]. På spilkonsollerne Xbox 360 og Playstation 3 har modeller gennemsnitligt et antal polygoner på 5.000-7.000.

Dog er det værd at indberegne antallet af modeller på skærmen samtidig, da et højt antal modeller naturligvis medfører højere krav til enhedens ydeevne. Netop fordi det handler om antallet af polygoner på skærmen til enhver given tid, og ikke antallet af den enkelte models polygoner.

Det næste mål for optimeringen er at bruge så få materialer som muligt. Et materiale i 3D-modelleringssammenhæng er en samling af teksturer samt shaders mv. på modellen. Dog er det typisk ikke nødvendigt med mere end ét materiale, men der kan opstå situationer, hvor kvaliteten kan forbedres ved brug af flere. Dette er typisk ved situationer, hvor en figur bruger mere end én shader [56].

Endvidere bør det være et mål at bruge så få knogler i modelleringen som muligt. Knogler hentyder til et system, som gør det muligt at animere og styre 3D-modeller uden at skulle ændre på selve polygonerne.

Det anbefales at bruge højst omkring 30 knogler i en model til brug for mobilapplikation. Dette kan sammenlignes med et computerspil typisk hvor tallet er imellem 15 og 50 [56].

Det anbefales kun at bruge en enkelt ”skinned mesh renderer” for hver karakter i spillet. Mesh renderen er den, som indlæser modellen samt materialer mv. Tilføjer man derfor flere mesh rendere tager renderingen tilsvarende længere tid, og der er oftest ikke nødvendigt med mere end én renderer [56].

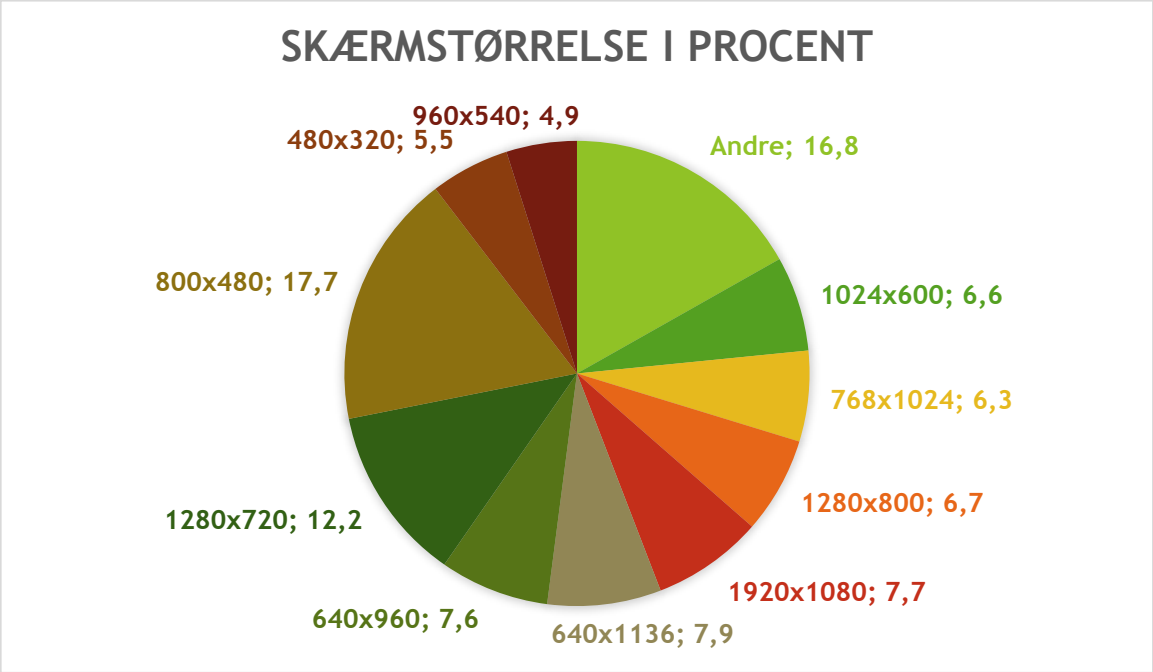
Skærmstørrelser

Når man skal udvikle en brugergrænseflade er der flere elementer, der skal tages højde for. Heriblandt skærmstørrelse og skærmens dimensioner, altså forholdet mellem længden og bredden. Skærmstørrelse og dimensioner kan variere betydeligt alt efter hvilken platform der udvikles til. Det gælder for mobile enheder som smartphones og tablets, men også computerskærme, tv etc.

I dette projekt, hvor produktet er henvendt til smartphones og tablets, er der også et væld af forskellige skærmstørrelser som der skal tages hensyn til under udviklingen. Konsekvensen af ikke at indarbejde de nødvendige fleksible størrelser kan i værste tilfælde gøre produktet uanvendeligt, da brugeren ikke kan benytte sig af de funktioner som er implementeret. I bedste fald er brugergrænsefladen ikke æstetisk nydelig, men dog anvendelig

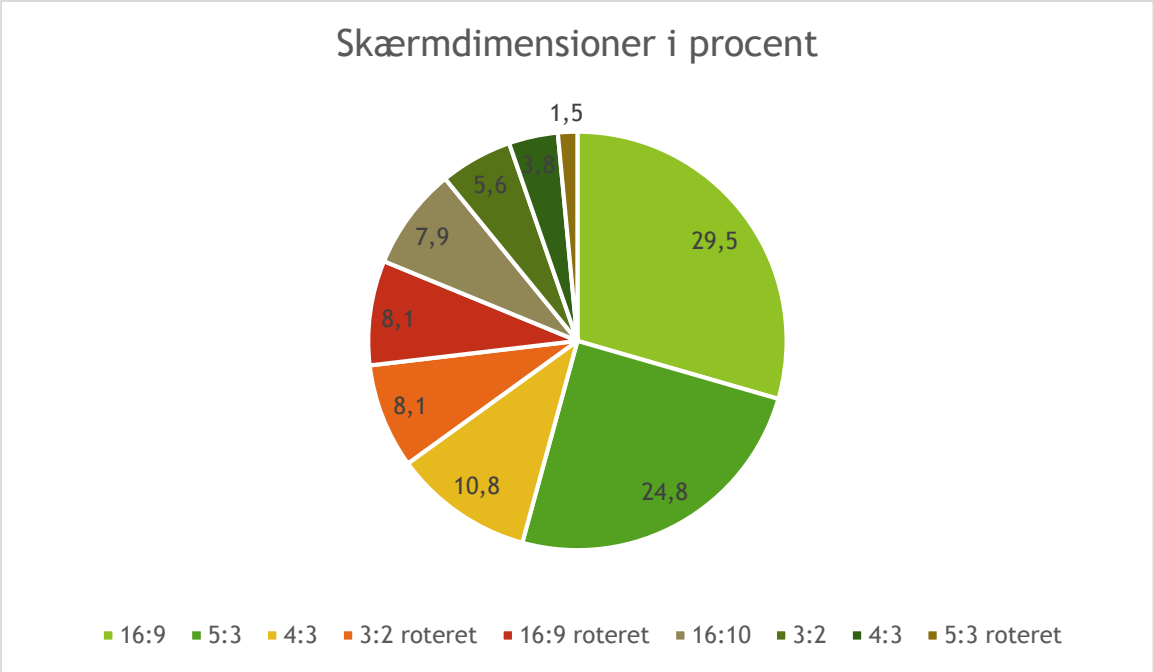
Analyse af skærmstørrelser

Før udviklingen påbegyndes er det nødvendigt, at vide præcist hvilke skærmstørrelser der er mest udbredte og som derved gør sig gældende. Med denne viden er det muligt at spare ressourcer, da man kan se bort fra eventuelle obskure formater. Ligeledes kan det bruges til at beslutte hvilke formater som er vigtigst at få et godt resultat på.



FIGUR 11 - UDBREDELSE AF SKÆRMSTØRRELSER I PROCENT [57]

Som kan ses ud fra Figur 11, ligger størstedelen af de mobile enheder inden for relativt få forskellige skærmstørrelser, da 'Andre' kun er 16,8% af alle enheder.



FIGUR 12 - UDBREDELSE AF SKÆRMDIMENSIONER I PROCENT [57]

Kigger man på de typiske dimensioner på markedet, som vidst på Figur 12, ser man at ’16:9’ og ’5:3’ udgør knap 50% af markedet. Da disse udgør de ydre ekstremer af de mest brugte dimensioner, er det derfor oplagt, at lægge fokus her.

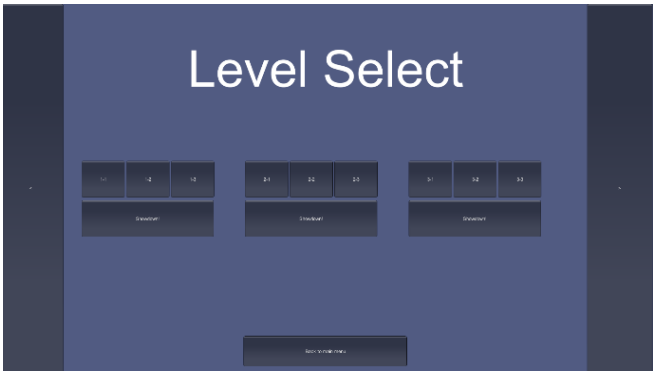
Design af brugergrænseflader

Ved design af grænseflader, er der en forskellige teorier og designregler, som kan understøtte udviklingsarbejdet. I dette projekt anvendes der en række af disse designregler.

Vores menusystem er, i denne prototype, der hvor man bedst kan se hvilke designprincipper vi anvender. Helt konkret er der tale om "De Fem Gestaltlove": nærhed, lighed, lukkethed, forbundethed og figur og baggrund.

- Loven om nærhed beskriver, hvordan elementer, som står tæt, opfattes sammenhængende.
- Loven om lighed beskriver, hvordan elementer, som visuelt ligner hinanden, opfattes som sammenhængende.
- Loven om lukkethed beskriver, hvordan brug af rammer og bokse indikerer sammenhæng.
- Loven om forbundethed beskriver, hvordan elementer har sammenhæng ved brug af eksempelvis streger eller baggrundsfarve.
- Loven om figur og baggrund beskriver forskellen imellem, hvornår en figur opfattes som en figur og hvornår den opfattes som baggrund.

Tager man eksempelvis udgangspunkt i menuen, 'Level Select', ses op til flere af disse love i brug. Menuen består af tre grupper, som hver indeholder fire knapper. Loven om nærhed indikerer her sammenhængen mellem de tre grupper og de tilhørende knapper. Samtidig kommer loven om lighed i spil, da de tre øverste knapper i hver gruppe har samme størrelse, udformning og navngivning, hvilket indikerer ens funktionalitet. Under disse knapper findes en større knap, som derfor skiller sig. Igen kommer loven om lighed i spil, da den store knap i de tre grupper åbner en anderledes bane end de ovenstående.



Ligeledes gøres der brug af disse love i resten af menu-systemet, hvor simple bokse og placeringer indikerer sammenhæng i funktionaliteten.

Testplan

Formålet med dette dokument er at sikre en bedre kvalitet af produktet, samt støtte selve testprocessen. For at fuldføre disse test er der et behov for en række værktøjer mv:

- Der er behov for minimum 1 PC, hvorpå det er muligt at afvikle produktet. På denne computer skal der være en kørende version af Unity3D samt et udviklingsværktøj, som eksempelvis Visual Studio eller MonoDevelop.
- Derudover er der behov for minimum 1 Apple enhed, hvorpå det testes, hvorvidt produktet afvikles på iOS. Det er foretrukket at teste produktet på andre Apple enheder, da eksempelvis skærmstørrelse er et nødvendigt element at kvalitetssikre. Derudover er der behov for Apple Developer software.
- Ligeledes er der behov for en Android enhed. Dog er Android-enhederne en del mere varierede og et bredt udvalg er derfor attraktivt for testen. Derudover skal Android Development Kit være installeret for at muliggøre disse test.
- Der forventes to udviklere til udførelse af disse test, og begge udviklere har samme niveau af ansvar.

Play test

Da der i dette projekt ikke eksisterer en kompliceret kodebase, men i langt højere grad separate scripts, der hver især udfører relativt simple opgaver, er det derfor besluttet at fravælge en række testtyper som under andre omstændigheder kunne være relevante. Dette gælder f.eks. unit test og integrationstest. Disse testtyper er erstattet af såkaldt "Play testing" hvilket betyder, at funktionalitet testes ved at spille spillet. Til formålet udarbejdes test dokumenter angivelse af den funktionalitet, som skal testes. Dette dokument ses i bilagslisten.

Når der observeres en fejl i spillet, udfylder testeren en fejlrapport, hvor de trin som ledte frem til fejlen beskrives, samt den observerede fejl. Udvikleren kan med disse informationer i hånden søge at udbedre fejlen. Når dette er sket sendes fejlrapporten tilbage til testeren, som så verificerer at fejlen er udbedret.

Brugertest

Produktet skal i høj grad udsættes for brugertest, da det er brugerens oplevelse som er hovedfokuspunktet i udviklingen. Formålet med brugertesten er at undersøge om spillet i sidste ende er tiltalende for brugeren. Under brugertesten bliver der fokuseret på spillets tempo, styringsfunktionalitet, sværhedsgrad og generelt om spillets indhold er passende. Brugertesten foregår ved, at en bruger gennemspiller spillet, hvor en udvikler observerer processen. Efterfølgende modtages der feedback fra brugeren.

Udvikling

Udvikling med Unity er en øvelse i sig selv, og jf. vores beslutning om, at centrere firmaet omkring denne Game Engine er det at opnå skarpe kompetencer i brugen af den, et af vores absolut vigtigste fokusområder. I det følgende kapitel gennemgås en række af de væsentligste brugerflader og arbejdskoncepter i Unity.

Introduktion til brugergrænsefladen i Unity

Brugerfladen i Unity består af en række vinduer, der hver især tjener sit eget formål i forbindelse med udvikling. For overhovedet at kunne bruge Unity er det vigtigt at forstå formålet med en række af de vigtigste:

Scene

Scene-vinduet er der, hvor man opstiller sin scene [58]. Man kan positionere sit landskab, sine pro- og antagonister, kameraer, lydkilder og alle de andre dele som udgør spillet.

Et spil kan bestå af så mange scener, som man ønsker, og det er helt og holdent op til udvikleren at afgøre hvor mange der er behov for. I vores prototype benytter vi en scene for hver menu, og en scene til vores spil.

Game

I game-vinduet vises spillet, som det ser ud, når det er renderet af scenens kameraer. Som sådan er det en rimelig approksimation af, hvordan spillet ville tage sig ud, hvis man kompilerede det og kørte det fra den platform, det er tiltænkt [59].

Project

Under projekt kan man organisere sine assets, samt importere eller oprette nye. Project-vinduet er at sammenligne med en materialekasse, hvorfra man tager de redskaber som man skal bruge for at kunne sætte scenen op [60].

Hierarchy

I hierarchy-vinduet kan man se hvilke GameObjekter, man har tilføjet til den valgte scene, samt deres indbyrdes hierarki. I kombination med Inspector-vinduet øger bidrager oplysningerne her i væsentlig grad til at øge overskueligheden i scenen.

Inspector

Inspektor-vinduet er kontekstuelt. Ved at markere et GameObjekt i hierarki-vinduet vises en liste over de komponenter som er tilknyttet dette GameObjekt. For hver komponent er har man både adgang til at manipulere den enkelte komponents public variabler. Og man kan tilføje et andet objekt til det valgte, for dermed at skabe en forbindelse imellem de to.

Hvis man derimod markere et asset under project-vinduet vises den type assets mulige importindstillinger i stedet i inspector-vinduet [61].

Animator
Animator-vinduet er en del af Mecanim systemet. Fra dette vindue, kan man opbygge den tilstandsmaskine som håndterer under hvilke omstændigheder forskellige animationer skal afspilles.

Spiludvikling med Unity
Spiludvikling med Unity handler grundlæggende om at manipulere GameObjekter, som vi herfra også refererer til som et GO, og deres komponenter. Tidligere i kapitlet beskrev vi, hvordan komponentbaseret udvikling fungerede på et konceptuelt plan, herunder forklares det hvordan Unity helt konkret implementerer det.

GameObject
Så snart man flytter et asset fra project- til Hierarchy-vinduet (eller direkte ind i scenen), bliver der oprettet GameObjekt som kan indeholde det asset [62]. I sig selv er et GO blot en beholder som alle typer assets og komponenter kan tilknyttes, for at opnå den ønskede funktionalitet.

Et GameObjekt består som minimum af et navn, et tag, et lag og en transform-komponent.

Det er også muligt at indlejre GameObjekter under hinanden, for at opnå et ønsket resultat. Et GameObjekt kunne være en mand, mens et andet er en hat. De to kan med fordel indlejres under et tredje GO kaldet: ”Mand-Med-Hat”.

Component
En Unity komponent en konkret ”opførsel” som man kan tildele et GameObjekt. En texture er en ”opførsel” der beskriver hvordan GO visuelt tager sig ud. Et script der får GO til at hoppe er en komponent osv.

Den eneste komponent som alle GameObjekter har tilknyttet, er en Transform. Denne komponent beskriver objektets position, rotation og skala i et tredimensionelt plan.

Prefab
En prefab er at betragte som en plantegning over et givent objekt [63]. Det er ofte en god ide, at lave en prefab ud af et GameObjekt når de ønskede komponenter er tilføjet og justeret. Man kan godt kopiere ”Mand-Med-Hat”-gameobjektet uden at lave en prefab med objektets oplysninger. Men det kan give problemer hvis man senere vil ændre i objektet. Tilføjede man f.eks. et par briller til ham, så ville kun det ene objekt blive opdateret. Hvis man i stedet lavede ”Mand-Med-Hat” til en prefab, så kunne man opdatere den, og så ville ændringen filtreres ud til alle kopier af denne prefab.

En prefab kan også instantieres på runtime via kode, hvilket er nyttigt hvis det ikke på forhånd er besluttet hvor og hvor mange af et givent GameObjekt der skal bruges i scenen [64].

Mecanim
Mecanim er Unity´s indbyggede animationssystem [65]. Mecanim giver mulighed for, ved hjælp af drag-and-drop-funktionalitet, at opsætte en tilstandsmaskine hvormed vores karakteres animationer kan håndteres.

Mecanim giver, blandt mange andre ting, mulighed for at ”retarget” animationer. Det vil sige, at en animation som er lavet til en anden karakter i et helt andet spil, kan tilknyttes vores karakter. Det giver mulighed for at købe animationer som et selvstændigt asset, da den ikke længere er bundet op på den figur den er lavet med. Alternativt til at udvikle spillet ved hjælp af Mecanim, er selv at scripte førnævnte tilstandsmaskine, hvilket absolut er muligt, men helt unødvendigt.

Object Pooling

'Object pooling' er en teknik, som bruges til at optimere brugen af større mængder objekter af den samme type. Et typisk eksempel af denne situation er brug af projektiler, hvor hvert projektil er et objekt.

Den simple løsning er at oprette et nyt objekt, når skyde-metoden køres, og ødelægge objektet, når det ikke bruges mere. Ulempen ved denne metode er, at det kræver CPU-kraft at oprette et objekt i hukommelsen. Dette kan især være et problem for mobile enheder, som har meget begrænset processorkraft.

I stedet for at oprette og ødelægge objekterne efter behov, så opretter man alle objekterne (af samme type) i hukommelsen, når scriptet bliver kaldt første gang. Objekterne bliver herefter sat til aktive, når de bliver brugt, og sat til inaktive, når de ikke bruges mere.

GUI Udvikling med Unity

I dette projekt har det været nødvendigt at gøre brugergrænsefladen dynamisk til forskellige skærmstørrelser, hvilket har været i fokus under udviklingen af menu systemet. Systemet, som består af en række GUI-elementer, er bygget i Unity3D's eget miljø vha. scripting.

Menuen, som i første omgang var udviklet ved brug af faste koordinator på skærmen samt faste pixelstørrelser, var ikke optimal til forskellige skærmstørrelser. For at overkomme dette problem er der gjort brug af de statiske variabler "Screen.Height" og "Screen.Width", som henviser til skærmens længde og bredde. Uanset hvilken skærmstørrelse der er tale om, kan man derfor, for eksempel, altid finde midten af skærmen ved at skrive "Screen.Width / 2" i scriptet.

Et eksempel på en knap kan derfor se sådan ud:

```
GUILayout.Button("New Game / Continue", GUILayout.Height(Screen.height / 12));
```

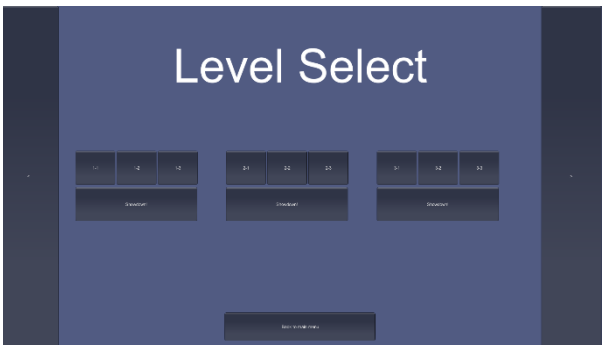
FIGUR 13 – KNAPPEN ER 1/12 AF SKÆRMENS HØJDE.

GUILayout opretter et element, som automatisk bliver lagt i rækker på skærmen, som gør koden mere overskueligt. Derefter bliver der oprettet en knap via "Button"-metoden, som indeholder følgende parametre, tekst, højde og vidde.

Teksten henviser til den tekst, som vises på selve knappen på i brugergrænsefladen. Knappen på Figur 13 er "New Game / Continue", som er det første punkt hovedmenuen. De næste variabler; højde og vidde, henviser til størrelsen på selve knappen, hvor højden i dette stykke kode er skærmens højde divideret med 12.

Vidden er ikke angivet her, da 'GUILayout' automatisk henter højden og vidden fra parent-objektet, medmindre andet er angivet.

Det parent-objekt, som der hentydes til, kan bestå af flere elementer. I denne sammenhæng er det funktionen, "GUILayout.BeginArea()", som opretter et område, der kan redigeres. Det bliver eksempelvis brugt i scriptet: "LevelSelect.cs", hvor det enkelte områder, som bliver oprettet indeholder en række knapper.



FIGUR 14 – LEVEL SELECT MENUEN.

Som det ses på Figur 14, findes der 3 grupper i midten af grænsefladen, som hvert består af et "BeginArea". Hver enkelt har skærmkoordinater, som fortæller, hvor på skærmen, knappen skal sidde. Disse koordinator kan dermed gøres dynamiske, og derved opnå en større ensformighed i brugergrænsefladen og mere generisk kode.

Kigger man på koden bag en af disse områder, er det muligt at se mekanikken bag sammenhængen imellem område og knapper. Koden for et enkelt af disse områder kan ses i Figur 15

```
GUILayout.BeginArea(new Rect(Screen.width/4-Screen.width/8, Screen.height/2-80, Screen.width, 240));
  GUILayout.BeginVertical("");
    GUILayout.BeginHorizontal("");
      GUILayout.Button("1-1", GUILayout.Height(Screen.height/10), GUILayout.Width(Screen.width/15));
      GUILayout.Button("1-2", GUILayout.Height(Screen.height/10), GUILayout.Width(Screen.width/15));
      GUILayout.Button("1-3", GUILayout.Height(Screen.height/10), GUILayout.Width(Screen.width/15));
    GUILayout.EndHorizontal();

    GUILayout.Button("Text!", GUILayout.Height(Screen.height/10), GUILayout.Width(Screen.width/5+5));
  GUILayout.EndVertical();
GUILayout.EndArea();
```

FIGUR 15 - LEVELSELECT.CS

Først oprettes et område som bliver tildelt dynamiske værdier, for at sikre det dynamiske layout. Derefter bliver der påbegyndt en funktion: ”GUILayout.BeginVertical”, som starter en lodret række af elementer. Inde i den påbegyndes den tilsvarende vandrette funktion, som indkapsler 3 knapper, som nu arver ”BeginArea”’s koordinater og automatisk er sat i række via ”BeginVertical” og ”BeginHorizontal”.

Den vandrette række bliver afsluttet med ”EndHorizontal”, hvorefter endnu en knap oprettes. Denne knap ligger nu under den første vandrette række. Sidst afsluttes den lodrette række, hvorefter området afsluttes.

Hver menu er blevet oprettet som en selvstændig scene med et tilhørende GUI, der tager højde for skærmens størrelse.

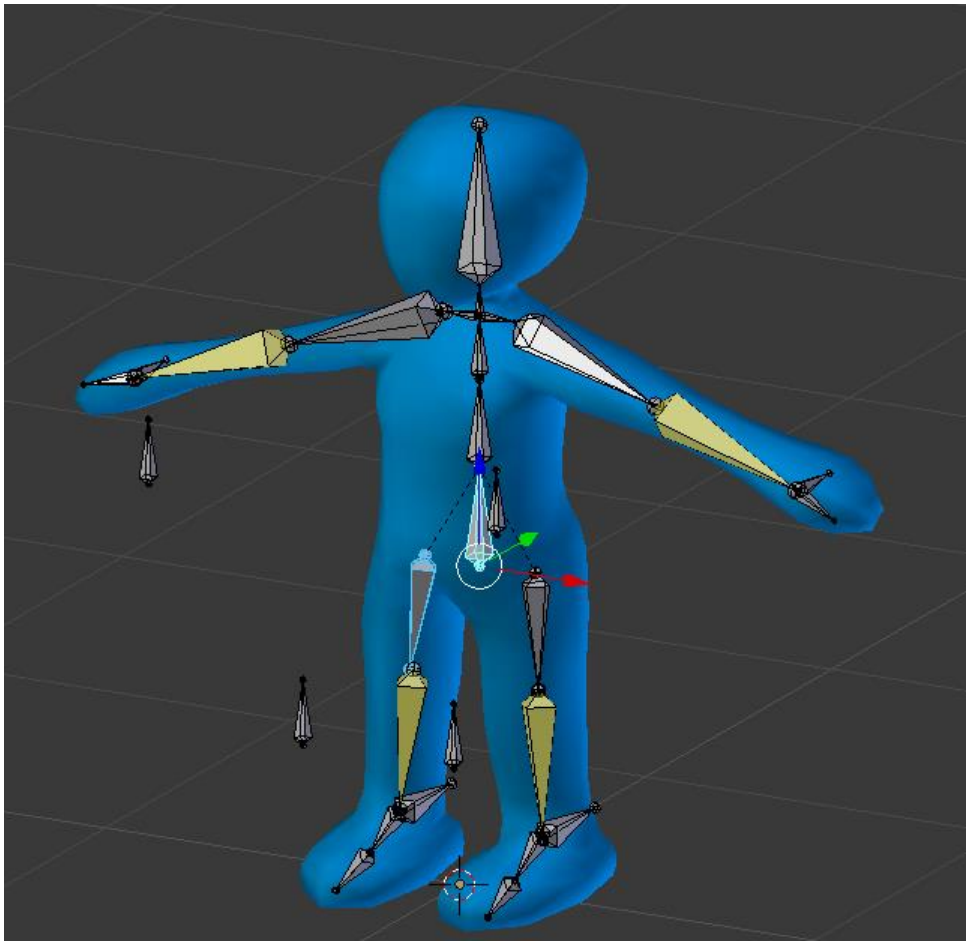
Mojo implementation

Mojo er spillets hovedperson og protagonist. I det følgende kapitel gennemgås det, hvordan han er blevet importeret fra 3d model til fuldt funktionel figur med character controller.

Import af humanoid Mecanim model

For at kunne importere en model til brug for mecanim ind i Unity, skal 3D modellen være sat op på et bestemt måde, før den importeres til Unity. Den proces, hvorved man gør en 3D model bevægelig, kaldes for "rigging" [66]. Konkret betyder det, at man allerede i modelleringsprogrammet tilføjer et antal "knogler" til modellen, på Figur 16 er det vist, hvordan knoglerne tager sig ud i Blender modelleringsprogrammet. I Blender er der også funktionalitet til at knytte knoglerne sammen med modellens mesh, altså dens væv.

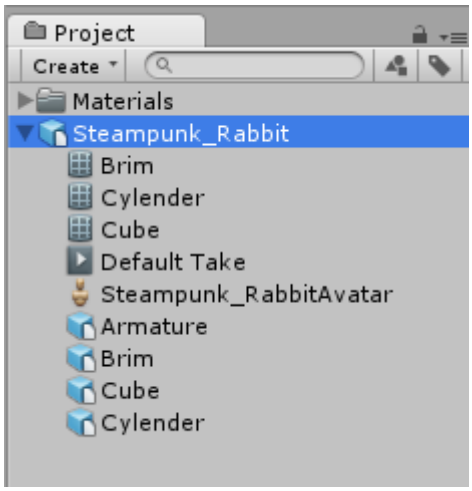
Når modellen er blevet rigget, kan man bevæge modellens mesh ved at manipulere med dens knogler.



FIGUR 16 - MODELRIGGING I BLENDER

Første trin for at kunne oprette Mojo, er er at producere en rigget 3D model af ham, og importere den til projektmappen.

Hvis man åbner modellen i project-vinduet, kan man se hvilke GameObjekter Unity inddeler modellen i. På Figur 17, ses de tre øverste mesh-objekter der tilsammen udgør figurens udseende. Default_Take er et animationsobjekt som Blender automatisk genererer ved eksport. Det er uden praktisk betydning for vores implementation.



FIGUR 17 - UNITY DELER EN IMPORTERET MODEL OP I GAMEOBJEKTER

Hvis man har oprettet en avatar til figuren, så ses det også her som et gameobjekt for sig selv. De tre nederste ikoner repræsenterer hver 3d model [67] som den samlede figur består af, mens armature-objektet er en reference til modellens rigging.

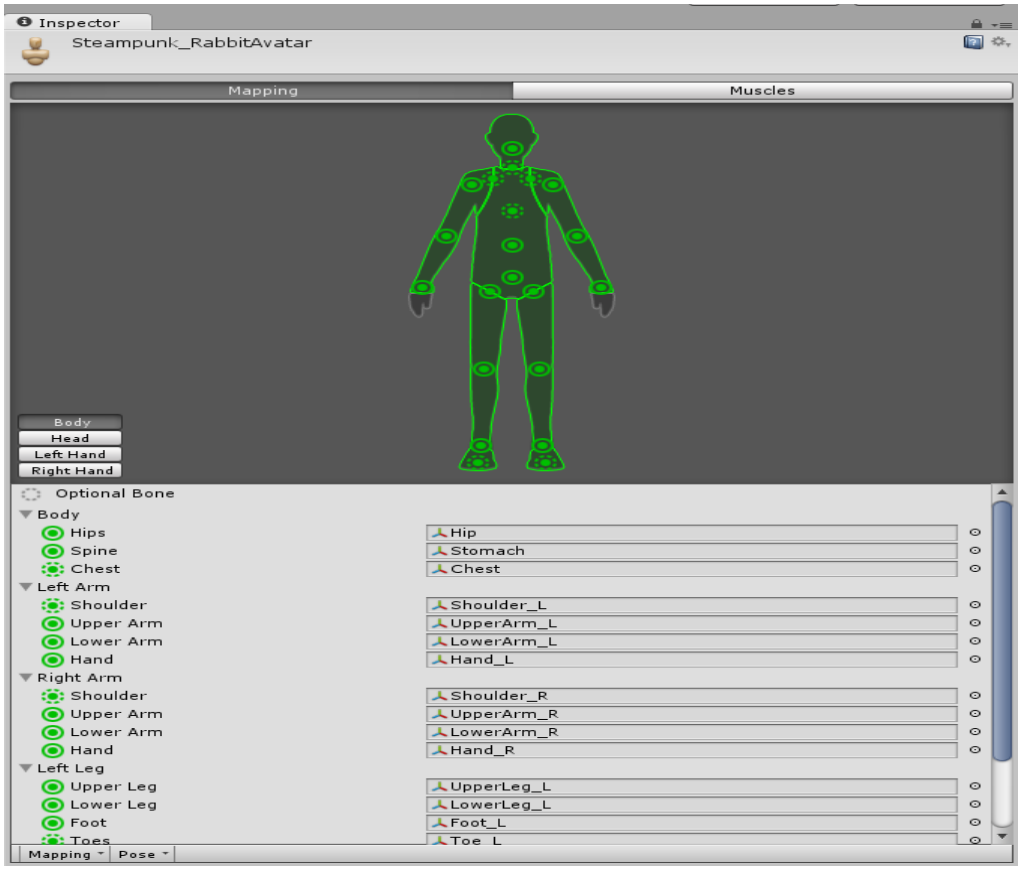
For at kunne benytte Mecanim systemet, skal der være oprettet en avatar af figuren. Ved at markere Mojo-modellen i projektmappen, vises importindstillingerne for 3d-modeller i inspector-vinduet. Under "*Rig*"-tabben vælger man hvilken animationstype Unity kan forvente sig af modellen. Mulighederne er "*Legacy*" hvorved man blot vælger Mecanim fra, og bruger Unitys gamle system i stedet. "*Generic*", hvilket resulterer i, at man selv skal sætte avataren op. Dette giver mulighed for at man kan have alle mulige typer knoglestrukturer i sine figurer. F.eks. firbenede væsener.

Sidste mulighed er "*Humanoid*", hvor Unity selv opsætter knoglerne på deres formodede position. Dette virker kun hvis der er tale om en figur med to arme, to ben, et hoved, hvilket er tilfældet her, og derfor vælges denne indstilling.

Indstillingen *Avatar Definition* afgør om Unity skal forsøge at opsætte avataren ud fra den valgte model, eller en anden. Mojo modellen har alt hvad der er påkrævet for at oprette en avatar, så det er indstillingen her. For at gemme indstillingerne klikkes på "*apply*", og dernæst "*configure*" for ved selvsyn at kontrollere Unity´s arbejde.

For at modellen kan fungere som en humanoid kræver Unity som minimum, at der er 15 knogler til stede. Fornuftig navngivning af knoglerne i Blender giver mindre manuelt arbejde nu, da det øger muligheden for at Unity selv kan placere hver knogle i den korrekte position. På Figur 18 kan man se hvilke positioner der er påkrævede i og med, at disse er hele cirkler.

Cirklerne med stiplede linier omkring indikere at der kan være en knogle på den position, men at det ikke er påkrævet. Mojo har kun de påkrævede 15 knogler, 3 i hver arm, 2 i torsoen, 3 i hvert ben og 1 i hovedet.



FIGUR 18 - CIRKLERNE MED STIPLEDE LINJER MARKERRE KNOGLER SOM ER VALGFRIE AT TILFØJE.

Tilbage til *Import*-indstillingerne [68], er der to yderligere tabs med indstillinger. Den ene er "*Animations*" hvorfra man kan indstille eventuelle animationer som medfølger på figuren. Som tidligere nævnt benytter vi, til vores prototype, udelukkende eksisterende animationer. Den anden tab er "*Model*", der indeholder forskellige importindstillinger for figurens mesh, normaler og materials. Der er ingen materialer på vores prototype figur ved import, så der er intet behov for at ændre i standardværdierne under dette punkt. Normals & Tangents kan med fordel indstilles når der opstår en situation hvor det er relevant at optimere på spillets størrelse. Uden de endelige modeller er der ikke noget behov for at ændre i standardværdierne på dette tidspunkt.

Reelt er *scale den eneste* værdi som der er behov for at indstille på denne side. Unity´s målestok er således udformet, at 1 enhed er lig med 1 meter. Blenders enhed er beregnet på en anden måde, så for at kunne få vores model importeret i den størrelse vi ønsker, skal den opskaleres. Det er forskelligt hvilket skaleringsforhold der skal anvendes, fordi det beror på hvilken størrelse modellen har haft da den blev modelleret. Vores grafiker har lavet modellen i Blender, så den passer med en 1:1 skalering.

Oprettelse af Mojo som GameObjekt

Efter at have oprettet Mojo som Mecanim-kompatibel model, kan han oprettes i scenen.

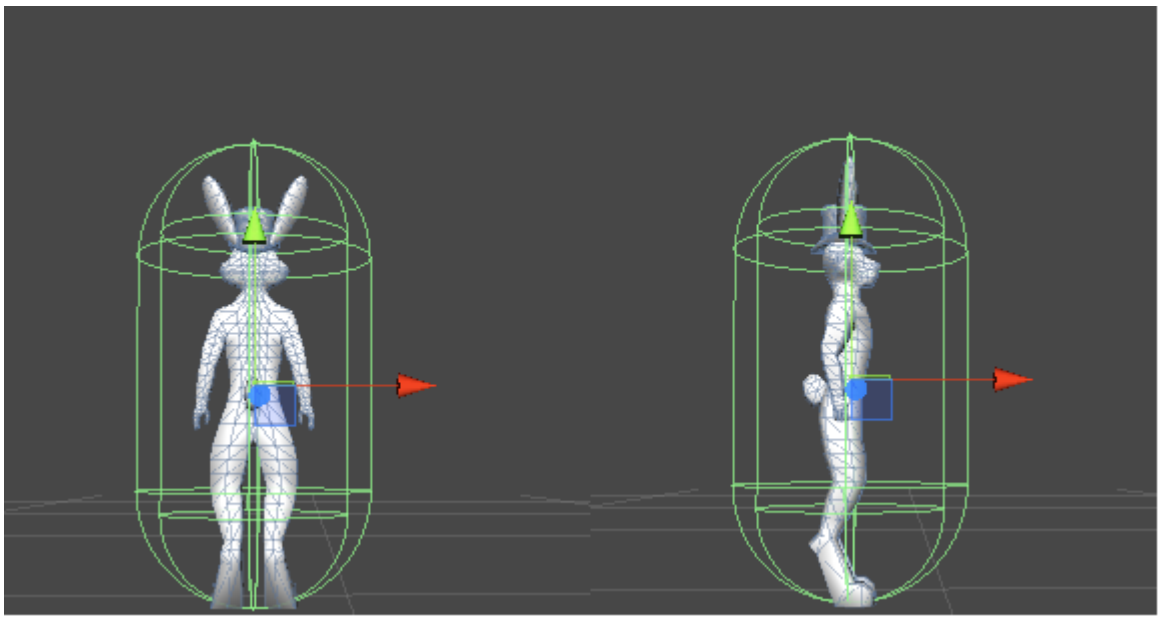
Som det første trækkes modellen fra project-view og ind i scene-view. For at kunne se om hans størrelse nu også passer, er det en god ide at lave et standard cube-objekt, til at sammenligne modellen med. Et cube-objekt er altid 1*1*1, hvilket gør det til en ideel målestok.

Næste udfordring opstår med erkendelsen af, at Unity og Blender ikke bruger samme koordinatsystem. I Unity er Y-aksen den vertikale, mens det i Blender er Z-aksen. Hvis man ikke tager højde for dette, ender man med uhensigtsmæssig opførsel når man senere forsøger at bevæge figuren. Vores scene er sat op i en forventning om at Mojo altid bevæger sig ud af X-aksen. Men pga. den forskel ville han rent faktisk bevæge sig ud af Z-aksen.

Der er flere måder at komme omkring problemet på. Der kunne tages højde for det ved udviklingen af character controlleren, så i stedet for at flytte ham ad Unity´s X-akse, kunne man i stedet flytte ham af Y-aksen, hvilket ville løse det umiddelbare problem.

Samtidigt ville det også i væsentlig grad bidrage til at gøre vores kode uoverskuelig, da det betyder at forskellige objekter bruger forskellige koordinatsystemer. En anden mulighed er at tilrette modellen i Blender, så den fra starten har den rigtige retning. Men det er kun en mulighed hvis vi altid selv laver alle modeller, og det er ikke givet.

For både at kunne bevare overskueligheden, og altid have kontrollen med alle figurer, løses problemet i stedet ved at roterer modellen 90 grader om Y-aksen, kompenseres der for den rotation som modellen blev importeret med. (Figur 19)



FIGUR 19 - TV. FØR ROTATION. TH. EFTER ROTATION

Før der tilføjes flere komponenter, oprettes der en prefab som kan indeholde Mojo-GameObjektet fremadrettet. Den oprindelige figur indeholder ingen af de komponenter som skal tilføjes fra dette punkt og frem, ved at gemme Mojo i en prefab, og blive ved med at opdatere den som processen skrider frem, kan vi nemt oprette en ny instans af Mojo, hvis det skulle blive nødvendigt.

Opsætning af animator controller-komponenten

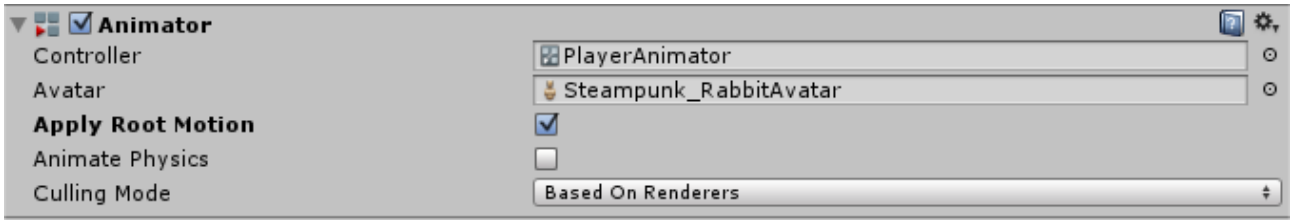
Da Mojo er oprettet som en Mecanim kompatibel rig, blev der automatisk oprettet en Animator-komponent på det GO han blev konverteret til da han blev trukket ind i scenen. Animator-komponenten (Figur 20) kræver to andre komponenter for at kunne anvendes: Avataren, som vi oprettede tidligere, og en controller. Desuden er der mulighed for at slå *Apply Root Motion* til, hvilket medfører at selve

bevægelsen af objektet kommer fra den animation som afspilles i den givne stund [69]. Vi anvender denne metode til at bevæge vores figur, fordi de animationer vi benytter os af passer i tempo.

Vi udvikler selv vores produktionsanimationer også, og dermed kan vi selv sikre at de også er afstemt efter hinanden.

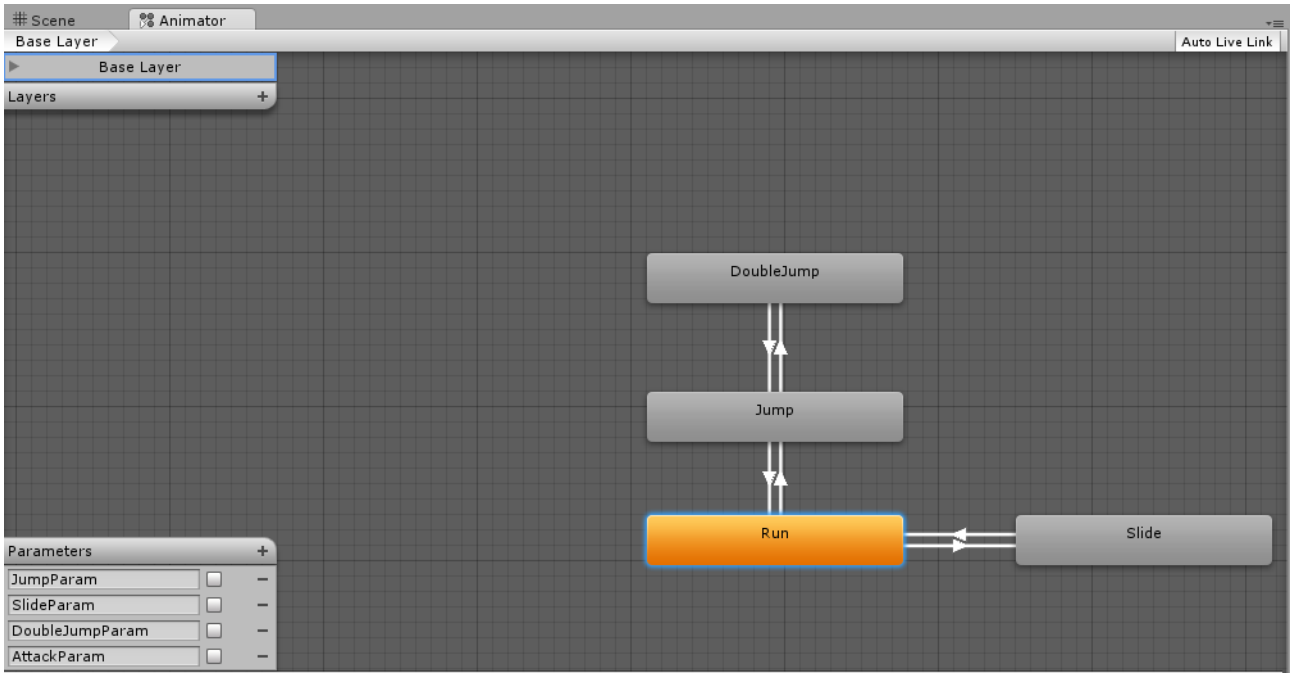
Animate Physics vil, hvis funktionen er slået til, betyde at animationen afvikles i Unity´s fysik loop. Hvilket kun er til nytte hvis GameObjektet, som Animatoren er tilknyttet har en Kinematisk Rigidbody [70]. Det er Mojo´s ikke, og derfor er funktionen slået fra.
Sidste egenskab som der skal tages hensyn til ved oprettelsen er *Culling Mode*. "Based On Renderer"-muligheden medfører at GameObjektet kun animeres når de renderer som er tilknyttet objektet ikke er synlige. Den anden mulighed "*Always Visible*" bevirker at GameObjektet altid animeres, uanset om det er synligt eller ej. I forhold til Mojo er det ikke så vigtigt, da han altid vil være synlig på skærmen. Men generelt er der ingen grund til, at animere et objekt som spilleren ikke kan se. Det tjener kun til at optage processorkraft. Uanset hvilket *Culling mode* man vælger, så bliver objekterne flyttet, selv om de er uden for skærmen så længe *Root Motion* benyttes [71].

Controller-komponenten indeholder den tilstandsmaskine som kontrollerer de skift i animationer som vi har behov for på vores protagonist. I vores GDD er det beskrevet hvilke features Mojo skal have til sin rådighed. Det vil sige, han skal kunne løbe, hoppe, dobbelthoppe, og slide. Controller-komponenten skal opsættes så der er overgange imellem de relevante tilstande.



FIGUR 20 - ANIMATOR KOMPONENTEN EFTER OPRETTELSEN AF CONTROLLER KOMPONENTEN.

Selve komponenten oprettes som et almindeligt asset igennem unity, og navngives "PlayerController". Dernæst tilføjer vi den, for nuværende, tomme komponent til Animator-komponenten.
For at kunne redigere i controlleren, åbnes Animator-Viewet. (Figur 21)

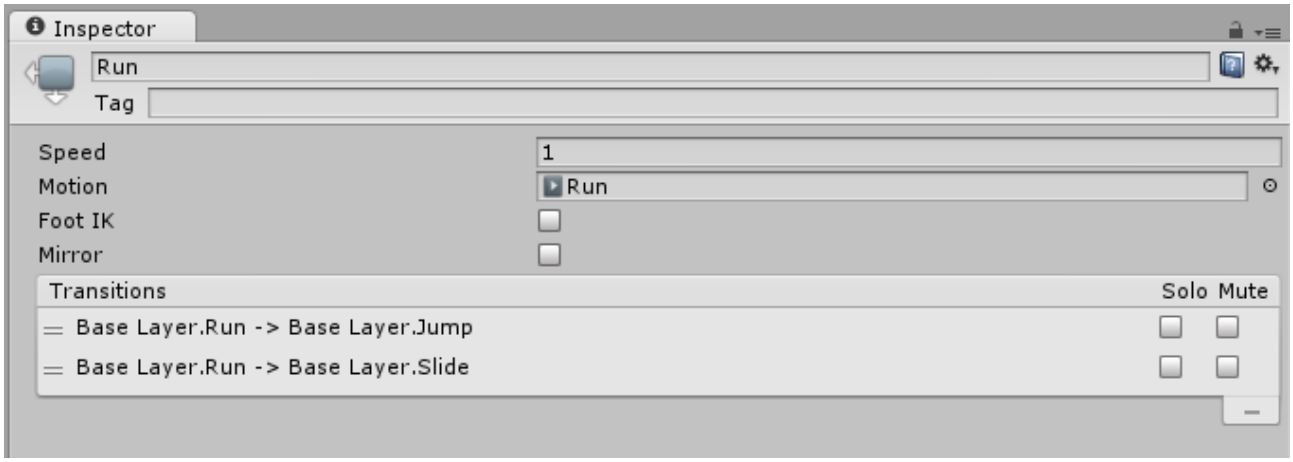


FIGUR 21 - PLAYERCONTROLLEREN

Herfra oprettes de forskellige tilstande, og de transitioner som knytter dem sammen. Desuden kan man her oprette de parametre som benyttes til at udløse en transition. Der er fire muligheder: int, float, bool og trigger, og de kaldes alle via script, som gennemgås herunder. Den eneste type vi benytter her er ”trigger”, som er en afart af en boolean. Når den bliver udløst sættes den til ”true” indtil transitionen er gennemført, hvorefter den uden videre vender tilbage til ”false”-tilstanden.

En tilstand indeholder en animation, der skal afspilles under en given omstændighed. Der er altid en tilstand, som skal være standard. I Figur 21 kan man se, at den er markeret med orange. I vores spil står protagonisten aldrig stille. Derfor er ”*Run*” vores udgangspunkt. For at tilføje en animation, markeres den ønskede tilstand, og animationen tilføjes til *motion*-egenskaben i inspector-vinduet (Figur 22). *Speed* indikerer hvor hurtigt animationen skal afspilles relativt til den hastighed hvormed den er optaget [72].

Foot IK angiver om der skal tages højde for invers kinematik i animationen. Kort fortalt er det en metode der benyttes for at det skal se mere realistisk ud hvis en figur bevæger sig på en ujævn overflade [73]. Det gør ingen af vores figurer, hvorfor vi har slået det fra. *Transitions*-feltet angiver hvilke ind og udgående transitioner der er på den valgte tilstand. Solo og Mute-checkfelterne er værktøjer som kan anvendes til at debugge tilstandsmaskinen, men som vi ikke benytter os af for nuværende.

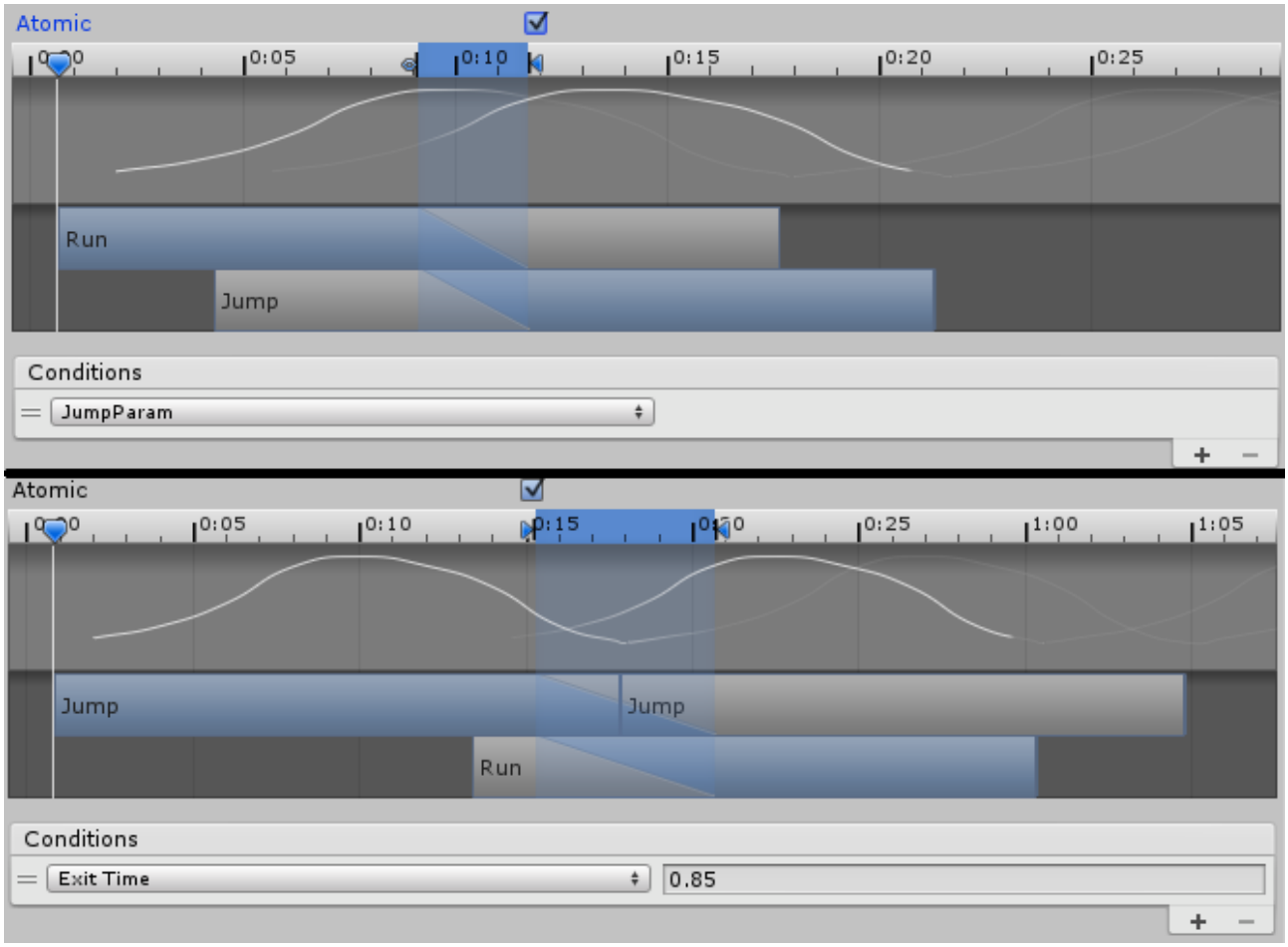


FIGUR 22 - INDSTILLINGSMULIGHEDER FOR EN TILSTAND

Hvis spilleren hopper, skal der være en overgang til at indikere den handling. Derfor er der imellem Run- og Jump-tilstandende oprettet en transition. På Figur 23 ses det, at den "*Condition*" der kræves for at udløse transitionen er at JumpParam er udløst. Det fremgår også, at overgangen er sat til atomic hvilket bevirker, at overgangsanimationen altid færdigøres før en ny transition kan udløses [74]. Kurven under indikerer den valgte overgangsfase. Hvordan den indstilles vil være forskellige fra gang til gang, og dikteres af hvordan den mest glidende overgang ser ud. Da vi bruger prototype animationer, er det indstillinger som skal revurderes ved implementeringen af produktionsanimationerne.

Af Figur 24 fremgår det også at, at der transitionernes tilbage til Run-tilstanden efter 0,85 sekund. Da vi har et dobbeltjump, skal det altså foretages inden for den tidsperiode for at udløse den næste transition.

Af transitionerne på Figur 21 fremgår det, at man kun kan hoppe, hvis man kommer fra løb, og kun kan dobbelthoppe hvis man allerede er i gang med at hoppe. På samme måde kan man kun slide, hvis man løber. Således er dette aspekt af vores Character Controller færdig.



FIGUR 23 - ØV. TRANSITION FRA RUN TIL JUMP. NDST. TRANSITION FRA JUMP TIL RUN.

Tilføjelse af yderligere komponenter

For at kunne fungere som ønsket, og for at kunne interagere med de andre elementer i scenen, kræves yderligere en række komponenter tilføjet til Mojo.

Rigidbody komponenten

En rigidbody-komponent gør det muligt for det GameObjekt som den tilknyttes at blive påvirket af Unity´s fysikmotor. Vi bruger kraft til at bevæge spilleren, hvilket ikke ville kunne lade sig gøre uden rigidbody, ligeledes forlader vi os på tyngdekraft i forbindelse med vores jump metoder.

Der findes andre måder at bevæge GameObjekter på i Unity, f.eks. kan man bevæge objektets transform direkte, eller man kan benytte den character controller-komponent som kommer med Unity. (Ikke at forveksle med det generiske term Character Controller, som dækker over alle de aspekter som tilsammen danner grundlag for karakterens bevægelse)

Det er et langt stykke hen af vejen et spørgsmål om at træffe et valg. Den ene metode kan ofte være lige så god som den anden. Vi har valgt at benytte fysik til at bevæge vores pro- og antagonister fordi vi mener det giver os den fornødne direkte kontrol med de objekter.

Attributten *mass* repræsenterer objektets vægt. Massen har en betydning ifh til hvor meget kraft der skal til for at bevæge objektet. Vi har beholdt standardværdien, fordi vi alligevel kontrollerer den kraft Mojo skal påvirkes med.

Drag og *Angular Drag* henviser til vindmodstand, hvilket ikke er en faktor i vores spil hvorfor vi bibeholder standardværdierne 0 og 0.05.

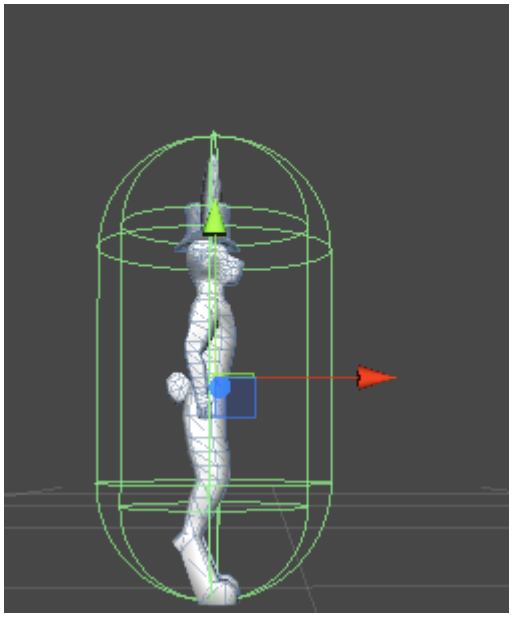
For at vi kan drage nytte af, at Unity's indbyggede fysikmotor beregner tyngdekraft, skal *Use Gravity* være slået til. *IsKinematic* bevirker at objektet ikke påvirkes af fysikmotoren. I den sammenhæng er det værd at bemærke, at der kan være andre grunde til at bruge en rigidbody, end det vi anvender den til. Til vores formål ville *IsKinematic* aldeles modvirke hensigten.

Interpolate og *Collision Detection* bruges til hhv. at smidiggøre animationen hvis den "hakker", og til at sikre at objekter der bevæger sig meget hurtigt ikke slipper igennem kollisionsmaskeringen. Vi har ladet begge dele stå på deres standardindstillinger, da vi dels ikke har et behov for at ændre det, og dels at det er de mindst ressourcekrævende indstillinger [75].

De sidste indstillinger er objektets *Constraints*. Som nævnt bevæges Mojo ved hjælp af kraft, ligesom alt hvad der findes i den virkelige verden. Men han har ingen balanceevne, så for at han ikke skal vælte omkuld kan vi låse hans rotation omkring både X-, Y- og Z-aksen. Det er ligeledes muligt at låse objektets position på alle tre akser. Det gør vi dog ikke, fordi det giver uhensigtsmæssigheder i kombination med at vi anvender *Root Motion* til fremdriften.

Capsule Collider, fysik-komponenten

For at undgå, at Mojo falder igennem de objekter som han står på tilføjer vi en Capsule Collider-komponent. Der findes en række forskellige collider-komponenter indbygget i unity. Vi benytter en capsule, fordi formen passer godt på en humanoid.

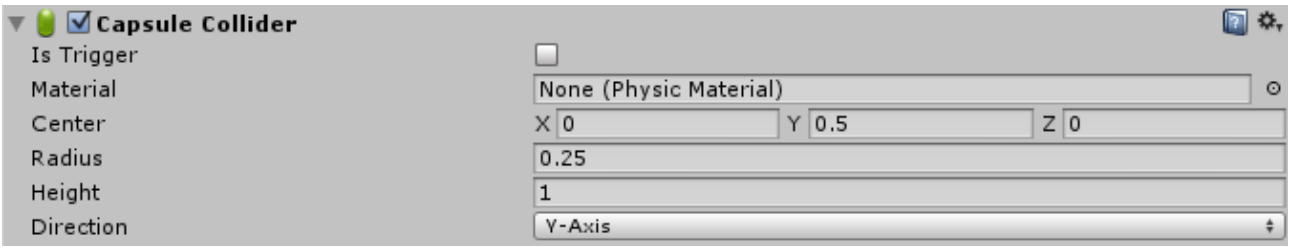


FIGUR 24 - MOJO OMGIVET AF CAPSULE COLLIDER

Attributten *Is Trigger* angiver om Collideren (Figur 25) skal fungere som en trigger eller, som her, om den skal være en fysik collider.

Material åbner mulighed for, at man kan tilføje et fysikmateriale til komponenten. Ved hjælp af et sådant kan man påvirke friktionen mv. imellem to gameobjekter. Det er ikke indenfor scope af denne prototype at benytte fysikmateriale.

De resterende egenskaber benyttes til at indstille colliderens størrelse og position i forhold til det GO som den er tilknyttet. På Figur 24 er den indstillet til at dække Mojo, og kan ses som et grønt "grid" over selve figuren.



FIGUR 25 - CAPSULE COLLIDER-KOMPONENTEN

Capsule Collider, Trigger-komponenten

Udover at have et behov for ikke at falde igennem objekter, skal Mojo også bruge en trigger collider. Med en sådan tilknyttet, kan andre objekter bekræfte når deres egen trigger collider interagerer med Mojo´s. Fordi der allerede er tilknyttet en collider på Mojo´s øverste GameObjekt, er det ikke praktisk at tilknytte en anden af samme type, da det forvirrer den metode hvormed man kalder komponenter.

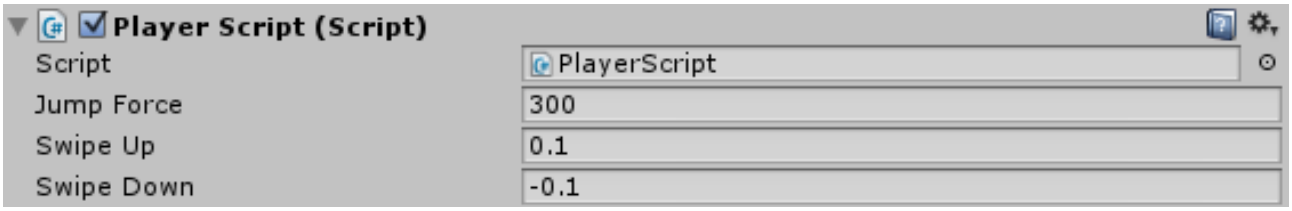
Løsningen er at oprette et tomt GO, og gøre den til et barn af Mojo-GameObjektet. Derefter kan man tilføje en Capsule Collider til det objekt, og give den de samme dimensioner som fysik collideren (figur 34). Eneste forskel er, at til denne collider skal *Is Trigger*-checkboksen vinges af.

PlayerScript-komponenten

PlayerScriptet oprettes fra Asset-menuen som et C#-script og placeres i projektmappen under Scripts. Formålet med scriptet er at forbinde de allerede oprettede komponenter således, at den opførsel, som de er tilføjet for at bidrage med, kommer til udtryk på den måde, vi ønsker. Det første vi gør i Scriptet, er at erklære, at UnityEngine skal anvendes, dernæst scriptets navn, og derefter, at vi nedarver fra MonoBehaviour.

Dernæst oprettes en række lokale og private variabler. De variabler som oprettes som public, bliver vidst i Unity-editoren, på samme måde som de attributter der vises i f.eks. RigidBody-komponenten. Det har begrænset praktisk betydning hvilke variabler vi sætter til hhv. privat og public, da vi er så lille et team som tilfældet er.

Ideen med at have denne opdeling er, at en designer, producer eller grafiker skal kunne justere i visse variabler uden at skulle håndtere koden. Fordi det er vores mening, at det er en god måde at effektivisere samarbejdet på, forsøger vi hele tiden at skelne til hvad der skal kunne ændres af alle, og hvad der er programmørens arbejde at ændre i.



FIGUR 26 - PLAYERSCRIPT-KOMPONENTEN. MULIGHED FOR INDSTILLINGER UDEN ADGANG TIL KODEN

Som det fremgår af Figur 26, kan man ændre i den kraft som objektet bliver påført når man hopper.Og man kan bestemme hvor kraftigt et ”swipe” på skærmen hhv. op og ned skal være for at blive evalueret til et ”swipe”.

De værdier afgør i høj grad om gameplayet fungerer som vi ønsker det, og ved at gøre dem public kan alle i teamet let teste forskellige kombinationer af.

Gennemgang af PlayerScript.cs

Ved GameObjektets oprettelse køres scriptets Start-metode som det første. Og for eneste gang i øvrigt, indtil objektet oprettes på ny.

I den får vi cachet, altså lagret, referencer til de komponenter som skal bruges i løbet af scriptet. Formålet med at gøre det på denne måde, er at det er ”dyrt” i ressourceforbrug at benytte de metoder som fremfinder komponenterne, hvis man gør det for f.eks. hver frame. I vores prototype ville det ikke være mærkbart, men det er essentielt altid, at udvikle med tanke på, at mindske systemkravene. Helt særligt ved udvikling til mobilenheder.

```
void Start () {
     _anim = GetComponent<Animator>();
     _psysCol = GetComponent<CapsuleCollider>();
     _body = GetComponent<Rigidbody>();
     _collisionCol = GameObject.Find("CollisionObj").GetComponent<CapsuleCollider>();
}
```

FIGUR 27 - PLAYERSCRIPT START()

GetComponent<T>() gennemsøger det GO som scriptet er tilknyttet efter en komponent af den type som er angivet [76]. Det er på grund af denne metode, at det ikke er hensigtsmæssigt at have to komponenter af samme type på det samme GameObjekt. Søgningen starter nemlig altid samme sted, og stopper når den første komponent er fundet.

Ret beset findes der en alternativ metode: GetComponents<T>() hvormed man kan søge flere af samme type. Men i givet fald skulle det håndteres i en form for liste, og dernæst alligevel gemmes i hver sin variabel, hvormed det er en mere akavet løsning end den valgte.

For at finde frem til den anden collider benytter vi metoden Find(String), der tager navnet på det eftersøgte GO som parameter. Da det ikke er GameObjektet vi skal bruge, benyttes GetComponent<T> direkte på det fundne GO.

Efter Start() er afsluttet går scriptet direkte ind i Update(), som bliver kaldt for hver frame, hvilket i praksis vil sige, at alt hvad der foregår i denne metode kaldes mange gange i sekundet.

For PlayerScriptet er der tre separate trin der udføres i Update(). Først opdateres den private variabel _currentBaseState, med den nuværende aktive tilstand i vores Controller-komponent (som jo er tilknyttet Animatorkomponenten, hvorfra denne oplysning trækkes.)

Den næste linie i Figur 28 sikrer at Mojo altid løber lige ud. De animationer der er tilknyttede er ikke hundrede procent i vater, hvilket uden denne linie, der for hvert kald til metoden nulstiller Mojo´s position på Z-aksen, ville betyde at Mojo løb skævt, og til sidst faldt af platformen.

Under kapitlet om RigidBody blev det nævnt, at det er muligt at låse Z-aksen helt. Men gør man det, så forhindrer man samtidigt at en animation i det hele taget kan bevæge sig på de låste akser, og det er ikke hensigtsmæssigt. Derfor håndterer vi det således.

```
void Update () {

     _currentBaseState = _anim.GetCurrentAnimatorStateInfo(0);
     transform.position = new Vector3(transform.position.x, transform.position.y, 0);

#if UNITY_ANDROID
     SwipeControls();
#endif

#if UNITY_EDITOR
     KeyboardControls();
#endif
```

FIGUR 28 - PLAYERSCRIPTETS UPDATE-METODE

De to if-sætninger er en speciel Unity-konstruktion, der sikrer platformspecifik kompilering. Alt hvad der ligger under UNITY_ANDROID bliver kun kompileret hvis man bygger et Android build. Det giver udviklere en mulighed for at designe scripts med henblik på udgivelse til flere platforme. Alternativt ville hele scriptet skulle kopieres, blot for at ændre det som er platformsspecifikt.

Vi anvender det her, for at sikre at vi har input-controls teste spillet både på vores mobile enheder, og i Unity editoren. Da KeyboardControls() kun er til testbrug, vil vi ikke uddybe den nærmere.
Den interessante af de to, er SwipeControls() der tager touch input. Metoden ses i dens helhed i Figur 29.

I den første if-sætning kontrolleres det om der er registreret mere end 0 berøringer, og om den første berøring er i "Moved"-fasen. Hvis det er tilfældet så findes berøringens deltaposition, altså dens position i denne frame, kontra der hvor den blev registreret sidst.

Vi er kun interesserede i swipet, hvis det bevæger sig enten op, eller ned. Hvis deltapositionen, som er en todimensionel vektor, er større end den værdi som er angivet til at tælle som et opadgående swipe, og den nuværende controller-tilstand er Run, så kaldes Jump(), og vi udløser den trigger der transitionerer controller-tilstanden fra Run til Jump. Ydermere ændres debug-teksten.

Hvis swipet er opadgående, og controller-tilstanden evaluerer til Jump, så udløses DoubleJump-transitionen. Og Jump-metoden kaldes på ny.

```
private void SwipeControls()
{
    if (Input.touchCount > 0 && Input.GetTouch(0).phase == TouchPhase.Moved)
    {
       Vector2 deltaTouchPos = Input.GetTouch(0).deltaPosition;

        if (deltaTouchPos.y > swipeUp && _currentBaseState.nameHash == _runState)
        {
            Jump();
            _anim.SetTrigger("Jump");   //Opdaterer animatoren
            _demoText = "Jump";
        }
        //hvis swipet er opadgående, og spilleren i forvejen i luften
        else if(deltaTouchPos.y > swipeUp && _currentBaseState.nameHash == _jumpState)
        {
            Jump();
            _anim.SetTrigger("DoubleJump"); //Opdaterer animatoren
            _demoText = "Double Jump";
        }

        //Hvis swipet er nedadgående og spilleren har jordforbindelse
        else if (deltaTouchPos.y < swipeDown && _currentBaseState.nameHash == _runState)
        {
            _demoText = "Duck";
            _anim.SetTrigger("SlideParam"); //Opdaterer animatoren
            Slide();
            StartCoroutine(WaitForSlide());
        }
    }
}
```

FIGUR 29 - SWIPECONTROLS() HÅNDTERER BRUGERINPUT

Hvis swipet derimod er nedadgående, altså en negativ y-værdi på deltapositionsvektoren, og Mojo er i løb, så udløses Slide-transitionen. Slide() ændrer størrelsen, og placeringen af den fysiske collider som er tilknyttet Mojo.

Grunden til denne manøvre er, at Mojo´s collidere skal ændre størrelsen. Animationen som sådan har nemlig ingen påvirkning på collider-komponenterne. Det vil sige, at selv om man kan se Mojo dukke sig på skærmen, så ville han stadig kollidere med eventuelle forhindringer som han ville have ramt hvis han stod op. Jf. koden i Figur 30 sættes højden på fysik collideren til en fjerdedel af normal, ligesom dens centrum flyttes til en fjerdedel af dens normale placering.

```
void Slide()
{
    _collisionCol.enabled = false;
    _psysCol.height = 0.25f;
    _psysCol.center = new Vector3(0, 0.25f, 0);
}
```

FIGUR 30 - SLIDE() SÆTTER COLLIDERE TIL STØRRELSE HVOR DE KAN GÅ UNDER FORHINDRINGER

Coroutinen WaitForSlide() har til formål at sikre, at fysik collideren ikke bliver sat tilbage til sin oprindelige position, før slide-animationen er udført. Derfor venteperioden på 1.2 sekund, før StandUp() kaldes.

```
IEnumerator WaitForSlide()
{
    //Debug.Log("waitforit");
    yield return new WaitForSeconds(1.2f);
    StandUp();
}
```

FIGUR 31 - COROUTINE EKSEMPEL

Efter slide-animationen er færdig, transitionerer tilstanden tilbage til Run. Derfor skal vi sørge for at Mojo´s collidere igen passer med hans oprejste form. StandUp-metoden som vidst i Figur 32 tilser dette.

```
void StandUp()
{
    _psysCol.height = 1f;
    _psysCol.center = new Vector3(0, 0.5f, 0);
    _collisionCol.enabled = true;
}
```

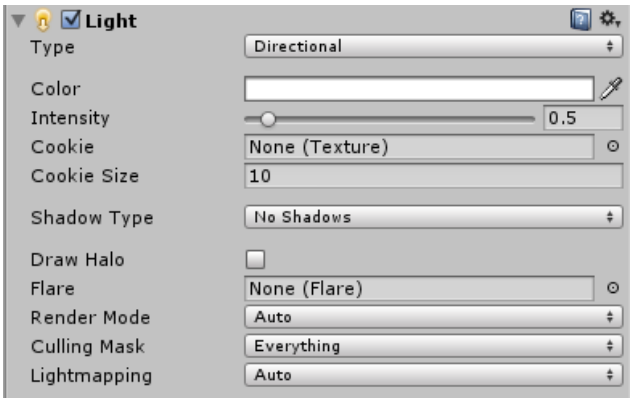
FIGUR 32 - STANDUP()

Opsummering

Afslutningsvis opdateres Mojo-prefabben med alle tilføjelser, og han placeres på det sted i scenen hvorfra han skal være placeret ved spilstart. Efter disse trin er vores protagonist klar til brug.

Implementation af lys i scenen

Lyset, som er brugt i dette projekt er relativt simpelt. Lyset i selve banen består af en enkelt *directional light*-komponent, som fungerer som en sol. Dette betyder, det kun er rotationen på lys-objektet, som gælder, da det belyser samtlige objekter i scenen. Samtidig den mindst krævende lyskilde målt på GPU-kraft [55].



FIGUR 33 - DIRECTIONAL LIGHT KOMPONENT

I forsøget på at holde den påkrævede GPU-kraft nede bruges der ikke realtime skygger, da disse er tunge for grafikkortet [55]. Dette betyder ingen pixellys, som er beskrevet i afsnittet "Optimering af lys". Som det ses i Figur 33, er lyset bygget så simpelt som muligt. Objektet gør ikke brug af *cookies*, *halo*, *flare* mv.

"Render Mode" definerer vigtigheden af et bestemt lys og har tre indstillinger: "*Important"*, som betyder, lyset altid renderes per pixel, "*Not Important"*, hvor lyset kun renderes vha. vertex-lys, og "*auto*". Lyskomponenten i dette projekt står til "*Auto*", hvilket betyder, at den gør brug af Unitys indbyggede kvalitetsindstillinger [51].

Det samme gælder for *Lightmapping*, som definerer, hvorvidt der bruges realtime lys, "baked" lys, eller der hentes fra kvalitetsindstillingerne. Realtime-lys, som navnet hentyder, beregnes i nutid, og er derfor en tung teknik at afvikle. "Baked" lys er en teknik, hvor lyset beregnes og indtegnes i selve objekternes tekstur. Et krav for dette er bl.a., at objekterne kategoriseres som værende statiske, da det ikke giver mening at bage lyset ind i bevægelige objekter. Da der i dette projekt ikke gøres brug af statiske objekter, er "baked" lys ikke en mulighed. Tilbage er det naturlige valg at overlade det til Unity3Ds egne kvalitetsindstillinger [51].

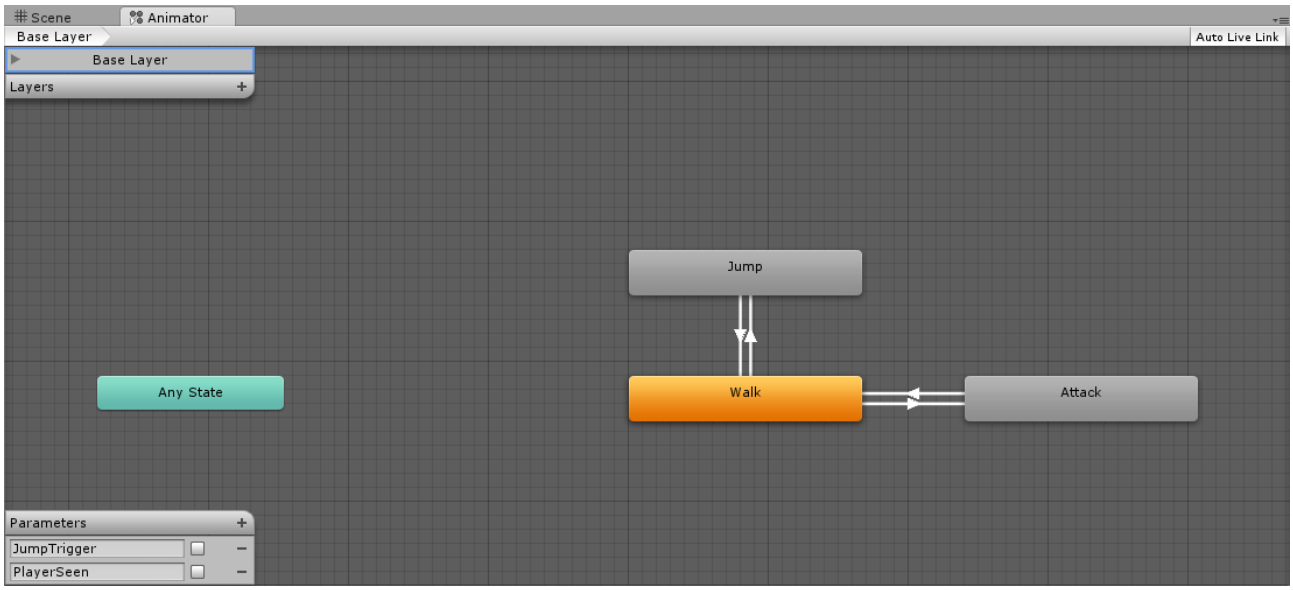
"Culling mask" definerer, hvilke objekter, som belyses i scenen. Indstillingen er sat til "*everything"*, hvilket vil sige at alle objekter bliver lyst op. Vi skal gerne have lys på det hele, og det er med til at afgøre antallet af objekter, som kan vises på skærmen på samme tid [51].

Hvordan er vores NPC implementeret?

Vores NPC er en massefabrikeret robotkanin, der blot er designeret ”EvilBunny”. Modellen er importeret og oprettet efter samme metode som beskrevet i kapitlet omhandlende Mojo-karakteren. I lighed med Mojo, har denne figur tilknyttet en rigidbody, en fysik-collider, og en animator med ”*Apply Root Motion*”-funktionalitet slået til. Disse komponenter er der gjort rede for i kapitlet der omhandler implementeringen af Mojo, og derfor beskrives de ikke nærmere her.

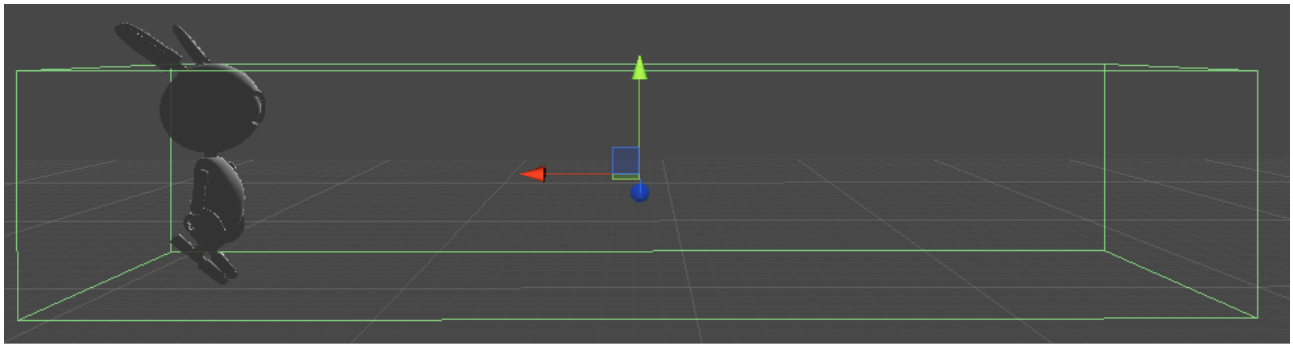
Vi har oprettet en ny controller-komponent til EvilBunny, da den ikke har samme funktionalitet som Mojo. Som det fremgår af Figur 34, så er udgangspunktet for EvilBunny Walk-animationen, mens de to andre mulige animationer er Jump og Attack.

På samme måde som med Mojo, bliver Jump-tilstanden aktiveret af en Trigger. Mens det er en regulær boolean der benyttes i til ”PlayerSeen”-parameteren. Grunden til det, er at når spilleren er erkendt, så skal NPC´en forblive i Attack-tilstanden indtil den ikke længere kan ses.



FIGUR 34 - EVILBUNNY´S CHARACTER CONTROLLER

For at kunne se spilleren har EvilBunny fået tilknyttet en Box Collider med IsTrigger-funktionaliteten slået til. Hvis en anden collider, som også har IsTrigger slået til, interagerer med den i Figur 35 viste kasse, så kan man via script registrere det.

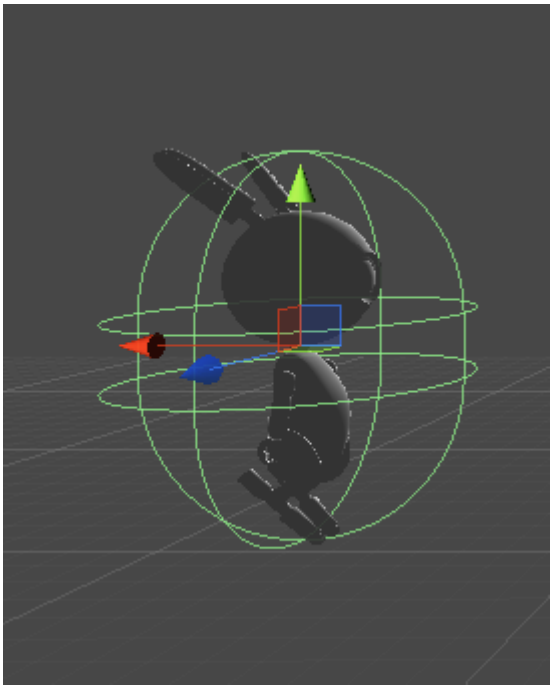


FIGUR 35 - BOX COLLIDER REPRÆSENTERER NPC´ENS SYNSFELT

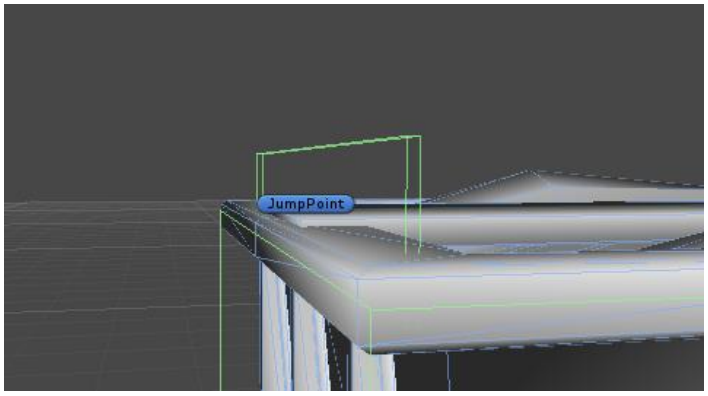
Udover at skulle være i stand til at kunne se, er det også en forudsætning for NPC´en at kunne opfange kollisioner på figuren selv. Den funktionalitet skal bruges for, at han kan hoppe imellem de forskellige platforme og for at opfange hvis han rammes af skud, eller af Mojo selv.

For at opnå dette er der tilknyttet endnu en collider med IsTrigger slået til, denne gang omkranser collideren kun lige figuren. (Figur 36) Scriptet refererer til denne collider, som NPC´ens: ”feels”. Det er med den han ”mærker” verden omkring sig.

For at AI´en kan hoppe på de rette steder, har vi til hver platform tilføjet en box collider i højre ende. (Figur 37) Denne collider er tagget med navnet: ”JumpPoint”, og da den også er en trigger, kan vi via script kalde en Jump-metode når NPC´en interagerer med et JumpPoint.



FIGUR 36 - EVILBUNNY OMKRANSET AF HANS *”FEELS”-COLLIDER*



FIGUR 37 - JUMPPOINT COLLIDER

Enemy AI-script

For at udnytte den funktionalitet der er til rådighed via de tilknyttede komponenter oprettes et nyt script, kaldet EnemyAI. I dette script implementeres den tilstandsmaskine som driver EvilBunny´s kunstige intelligens.

Første trin er at definere hvilke tilstande NPC´en kan befinde sig i. Af designdokumentet fremgår det, at den skal kunne bevæge sig imod Mojo, og angribe ham hvis han kommer ind i dens synsfelt. Det er givet at den ikke starter på samme platform som Mojo, da objektet der spawner NPC´erne er det samme som spawner platformene, derfor skal NPC´en kunne forcere distancen imellem platformene. Derudover skal NPC´en kunne dø, hvis han rammes af Mojo´s kugler.

Sammenlagt udmønter disse krav sig i de tilstande som er vist i Figur 38. Af figuren fremgår også tilstanden Init, som kun køres en enkelt gang som det første når objektet oprettes, og som har til formål at initialisere de variabler som anvendes så længe scriptet kører. Metoderne der benyttes til at fremfinde de forskellige komponenter er de samme som blev benyttet i PlayerScript.cs.

```
public enum State
{
    Init,
    Walk,
    Jump,
    Attack,
    Die
}
```

FIGUR 38 - AI´ENS MULIGE TILSTANDE

Figur 39 viser den tilstandsmaskinens primære cyklus. Start-metoden køres automatisk af Unity ved objektets oprettelse, men i hvor vi normalt benytter den til at initialisere scriptet, benyttes den her til at drive AI´en. For at kunne fungere som tilsigtet er det nødvendigt at while-løkken kører uendeligt, da det ikke er til at vide hvor mange gange den skal gennemgå cyklussen før NPC´en gøres inaktiv.

```
IEnumerator Start () {
   _state = State.Init;

   while (true)
   {
       switch(_state)
       {
           case State.Init:
               Init();
               break;
           case State.Walk:
               Walk();
               break;
           case State.Jump:
               Jump();
               break;
           case State.Attack:
               Attack();
               break;
           case State.Die:
               Die();
               break;
       }
       yield return 0;
   }
}
```

FIGUR 39 - TILSTANDSMASKINENS LØKKE

Normalt ville spillet (og Unity) standse og behøve manuel genstart hvis man implementerede en uendelig løkke, hvilket er hvad ”While (true)” reelt er. Men ved at ændre returtypen til en IEnumerator, og returnere Yield 0 undgås dette problem.

For hver gang while-løkken køres evalueres det hvilken tilstand NPC´en er i, og den relevante metode kaldes. Som det ses under metodesignaturen, sættes *_state*-variablen til Init ved første kald, derfra sættes *_state* til *Walk*, hvilket resulterer i at Walk() kaldes i den efterfølgende while-cyklus.

Walk tilstand

NPC´ens standardtilstand er at gå, hvilket også fremgår af Figur 34 derfor er det ikke nødvendigt at sætte ham i gang. Walk-metoden som vidst i Figur 40 indeholder et check af om NPC´en er død siden sidste kald. Hvis han er, så springes resten af metodens funktionalitet over, og tilstanden angives til Die.

Hvis det ikke er tilfældet kontrolleres det om spilleren kan ses, i hvilket fald animatoren skifter tilstand. Samtidigt angives det også at ved næste gennemløb NPC´en skifte til Attack-tilstand.

Det sidste check i metoden går på om NPC´en har ramt et JumpPoint. *_feels.NpcJump* er en property på npcFeels.cs, som sættes til true hvis NPC´ens *feels*-collider, som vist i Figur 36, har ramt en collider der er tagget som ”JumpPoint”.

```
private void Walk()
{
    if (!CheckForDeath()){
        if(_sight.playerInSight) {
            _anim.SetBool("PlayerSeen", true);
            _state = State.Attack;
        }

        if(!_sight.playerInSight)
            _anim.SetBool("PlayerSeen", false);


        if(_feels.NpcJump) {
            _feels.NpcJump = false;
                _state = State.Jump;
        }
    }
    else
        _state = State.Die;
}
```

FIGUR 45 - ’WALK’ TILSTAND

Attack-tilstand

Metoden der håndterer NPC´ens Attack-tilstand fremgår af Figur 41. Igen er der et check for at sikre at NPC´en ikke er død. Dernæst kontrolleres det igen om spilleren er i synsfeltet. Grunden til, at der det kontrolleres både i Walk() og Attack() er, at Attack() fortsætter med at blive kaldt så længe spilleren er hvor EvilBunny kan se ham. Derfor skal det checkes for hvert gennemløb, at spilleren rent faktisk kan ses, da der skal skiftes tilstand i samme øjeblik han forsvinder ud af synsvidde.

Hvis spilleren kan ses, standses NPC´ens bevægelse, og det kontrolleres om han er klar til at afgive skud. *_isWaiting* er til for at sikre at NPC´en ikke afgiver et skud for hvert gennemløb af metoden. Det ville

resultere i en endeløs strøm af kugler, hvilket ikke er hensigten. Coroutinen WaitForNextShot() som ses i Figur 42, sikrer at der går en prædefineret tidsperiode før der igen kan afgives et skud.

Kaldet til SpawnBullets-scriptet sørger for oprettelsen af en ny kugle. Dette script kan ses i dets helhed i bilaget, SpawnBullet-scriptet. Hvis spilleren ikke længere er synlig for NPC´en, ændres både animator- og AI-tilstand tilbage til Walk-udgangspunktet

```
private void Attack()
{
    if (!CheckForDeath()) {
        if(_sight.playerInSight) {
            _anim.applyRootMotion = false;
            if (!_isWaiting) {
                _isWaiting = true;
                _spawnBullets.Spawn();
                StartCoroutine(WaitForNextShot());
            }
        }
        else {
            _state = State.Walk;
            _anim.applyRootMotion = true;
            _anim.SetBool("PlayerSeen", false);
        }
    }
    else {
        _state = State.Die;
    }
}
```

FIGUR 41 - ATTACK-TILSTAND

```
IEnumerator WaitForNextShot()
{
    yield return new WaitForSeconds(timeBetweenShots);
    _isWaiting = false;
}
```

FIGUR 42 - VENTEPERIODE IMELLEM SKUD

Jump-tilstand

Som det fremgår af Figur 43, består Jump primært af et kald til en Coroutine kaldet WaitForDoubleJump(), som ses på Figur 44.

```
private void Jump()
{
    Debug.Log("Enemy Jump");
    if (!CheckForDeath()) {
        StartCoroutine(WaitForDoubleJump());
        _state = State.Walk;
    }
    else {
        _state = State.Die;
    }
}
```

FIGUR 43 - JUMP-TILSTAND

I WaitForDoubleJump-metoden får NPC´ens rigidbody tilføjet kraft til dens Y-akse, hvilket resulterer i et hop. Dernæst ventes der i 1,2 sekunder, før den får endnu en påvirkning. Denne gang med en halv gange mere kraft end første hop. Dette resulterer i en fin dobbelthop effekt, der sender NPC´en fra en platform til en anden.

```
IEnumerator WaitForDoubleJump()
{
    rigidbody.AddForce(Vector3.up * jumpForce);
    yield return new WaitForSeconds(1.2f);
    rigidbody.AddForce(Vector3.up * (jumpForce * 1.5f));
}
```

FIGUR 44 - NPC DOUBLE JUMP

Die-tilstand

NPC´ens Die-metode er til for at sikre oprydning efter objektet. I dette tilfælde er der kun tilbage, at sætte objektets tilstand til inaktivt. Efterfølgende det igen indgår i spawnpoolen.

```
private void Die()
{
    this.gameObject.SetActive(false);
}
```

FIGUR 45 - DIE-TILSTAND

CheckForDeath-metoden er vist i Figur 46. Det kontrolleres om npcFeels-scriptet har registreret en kollision med et GameObject der er tagget med ”projectile”. Hvis det er tilfældet er NPC´en at betragte som død.

Grunden til at at NpcHit sættes tilbage til *false*, er at NPC-objekterne oprettes ved object pooling. De variabler der er påvirket af NPC-objektet, bibeholder deres nuværende tilstand når NPC´en sættes til inaktiv. For at undgå, at NPC´en starter med NpcHit sat til *true* ved næste aktivering, sætter vi den med det samme til *false* her.

```
private bool CheckForDeath()
{
    bool dead = false;
    if(_feels.NpcHit == true) {
        dead = true;
        _feels.NpcHit = false;
    }
    return dead;
}
```

FIGUR 46 - KONTROLLER OM NPCFEELS HAR REGISTRERET EN KOLLISION MED ET PROJEKTIL

Procedural platformgenerering

En essentiel del af dette spil er den funktionalitet, som står for at skabe de objekter, spilleren bevæger sig på. Fokus ligger blandt andet på at skabe en autogenereret bane, som er tilfældigt udformet ved hvert gennemspil. Formålet er at holde spillet interessant, da banen ikke bliver indøvet. Samtidig effektiviserer det arbejdet ift. at opsætte hver enkelt bane, da alternativet er at placere hvert platform manuelt i samtlige baner. Selvom kompleksiteten ved at autogenerere platformene er højere, så er det stadig en mere effektiv proces.

I spillet består banen af tre forskellige typer platforme; start platformen, de mellemliggende platforme og den afsluttende platform.

- Startplatformen, som er den første platform i scenen, er altid tilstede, da den sikrer spillerobjektets startposition.
- De næste objekter er de mellemliggende platforme, som udgør størstedelen af banen. Disse bliver autogenereret sideløbende med spillerens fremgang i banen. De kommer herudover i tre forskellige typer, lav, medium og høj.
- Den sidste platform i banen er slutplatformen, hvor spillet afsluttes, når spilleren når denne.

Tilfældig generering af platforme

De platform-relaterede komponenter i selve scenen består af to objekter. Den ene er startplatformen, som er et simpelt objekt uden script-komponent pålagt, men med standard Transform-, Box Collider- samt Mesh Render-komponenter.

Det andet objekt er ”PlatformSpawn”, som står for at oprette de kommende platforme, som spilleren hopper på. ”PlatformSpawn”-objektet er igen et simpelt plane-GO, som altid er placeret uden for kameraets synsfelt. Tilknyttet er der to script-komponenter, ”SpawnScriptDistance” og ”SpawnEnemy”.

”SpawnEnemy” er det script, som står for at oprette fjender i banen, og vil derfor ikke blive redegjort for i dette afsnit. ”SpawnScriptDistance” omhandler baneopbygningen.

Dette script har to formål. Ét er at generere den række objekter, hvor den enkelte platform tilfældigt vælges ud fra de tilgængelige modeller. Herunder gøres der også brug af ”object pooling”-teknikken. Det andet er at oprette slutplatformen når den sidste autogenererede platform er sendt afsted.

Kigger man nærmere på scriptet gør det brug af fire metoder samt de indbyggede Unity-metoder, Start() og Update().

”Start”-metoden indeholder kald til metoderne ”FillPool()” og ”SpawnPlatform()”, hvor ”FillPool()” er metoden, som opretter en liste med objekterne, der udgør de autogenerede platforme. Efter ”FillPool”-metoden kaldes ”SpawnPlatform()”, som opretter objektet i scenen.

FillPool-metoden er den metode, som opsætter ’object pooling’.

Som ses i scriptet i Figur 47, oprettes først en liste til at opbevare pool-objekterne; *list*. Herefter oprettes hvert enkelt objekt i hukommelsen med tilhørende koordinater og rotationer i scenen, som er angivet ved ”transform.position” og ”Quaternion.identity”. Derefter sættes objektet til inaktiv, hvorefter det tilføjes til listen.

```
private void FillPool() {
  list = new List<GameObject>();
  for (int i = 0; i < pooledAmount; i++) {
    for (int n = 0; n < obj.Length; n++) {
        GameObject go = (GameObject)Instantiate(obj[n], transform.position,
            Quaternion.identity);
        go.SetActive(false);
        list.Add(go);
    }
  }
}
```

FIGUR 47 - FILLPOOL-METODEN

Variablen 'pooledAmount' bestemmer hvor mange objekter der skal oprettes til at indgå i "poolen".

Update-metoden er den metode, der bliver kaldt hvert frame i scenen, og er derfor hovedmetoden i dette script. Det første, som sker i metoden, er et kald til: "IsAllObjectsSpawned"-metoden, som tjekker på antallet af objekter placeret i scenen. Matcher dette antallet af objekter, som der er forventet, sætter den variablen "_allObjectsSpawned" til "true". Det bruges herefter i en "if"-løkke, som kalder metoderne, "SpawnPlatform" eller "SpawnUnique", som henholdsvis placerer det næste objekt i scenen, eller afslutningsplatformen afhængig af "_ allObjectsSpawned".

```
void Update () {
  IsAllObjectsSpawned();
  float distance = Vector3.Distance(this.transform.position, lastObject.transform.position);

  if(_allObjectsSpawned == false) {
    if(distance >= distanceBetweenObjects)
      SpawnPlatform();
  }
  else {
    if (distance >= distanceBetweenObjects)
      SpawnUnique();
  }
}
```

FIGUR 48 - UPDATE() KØRES ÉN GANG PR. FRAME

Update-metoden har endvidere en attribut, "distance", som er afstanden imellem det nuværende objekt og det sidst placeret objekt. Dette tal bruges til at kontrollere afstanden fra den sidst placerede platform og så til "PlatformSpawn"-GO. Når den distance som er givet ved "distanceBetweenObjects" nås, kaldes metoden der opretter en ny platform. Vi ved at alle platformene er lige lange, derfor kan vi bruge denne metode til at sikre, at der altid er samme afstand imellem dem.

"SpawnPlatform"-metoden er den metode, som vælger et tilfældigt inaktivt objekt i listen og sætter det til "aktivt". Konkret løbes alle objekterne i "poolen" igennem indtil der findes et inaktivt. Dette inaktive objekt gemmes i variablen "_lastObject" og sættes til "aktivt". Hvilket, fordi det allerede er indlæst, vil sige at det, og dets komponenter, bliver aktive i scenen.

Efterfølgende tælles "_spawnedObjects" op, og variablen "found" sættes til "true" så løkken afsluttes.

```
private void SpawnPlatform() {
  bool found = false;

  while (!found) {
    int i = Random.Range(0, list.Count);

    if (!list[i].activeInHierarchy) {
      _spawnedObjects++;
      lastObject = list[i];
      list[i].transform.position = this.transform.position;
      list[i].SetActive(true);
      _spawnedObjects
      found = true;
    }
  }
}
```

FIGUR 49 - SPAWNPLATFORM-METODEN

Den sidste funktion i dette script er "SpawnUnique", som er den metode der placerer slutplatformen i spillet, hvorefter der slukkes for SpawnPlatform-GO´et. Dette script kaldes når "spawnedObjects" er lig det antal som er givet som længden på banen.

```
private void SpawnUnique() {

  Instantiate(uniqueObject, transform.position, Quaternion.identity);

  this.gameObject.SetActive(false);

}
```

FIGUR 50 - SPAWNUNIQUE-METODEN

Perspektivering af proces og produkt

Ved afslutningen af projektperioden har vi gennemført nogle overvejelser omkring den arbejdsproces som vi har gennemgået. Deriblandt også de tekniske overvejelser og løsninger er implementeret i produktet. I dette afsnit bliver der redegjort for processen i form af en gennemgang af sprint 1 til 3, hvor forløbet og fremgangen er nedskrevet.

Herefter bliver produktet sat i sammenligning med alternative løsninger på flere områder. Det første er de designvalg, som er blevet træffet igennem processen, hvor overvejelser som genre, platform mv. tages i betragtning. Sidst dykker perspektiveringen ned i nogle af de designvalg, som har resulteret i det endelige produkt.

Sprint 1

Det første sprint i dette forløb startede naturligt ud med at redegøre og opsætte produktbackloggen samt de grundlæggende værktøjer, som bruges i Scrum og XP. I designfasen af dette projekt blev der redegjort for de features, som produktet skulle indeholde, hvilket i dette sprint blev omdannet til stories. Velocity, burndown chart mv. blev sat på plads og backloggen for det første sprint blev færdiggjort. Burndown-grafen for det første sprint ses i bilagslisten under 'Sprint 1'. Sprintbackloggen for det første sprint ses her:

- Player, Løb – 2 point
- Player, Hop – 5 point
- Player, Slide – 3 point
- Player, Opret- ½ point
- Level, Random platform generator – 2 point
- Level, Random baggrunds generator – 1 point
- Level, Baggrundslag – 2 point
- Level, Kamera - ½ point
- Level, ObjectPooling – 2 point
- Controls, mobil – 5 point

Med to udviklere på backloggen, blev opgaverne fordelt. En udvikler tog sig af Player-delen, hvor der skulle bygges funktionalitet omkring at få spilleren til at bevæge sige. Den anden havde fokus på den grundlæggende funktionalitet i banen. Begge disse emner var prioriteret højst i produktbackloggen, da de udgør den grundlæggende funktionalitet i produktet og samtidig gør den videre udvikling lettere.

For at kunne påbegynde Player-funktionaliteten var der brug for undersøgelse af emnet, da der ikke var erfaring med brug af Mecanim-teknologien. Der var problemer med placeholder-animationerne, som ikke passede til vores brug. Dette bremsede udviklingen, og selvom der var forventet en spike på dette område, så tog funktionaliteten længere tid end forventet.

Dog lykkedes det i sidste ende at få de grundlæggende bevægelser på plads, men da de endelige animationer ikke er færdiggjorte, har det ikke været muligt helt at afslutte player-delen og kræver mere arbejde længere henne i processen. Ved slutningen af dette sprint blev Player-delen nedskrevet da de sidste tilretninger mv. først kommer senere i forløbet.

Udviklingen med banen gik lettere end forventet, og det var muligt at nedskrive hele Level-delen hurtigere end estimeret. Arbejdet bestod af nogle relativt små spikes, omhandlende tilfældig generering af banen samt generering af baggrunden i banen. Dog er der igen brugt placeholder elementer, og det betyder igen tilretning senere i forløbet.

Sprint 1 - Retrospective
Alt i alt er det første sprint vellykket, da det har været muligt at nedskrive alle stories i sprintbackloggen, selv med spikes i forløbet. Dette gav mulighed for at finpudse og optimere den nuværende funktionalitet og inddrage nye stories i sprintet.

Her blev der fokuseret på at imødekomme de spikes som vi forventede ville opstå i det efterfølgende sprint. Derudover har det givet mere plads til at dokumentere funktionaliteten i rapporten. Vi har desuden overvejet den måde vi estimerer story points på, da der har været væsentlige afvigelser imellem estimat og forbrugt tid på mange stories i dette sprint. Det vi ved nu, er at det er relativt trivielt at lave en proceduralt genereret bane, mens det modsatte er tilfældet for oprettelsen af en spilbar karakter.

Sprint 2
I dette sprint har fokus hovedsagligt ligget på at oprette en funktionelt menusystem, samt at modellere hovedpersonen og animere bevægelserne. Derudover blev der brugt ressourcer på at optimere Mecanim. Ugen startede ud med en kort opsætning af backloggen, samt uddelegering af arbejdsopgaver. Menu systemet som skulle oprettes består af en brugergrænseflade hvor det er muligt at klikke sig rundt imellem de forskellige menuer.

Animationsdelen betod af at modellere en 3D-figur i Blender for derefter at animere nogle grundlæggende animationer, som skulle bruges i spillet. Der var forventet at der skulle bruges rigeligt tid på begge, da der ikke var gjort erfaringer med teknologien før af de enkelte udviklere. Dette resulterede i følgende sprintbacklog:

- Lav hovedmenu – 5 point
- Lav 'Vælg bane' – 5 point
- Lav 'tutorial' – 2 point
- Lav 'indstillinger' – 2 point
- Lav 'Credits' – ½ point
- Lav Exit-knap – ½ point
- Lav 3D-model – 3
- Lav hop-animation - 2
- Lav dobbelthop-animation - ½
- Lav slide-animation - 2

Med 22,5 story points i alt var der lidt spillerum, da både menuen og animeringen havde risiko for spike. Sprintets fremgang ses i bilagslisten under 'Sprint 2'. Opgaverne blev fordelt med menu-funktionaliteten til en udvikler og 3D-animeringen til en anden.

At skabe et menu system viste sig, i sig selv, at være relativt simpel opgave. Det var ikke et problem at få en forståelse for Unity´s GUI-system. Der blev hurtigt skabt en prototype, men da løsningen i første omgang var statisk, gav det ingen mulighed for at tilpasse brugergrænsefladen til forskellige skærmstørrelser.

Herefter fortsatte arbejdet med at gøre scriptet mere generisk og dynamisk. Arbejdet skred støt, men langsomt fremad, da Unity3D har meget få værktøjer til at håndtere relative positioner på skærmen. Det betød at alle knapper mv. skulle opsættes manuelt, hvilket øger opgavens længde drastisk. Imidlertid lykkedes i sidste ende at nå frem til et tilfredsstillende resultat inden for sprintets løbetid.

De animationer der blev produceret var af en så ringe kvalitet, at de end ikke kunne bruges i prototypesammenhæng. Det vi fik ud af det, var en grundlæggende forståelse for hvordan 3d-modellering foregår, samt hvordan animationer kan optages.

Sprint 2 – Retrospective

Sprint 2 har kun været en begrænset succes. Menu-opgaven kunne nedskrives helt og er dermed, i prototypesammenhæng, ude af verden. Animationsopgaven har været en tung opgave, som har derfor opslugt meget tid uden at resultere i noget brugbart. Da prototypen skal afsluttes inden for endelig tid, blev det besluttet at opgive egenproduktionen af denne type assets indtil vores grafiker kan få tid til at løse den opgave. I stedet har vi anvendt en gratis-tilgængelig løbeanimation til vores hop. Det resulterer i en række fejl i forhold til vores character controller, men prototypen er spilbar med denne løsning.

Det positive ved, at have investeret tid i modellering og animationsarbejder er, at vi som programmører har lettere ved at snakke sammen med den, og på længere sigt; de, personer som har det primære ansvar for at producere animationer og modeller. Som vi tidligere har skrevet, så er god planlægning et af de områder vi som vi vil gøre til en kernekompetence. Det kan vi kun hvis vi forstår hinanden på tværs af afdelinger, og den forståelse er denne type spike med til at styrke.

Generelt var estimeringen bedre i dette sprint, i hvert fald i forhold til menu systemet. Teknisk set nåede vi også at lave animationerne. De kunne bare ikke bruges.

Sprint 3

I dette sprint er fokus så småt flyttet fra den grundlæggende funktionalitet i spillet til det omkringliggende features, som er nødvendigt for en velfungerende applikation. I selve spillet har der været fokus på at oprette den funktionalitet, som håndterer fjenderne i banen. Endvidere er skydefunktionaliteten sat i fokus for at afrunde de grundlæggende features. Sidst er der lagt energi i at optimere og udrede de problemer, som er opstået i forbindelse med udviklingen af animationerne.

Ugen startede som sædvanligt ud med en opsætning af sprintbackloggen, samt uddelegering af arbejdsopgaver.

- Lav skyd-funktionalitet – 2 point
- Lav forhindringer i banen – 3 point
- Lav fjender – 8 point
- Lav Game Manager – 1 point
- Lav score counter – 2 point
- Lav en ’save state’-funktionalitet – 2 point
- Lav en ’load state’-funktionalitet – 2 point

I alt 20 story points er backloggen en smule kortere, igen i forventning om uforudsete spikes. Denne gang forventet på ’Lav fjender’, da denne story er på 8 point. Samtidig giver det spillerum til optimeringen af spillerobjektet, da styring indeholder et antal bugs. Sprintets forløb fremgår af burndown-grafen for sprint 3, som ses i bilagslisten.

Skydefunktionaliteten var en relativ simpel opgave, hvor et projektil skulle sendes afsted fra spilleren ved et tryk på en knap. Arbejdsprocessen gik flydende, da dele af funktionaliteten kunne etableres ved hjælp af allerede eksisterende kode.

Den anden store opgave, som har været i fokus, har været at oprette fjenderne i banen. Opgavens omfang gjorde, at vi fra starten tildelte den flest story points. Det første mål i featuren var at oprette fjende-objektet i scenen. Desuden skulle vi implementere en kunstig intelligens som gjorde det muligt for fjenden at bevæge sig platform til platform, hvilket betyder, at der skulle oprettes en hoppe-funktionalitet, som aktiveres ved slutningen af en platform. Et andet mål var at få fjenden til at reagere på spilleren, ved at skyde imod ham. I den sammenhæng lykkedes vi med at få skrevet scripts der er generiske nok til at kunne bruges uændret på spilleren og på fjenden.

Til gengæld er der ikke afsluttet flere stories i dette sprint, simpelthen fordi vi ikke har brugt den tid på udviklingsdelen, som ellers er forudsætningen for vores estimering. Dette blev erkendt ved afslutningen af

sprintets første uge, hvorefter vi har skiftet fokus til i stedet at benytte udviklingstiden til at afrunde de features som er implementeret i prototypen.

Sprint 3 - Retrospective

Sprint 3 har altså været en delvis succes. De features, som blev påbegyndt, blev afsluttet relativt effektivt, men grundet omprioritering af udviklingstid er resten af de øvrige stories flyttet til sprint 4.

Perspektivering af projektprocessen

I dette projektforløb var der fra starten fokus på udvikling vha. Scrum og Extreme Programming, som er velkendte udviklingsmetoder inden for softwareløsninger. Gruppen lagde sig hurtigt på iterative udviklingsmetoder, dels pga. gruppens tidligere erfaring med disse, men også grundet produktets tekniske natur.

Da produktet indeholdt en lang række nye elementer, som kunne resultere i spikes, var det nødvendigt med en arbejdsmetode, som tillod os at vende tilbage til en funktion, efterhånden som andre funktioner blev klargjorte og udviklet.

Den alternative tilgang ville eksempelvis være en vandfaldsmodel, hvis tilgang bevæger udviklingen fremad en fase af gangen. Da den grundlæggende vandfaldsmodel samtidigt tilskriver, at det ikke bør være muligt at vende tilbage til afsluttede opgaver, stiller den høje krav til den tekniske formåen som er påkrævet for at designe den komplette tekniske løsning. Det blev derfor vurderet til at modarbejde den teknisk udforskende tilgang, som projektet forventedes at have.

For at strukturere opgaverne forbundet med projektet, blev der gjort brug af den online opgavemanager: Trello. Målet med at benytte opgavemanageren var at skabe en simpel oversigt over de forskellige prototypefeatures samt skriveopgaver, som skulle udføres i forbindelse med projektet. Programmet blev valgt grundet tidligere erfaringer og har opfyldt de behov, som gruppen har haft. Alternativet ville være et fysisk oversigt med tilhørende post-its, der administreres i udviklingslokalet, men da udviklerne ofte arbejdede fra flere lokationer, var online tilgængelighed et krav til opgavemanageren.

Arbejdsfordelingen har hovedsagligt bestået af individuelt opgaveløsning, hvor en udvikler har ansvaret for at udvikle en given feature. Formålet med denne fordeling var at optimere produktionen, hvor en relativ stor del af udviklingsarbejdet bestod i af undersøge og lære en given teknologi at kende. Det er en udfordring med denne fordeling, at udviklerne ikke altid er i stand til at supplere hinanden med viden og erfaringer. Vi kunne have benyttet parprogrammering specifikt for at håndtere dette. Men det ville heller ikke være uden problemer, jf. kapitlet om XP. I øvrigt forhindrer arbejdsfordelingen ikke udviklerne i at hjælpe hinanden efter behov, så den endelige arbejdsmetode gør brug af både individuelt arbejde og parprogrammering. For at kunne arbejde mest effektivt uafhængigt af hinanden, har vi i høj grad benyttet os af, at Unity er et komponentbaseret værktøj hvor mange komponenter kan opsættes og udvikles uden at andre nødvendigvis er klart til brug.

Perspektivering af produkt

Under udviklingen af dette spil er der truffet en række designvalg, som har resulteret i den endelige udformning af produktet. Et af de første valg, som blev truffet var brugen af 3D-modeller i scenen, hvor alternativet var en 2D-verden. Dette har givet nogle udfordringer ift. de 3D-modeller, som kræver animationer. Her blev det dog vurderet fra udviklernes side, at brugen af 3D-modeller ville give spillet mere værdi. 3D-modellerne er alle udviklet af virksomhedens grafiker, og animationerne følger som standard med Unity3D.

Samtidigt er det også et grundlæggende element i moderne spiludvikling, hvilket medfører at det er en øvelse som virksomheden skal mestre, for at opnå succes. Brugen af 3D medfører også, at der bliver stillet størrer krav til optimering af disse modeller grundet den begrænsede ydeevne i mobile enheder.

Et andet grundlæggende designelement er opbygningen af banen, hvor målet var at skabe en selvgenerende bane med forskellige platforme. Selvom denne løsning er mere kompleks end den alternative løsning, hvor hver enkelt platform opsættes individuelt, så er autogenerering af platformene mindre tidskrævende da vi let kan lave et stort antal forskellige verdener med denne metode, når komponenterne først en gang er lavet. Samtidigt ligger der også en gameplay-mæssig gevinst ved denne metode, da spilleren altid bliver præsenteret for variation i spillet.

Konklusion

Dette projekt har omhandlet to problemstillinger, hvordan man opstarter en bæredygtig spilvirksomhed, og hvordan man designer og udvikler et salgsværdigt spil.

For at svare på det første spørgsmål blev der først dykket ned i de grundlæggende tal omkring opstart af ny virksomhed. Der blev redegjort for hvilke ressourcer, som virksomheden har til rådighed fra begyndelsen, virksomhedens evner og kompetencer og sidst virksomhedens situation i form af det eksterne miljø.

Herefter blev der udarbejdet et etableringsbudget, som redegør for de grundlæggende omkostninger ved virksomhedsopstart. Budgettet udregnes til et relativt lille beløb på ca. 35.000 kr., som er realistisk for virksomheden at overkomme uden behov for ekstern finansiering. I længden bliver virksomheden finansieret igennem forskellige salgskanaler, herunder salg af spillet selv, reklameindtægter og køb i spillet. Markedet er en enorm industri med rig mulighed for at komme ind, men hvor risikoen for at forblive uopdaget er høj, da selve mængden af konkurrerende produkter er massiv.

Kernekompetencerne i virksomheden bunder i indsigt og erfaring med design af computerspil, samt tekniske kompetencer som kan udvikles til at bære den kompleksitet som er forbundet med spiludvikling.

Derudover har virksomheden en høj fleksibilitet i form af udviklingsværktøjer samt udviklingstid. På virksomhedens svage side har udviklerne ingen erfaring med drift af virksomhed. Ikke desto mindre ser vi gode muligheder for at opstarte og drive virksomheden selv med få ressourcer, blot vi er opmærksomme på hvor afhængige vi er af at kunne skille os ud fra mængden.

Under udviklingen af spillet har fokus hovedsagligt været på at skabe et simpelt spil, som kan håndteres af så gamle mobilenheder som muligt. Det har betydet en række tiltag i udviklingen for at optimere kravene til hardwaren herunder optimering af de forskellige grafiske indstillinger, lysindstillinger og oprettelse af objekter i scenen. Samtidig har det været væsentligt for os, at tage højde for forskellige de skærmstørrelser som mobilenheder kommer med.

Endvidere er der taget en række beslutninger omkring design af produktet, som har indvirkning på både markedsføring af spillet, men naturligvis også udviklingen af spillet.

Udviklingsprocessen er foregået under tilpassede versioner af metoderne Scrum og XP, som har givet en god grundstruktur igennem forløbet. Dette har givet en mere overskuelig udviklingsproces, hvor de enkelte funktioner omskrives til stories og uddelegeres iblandt udviklerne.

Selve udviklingen har omhandlet en række emner, som udgør hovedfunktionaliteten i spillet. Den vigtigste og mest ressourcekrævende funktionalitet er spillerobjektet, som gør brug af adskillige forskellige typer komponenter. Det har været en krævende proces, da animationssystemet har voldt en del problemer. Resultatet i prototypen er ikke absolut tilfredsstillende, men det skyldes i høj grad, at vi er tvunget til at gøre brug af midlertidige animationer. Der er god grund til at forvente væsentlige forbedringer når de endelige assets bliver færdiggjort, i og med at Mecanim-funktionaliteten er bundet op på animationernes gennemløbshastighed.

Et andet stort element i spillet er genereringen af banen, som naturligvis er en essentiel del af produktet. Her var der under udviklingen lagt vægt tilfældighedselementet, som skulle sikre en frisk bane hver gang. Resultatet er blevet generering af en tilfældig type, og størrelse platform, som er optimeret til brug på mobile enheder. Da løsningen samtidig er generisk nok til at kunne anvendes med andre typer modeller, kan vi let oprette nye baner med de samme komponenter, hvorfor opgaven må siges at være løst.

Sidst er der de omkringliggende features, som er blevet afsluttet, herunder fjenderne i banen, menu-systemet og bevægelige baggrunde. Alt i alt er de grundlæggende rammer for spillet implementeret på en måde så det er let, at anskueliggøre overfor udenforstående hvad visionen for spillet er på længere sigt.

Projektet har givet et dybere indblik i at starte og drive en virksomhed i spilindustrien. Endvidere har det været en kilde erfaring inden for spiludvikling med Unity3D, da nye emner og konstant er blevet udforsket i løbet af projektperioden. Det er hensigten at fortsætte udviklingen af produktet, forsat ved brug af Unity3D, samt yderligere at undersøge mulighederne for støtte til opstart af virksomhed, både i form af rådgivning, og finansielt.

Bilag

Ordforklaring

AAA – Udtales ”tripple A”, benyttes som et udtryk for de spil der har de højeste budgetter, største teams og mest markante markedsføring.

Asset – Dækker over et stykke grafik, script eller lignende som indgår i spillet.

Antagonist – De(n) som spilleren er i mod. Alle fjenderne.

Content – Betyder indhold, og bruges specifikt til at henvise til en del af spillet.

Endless Runner – En variant af platformsgenren hvor protagonisten løber endeløst ud af en bane, uden mulighed for at stoppe, eller sætte hastigheden ned, mens han (m/k/u) undgår forskellige forhindringer og samler point sammen indtil han dør. Typisk øges hastigheden langsomt hen af vejen. Temple Run og Jetpack Joyride er populære eksempler på hhv. en 3D og 2D endless runner.

Feature – En specifik game mechanic.

Free-To-Play – Betalingsmodel hvor spillet I sig selv er gratis, og hvor udvikleren I stedet bliver betalt via reklamer, eller mikrotransaktioner.

Gameplay – Refererer til de features som spilleren har direkte til rådighed. F.eks. styringen af protagonisten.

Game Design Document (GDD) – Er et levende dokument der indeholder alle oplysninger om spillet. Dvs. En beskrivelse af spillets genre, spillets stil, hvilke features der indgår mv.

Game Engine – Er et system der bruges til udvikling af spil. Formålet er at forenkle spiludviklingen ved f.eks. at give adgang til en grafisk scene editor, indbyggede fysik-klasser, multiplatform kompilerer osv. Unity er en game engine.

Game Mechanics – Refererer til de regler som spillet er bygget op omkring. I vores spil er det en væsentlig game mechanic at protagonisten kan hoppe fra platform til platform, og at han dør hvis han ikke gør det.

In App Purchases – Bruges sammen med mikrotransaktioner, og er en ”butik” i spillet hvor brugeren kan købe virtuelle dele til spillet for rigtige penge.

Indie – Bruges enten om indie-spil, eller indie-udviklere. Det er en forkortelse for ”Independent”, og refererer til spil eller udviklere som er produceret uden støtte fra en udgiver. Normalt er det ensbetydende med mindre teams, mindre budgetter og viral markedsføring.

MMO – Massively Multiplayer Online. Et online multiplayerspil med en persistent verden.

Model – Refererer til et 3D objekt. Vores protagonist er en model, ligeså vores togvogne etc.

Monetiseringsstrategi – Fordanskning af monitization strategy. Dækker over de forskellige metoder man kan gøre brug af for at tjene penge på et produkt. Typisk menes der i forhold til

Free-to-Play-produkter, men i princippet dækker det alle salgsmetoder.

NPC – Non Playing Character. Termet refererer til enhver karakter i spillet som er styret af en AI. Modsætningen er Player Character, som er enhver karakter der er styret en spiller.

Placeholder – Kan være en texture, en model, en lydeffekt eller et andet spilelement, som optræder i stedet for det endelige asset. Formålet er at kunne bygge en prototype af spillet op uden at have de endelige assets klar på forhånd.

Platformsspil – En klassisk spilgenre, hvor spilleren har som mål at traversere en bane med forskellige fjender og forhindringer. I modsætning til en endless runner kan spilleren normalvis stoppe op, gå bagud i banen mv. Super Mario er måske det mest berømte eksempel på genren.

Mikrotransaktioner – Salgsmetode hvor brugeren køber mindre forskellige goder til brug i spillet for rigtige penge.

Playtest – Testmetode hvor man tester spillet ved simpelthen at spille det, og forsøge at "ødelægge" spillet via brug.

Proceduralt generet – Betyder at noget content er genereret ud fra en algoritme, og altså kan ændre sig fra spil til spil, baseret på den algoritme.

Protagonist – Vores hovedperson, den som spilleren styrer.

Script – Er et stykke kode der skrives oven på den game engine der benyttes, med det formål at manipulere spillets objekter. Konkret dækker det over al vor kode.

Scope – Relaterer til projektets størrelse. F.eks. ville det være ude af scope hvis vi besluttede at selvkomponere al musikken i spillet, eller lave motion-capture til vores animationer.

Skin – I denne sammenhæng refererer skin til en models udseende. F.eks kunne vi give vores protagonist et Skovhugger Skin, et Gangster skin, eller noget helt tredje. Det er altså et spørgsmål om at skifte figurens textures. Skins sælges ofte via IAP.

Standalone klient – En spilklient som kan afspilles på f.eks. PC uden at køre i en emulator eller igennem en webside (som f.eks. en Facebook app)

Texture – Er et billede som "beklæder" et objekt.

Game Design Dokument

Game Design Dokument
(Zen Rabbit)

Version: 1.0

Dato: 24/4/2014

Sidst opdateret:08/6/2014

Genre
Endless runner / actionplatform

Historie
Rumkaninen Mojo er som udgangspunkt en afslappet type. Siden han trak sig tilbage fra jobbet som rumsheriff har han levet en afsondret tilværelse i en lille flække langt ude i rum-vesten. Her bruger han tiden på stille meditation, og på at gro sine bonzai-træer. Idyllen brister dog da (rum)skurken Geraldine og hendes fæle håndlangere fælder Mojo´s træer for at rum-baronen Grimaldi kan få frit spil.
Mojo ser rødt, og kaster sig ud i et hovedkulds hævnorgie.

Karakterer
Mojo – Helt og rumkanin, tidligere lovmand. En kanin i balance, indtil balancen bliver brudt. Nu er han en kanin med en mission!

BunnyBot 1.0 – Masseproduceret bandit kanin. Standardfjende.

Grimaldi - Ond rumbaron og end boss.

Core gameplay
Spillet er delt op i 2 dele; platform og showdown.

I ”platform”-delen løber frem igennem banen. På sin vej skal han dukke sig for visse forhindringer, hoppe over andre, samle powerups op, og nedkæmpe fjender.

”Showdown”-mode er en bossfight, hvor spilleren via en serie quick-time-events skal nedkæmpe en særlig boss. Showdown fungerer som afslutningen på hver separat verden.

Egenskaber
I platform-mode kan Mojo løbe, hoppe, dobbelthoppe, slide og skyde.

I Showdown-mode kan han kun trække pistolen og skyde.

Game over
Hvis spilleren taber kampen resettes han til starten af banen.

Cutscenes
Enkeltside tegneserier der fortæller en historien kort, som spillet skrider frem.
Der skal være en efter hver verden, der afslutter den man kommer fra, og introducerer den nye.

Miljø
Hver ”Verden” i spillet har sit eget tema, hvor den røde tråd er fantasy steampunk. Den første verden vi bygger

Level oversigt
Baner er mindre enheder, som der er 5 af per verden. Hver verden afsluttes med en boss.
Hver verden har et overordnet tema, f.eks. western, luftskibe, mv.

Points
Spilleren får point for at skyde fjender, samle powerups op, og samle frø sammen.

Kamerasetup
Hovedparten af spillet, platform-mode vises fra siden i 2.5D. Dvs. forgrunden, hvor spilleren bevæger sig er i 3D, mens al baggrund er i 2D.
I Showdown-mode er kameraet placeret omkring hoftehøjde, og lidt bag ved spilleren.

Spilkontrol
Spilleren bevæger sig fremad af sig selv, og spilleren har ingen mulighed for at stoppe op. Dog kan visse powerups ændre i spillerens hastighed.
Spilleren kan dukke sig, hoppe og dobbelt-hoppe ved at lave swipe på skærmen, henholdsvis i op- og nedadgående retning. Dette foregår på skærmens venstre halvdel
Desuden kan spilleren bruge sin kamp-egenskab ved at trykke på skærmens højre halvdel.

Spil varighed
En bane tager 1-2 minutter at gennemføre.

Målgruppe
Spillet udvikles til mobile enheder, og sigter efter målgruppen 15-25 årige mænd og kvinder. Dog med en primær fokus på kvinder jf. markedsanalysen i rapporten.

Prissætning
Spillet er gratis med reklamer, og IAP. Der vises en reklame efter hvert tabt spil. Derudover har spilleren mulighed for at købe nye ”skins” til mojo, samt nye verdener.

Teamoplysninger
Ronnie Hemmingsen
Nicholas Alberg
Toke Olsen

UI Layout

Farveskema
Farveskemaet er de generelle farver, som går igen igennem spillet. Farverne har stor betydning på stemningen af spillet, og det er derfor en god praksis at have redegjort for dette.

I dette projekt er der fokus på en let og humørfyldt stemning, hvilket leder op til lyse og friske farver. En grundlæggende designregel for grafisk design er ’60-30-10’-reglen, som henviser til den praksis, hvor grundfarven udgør 60% af designet, den sekundære farve udgør 30%, og accent farven udgør de sidste 10%. Omgivelserne for spillet er ”steampunk”, som er kendt for sit messingfarvet miljø, og den matte mørkegule farve er derfor være en oplagt grundfarve. Spillet bevæger sig i igennem flere verdener, og indeholder derfor et tilsvarende antal forskellige farveskemaer, men grundfarven bør være gennemgående i alle verdener.

Hovedpersonen, som er Mojo, er en farverig og tegneserieagtig karakter, og derved tilfalder farverige og kontrastfyldte farver naturligt til ham. Samtidig er det vigtigt hovedpersonen skiller sig ud. De specifikke farve varierer fra verden til verden og eventuelt ud fra brugerens valg.

Fjenderne i spillet bør stå i kontrast til hovedpersonen. Det vil her sige fladere, mørkere farver, som ligger i den anden ende af farvespektrummet. ’Fjendtlige’ farver består typisk af sort for død, rød for blod og fare og gul for advarsel, hvor mindst én af disse burde indgå i fjendernes farveskema.

Titelskærm
Spillet skal indeholde en startskærm, som vises når spillet startes op. Startskærmen består af et billede, som gengiver essensen af spillet.

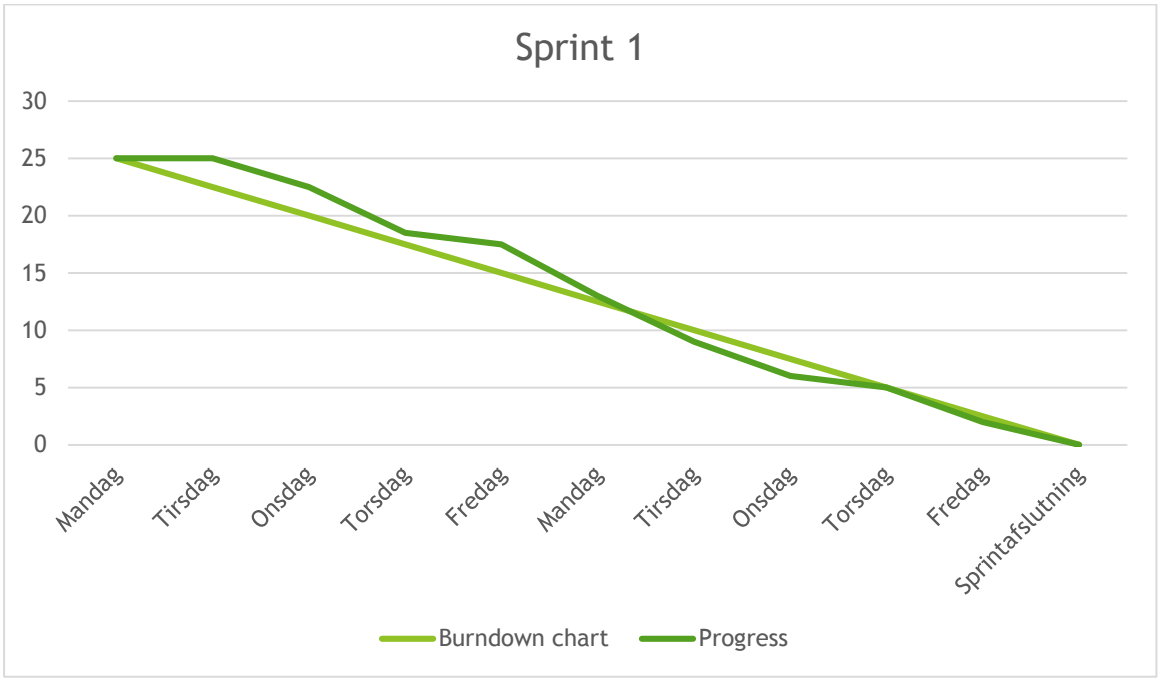
Menuskærmen
Hovedmenuen samt ’Options’-menuen er en traditionel menu med en antal punkter. Disse punkter skal være centreret og på række og derved overholde loven om nærhed. Endvidere kommer loven om lighed i kraft, da knapperne skal være udformet ens med samme skrifttype. Sidst er menupunkterne indkapslet i en boks, hvorved loven om lukkethed gør sig gældende.

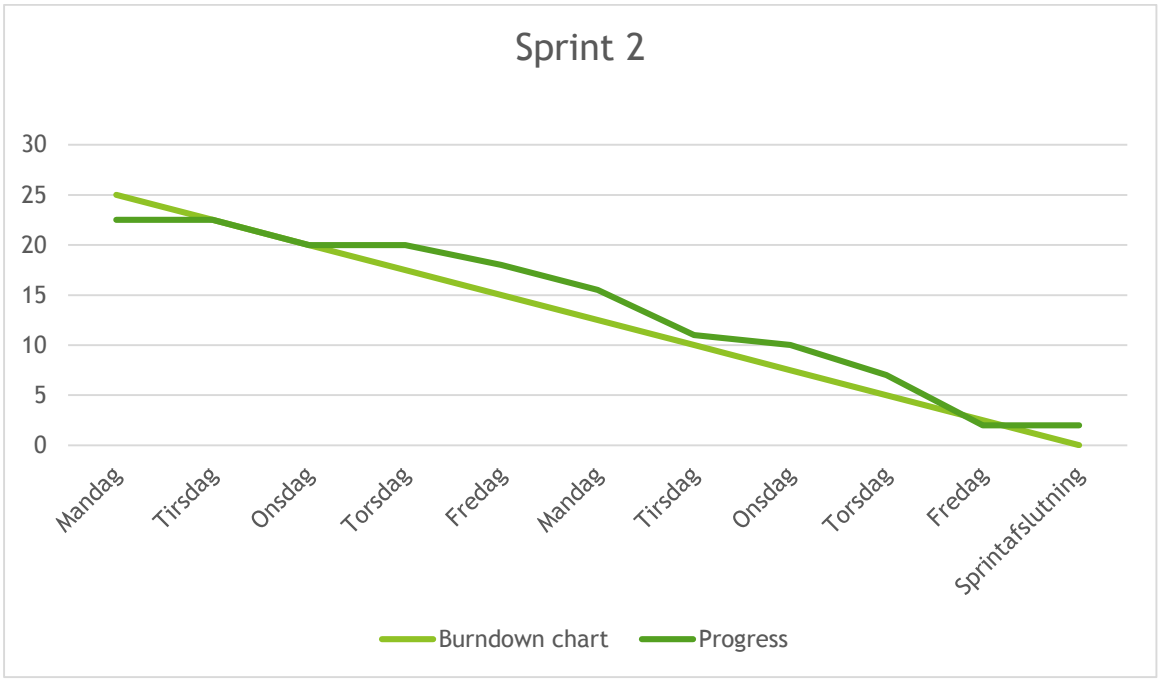
Banemenuen består af tre bokse, hvor hver boks indeholder yderligere tre små knapper og en enkelt stor. Her kommer igen flere af gestaltlovene i spil, da boksene overholder loven om lukkethed og nærhed. Endvidere skal de tre grundlæggende bokse stå på linje, hvorved loven om lighed gør sig gældende.

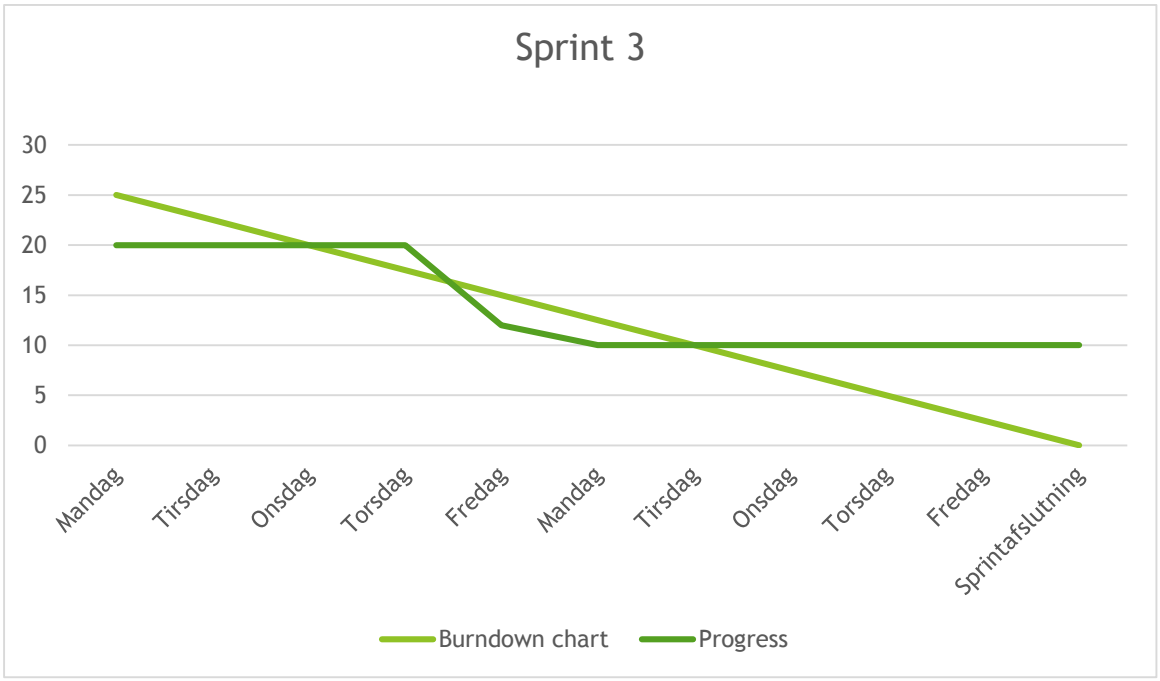
Tutorial
’Tutorial’-menuen skal indeholde omkring tre til fire billeder, hvorpå spillets instrukser står skrevet. Et enkelt klik på billedet skifter til næste i rækken og dirigerer tilbage til hovedmenuen når afsluttet.

Credits
Menupunkt med angivelse af hvem der har været involveret i projektet, og eventuelle licenser der er benyttet.

Burndown charts






Sprint 3
30
25
20
15
10
5
0
Mandag
Tirsdag
Onsdag
Torsdag
Fredag
Mandag
Tirsdag
Onsdag
Torsdag
Fredag
Sprintafslutning
Burndown chart
Progress

Playtestskema

	Funktion, der testes	Godkendt	Kommentar
Menu	Hovedmenu – menupunkterne		
	New level / Continue – starter knappen et nyt spil, hvis intet spil findes i forvejen		
	New level / Continue – Fortsætter knappen fra sidst afsluttede punkt i spillet		
	Level Select – banepunkterne		
	Level Select – frem/tilbage		
	Tutorials – Forstås instruktionerne		
	Tutorials – Skiftes billederne korrekt		
	Options – Virker lyd on/off		
	Options – Virker music on/off		
	Options – reset progress		
	Credits – Er informationen relevant og overskuelig?		
	Exit – afslut spillet		
Mojo	Løb		
	Hop		
	Dobbelt hop		
	Slide		
	Skyd		
	Saml PowerUp op		
	Dø – Falde		
	Dø - Forhindring		
	Dø – Fjende		
	Vind banen		
Banen	Er startplatformen tilstede		
	Autogenereres de kommende platforme		
	Virker sorteringen af platforme tilfældig		
	Genereres den sidste platform		
	Genereres PowerUps		
Diverse	Bevæger fjenderne sig		
	Tæller scoren fornuftigt		
	Påvirker PowerUps Mojo		

Kildeliste

[1] »Wikipedia,« [Online]. Available: http://en.wikipedia.org/wiki/List_of_game_engines.

[2] Crytek. [Online]. Available: Cryengine.com.

[3] [Online]. Available: http://forum.unity3d.com/threads/103284-Unity-Vs-Cryengine-3.

[4] Crytek. [Online]. Available: http://www.crytek.com/news/crytek-announces-its-cryengine-as-a-service-program-for-990-usd-per-month.

[5] »Unity3D.com,« [Online]. Available: http://unity3d.com/unity/collaboration.

[6] »Github.com,« 3 April 2014. [Online]. Available: https://help.github.com/articles/what-is-my-disk-quota.

[7] »Github.com,« 20 Marts 2014. [Online]. Available: https://help.github.com/articles/working-with-large-files.

[8] H. MOLTKE, »Wired.cu.uk,« Wired, 23 Februar 2012. [Online]. Available: http://www.wired.co.uk/news/archive/2012-02/23/the-sky-is-rising.

[9] »Wikia.com,« [Online]. Available: http://vgsales.wikia.com/wiki/Video_game_industry.

[10] »Gartner.com,« 29 Oktober 2013. [Online]. Available: http://www.gartner.com/newsroom/id/2614915.

[11] »EMarketer.com,« 16 Januar 2014. [Online]. Available: http://www.emarketer.com/Article/Smartphone-Users-Worldwide-Will-Total-175-Billion-2014/1010536.

[12] »Wikipedia.org,« Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/Gamer.

[13] »store.steampowered.com,« Valve, [Online]. Available: http://store.steampowered.com/.

[14] »Wikipedia,« [Online]. Available: http://en.wikipedia.org/wiki/Casual_game.

[15] J. Lafferty, »Insiderfacebook.com,« 3 Marts 2014. [Online]. Available: http://www.insidefacebook.com/2014/03/03/top-25-facebook-apps-march-2014-inside-candy-crush-sagas-dominance/.

[16] J. Grubb, »Venturebeat.com,« 18 Februar 2014. [Online]. Available: http://venturebeat.com/2014/02/18/candy-crush-saga-publisher-king-by-the-numbers-inforgraphic/.

[17] J. Conditt, »Joystiq.com,« 9 April 2014. [Online]. Available: http://www.joystiq.com/2014/04/09/minecraft-pocket-edition-chips-off-21-million-copies/.

[18] »play.google.com,« Google, [Online]. Available: https://play.google.com/store.

[19] »Apple.com,« Apple, [Online]. Available: http://www.apple.com/osx/apps/app-store.html.

[20] »Theesa.com,« 2012. [Online]. Available: http://www.theesa.com/facts/pdfs/ESA_EF_2012.pdf.

[21] »Theesa.com,« 2013. [Online]. Available: http://www.theesa.com/facts/pdfs/esa_ef_2013.pdf.

[22] »Statisticbrain.com,« 3 Februar 2014. [Online]. Available: http://www.statisticbrain.com/mobile-phone-app-store-statistics/.

[23] B. Mitchell, »USAToday.com,« 12 Juni 2013. [Online]. Available: http://www.usatoday.com/story/tech/gaming/2013/06/12/women-50-percent-gaming-audience/2411529/.

[24] Calvin, »Apptopia.com,« 28 Maj 2013. [Online]. Available: http://blog.apptopia.com/game-demographics-that-every-developer-should-know/.

[25] J. Brownlee, »Cultofandroid.com,« 15 Maj 2013. [Online]. Available: http://www.cultofandroid.com/27547/android-piracy-outnumbers-ios-piracy-by-141-driving-devs-to-freemium-only-model/.

[26] E. Ravenscraft, »Androidpolice.com,« 31 Juli 2012. [Online]. Available: http://www.androidpolice.com/2012/07/31/editorial-just-how-bad-is-app-piracy-on-android-anyways-hint-were-asking-the-wrong-question/.

[27] S. Dredge, »Theguardian.com,« The Guardian, 15 August 2013. [Online]. Available: http://www.theguardian.com/technology/appsblog/2013/aug/15/android-v-ios-apps-apple-google.

[28] M. Zuckerberg, »Facebook.com,« Facebook, 25 Marts 2014. [Online]. Available: https://www.facebook.com/zuck/posts/10101319050523971.

[29] D. D'Orazio, »Theverge.com,« The Verge, 10 September 2013. [Online]. Available: http://www.theverge.com/2013/9/10/4715256/apple-700-million-ios-devices-sold-by-end-of-september.

[30] »Nin.dk,« Nordjysk Iværksætter Netværk, [Online]. Available: nin.dk.

[31] »vaekstguiden.dk,« Vækstguiden, [Online]. Available: http://vaekstguiden.dk/creative/0/3.

[32] Erhvervsstyrelsen, Januar 2014. [Online]. Available: http://erhvervsstyrelsen.dk/file/437960/vejledning-om-ivs.pdf.

[33] J. Sterling, »escapistmagazine.com,« The Escapist, 10 Februar 2014. [Online]. Available: http://www.escapistmagazine.com/videos/view/jimquisition/8773-Free-To-Wait.

[34] »seriousgames.net,« Serious Games, [Online]. Available: http://www.seriousgames.net/.

[35] C. Parkinson, »tutsplus.com,« Tutsplus, 10 Januar 2013. [Online]. Available: http://gamedevelopment.tutsplus.com/articles/how-my-brother-and-i-funded-our-indie-game-company--gamedev-2929.

[36] C. Smith, »Startapp.com,« Startapp, 1 Oktober 2013. [Online]. Available: http://blog.startapp.com/a-guide-to-ecpm/.

[37] A. Falcon, »Hongkiat.com,« Hongkiat, [Online]. Available: http://www.hongkiat.com/blog/mobile-app-monetizing-networks/.

[38] F. Bea, »Appflood.com,« Appflood, 2 Oktober 2013. [Online]. Available: http://appflood.com/blog/about-mobile-ad-fill-rates.

[39] »Quora.com,« Quora, 13 Maj 2011. [Online]. Available: http://www.quora.com/What-ranges-of-CPM-eCPM-can-a-successful-Android-game-command.

[40] H. Kniberg, Scrum and XP from the Trenches, C4Media Inc, 2007.

[41] K. S. a. J. Sutherland, »scrum.org,« 2013. [Online]. Available: https://www.scrum.org/Portals/0/Documents/Scrum%20Guides/2013/Scrum-Guide.pdf#zoom=100.

[42] D. Wells, »Extremeprogramming.org,« 8 Oktober 2013. [Online]. Available: http://www.extremeprogramming.org/.

[43] »hp.com,« HP, [Online]. Available: http://h10025.www1.hp.com/ewfrf/wc/document?cc=us&lc=en&docname=c02655320.

[44] »docs.unity3d.com,« Unity3D, [Online]. Available: https://docs.unity3d.com/Documentation/Manual/MobileOptimisation.html.

[45] »docs.unity3d.com,« Unity3D, [Online]. Available: https://docs.unity3d.com/Documentation/Manual/OptimizingGraphicsPerformance.html.

[46] M. A. T. F. J. Z. Anand Rangarajan, Energy Minimization Methods in Computer Vision and Pattern, Springer, 2003.

[47] M. Georgiou, »melgeorgiou.wordpress.com,« 25 December 2012. [Online]. Available: https://melgeorgiou.wordpress.com/category/unity3d-tutorials/unity3d-optimization-techniques/.

[48] »Wikipedia,« [Online]. Available: http://en.wikipedia.org/wiki/UV_mapping.

[49] »nuvera.com,« Nuvera, [Online]. Available: http://nuveraonline.com/kb/article.php?id=21.

[50] »Imgur.com,« [Online]. Available: http://i.imgur.com/aFKEttJ.png.

[51] »docs.unity3d.com,« Unity3D, [Online]. Available: https://docs.unity3d.com/Documentation/Components/class-Light.html.

[52] »cgchannel.com,« CG Channel, [Online]. Available: http://www.cgchannel.com/wp-content/uploads/2010/11/lighting.jpg.

[53] »docs.unity3d.com,« Unity3D, [Online]. Available: http://docs.unity3d.com/Manual/HOWTO-bumpmap.html.

[54] »docs.unity3d.com,« Unity3D, [Online]. Available: http://docs.unity3d.com/Manual/class-Light.html.

[55] »docs.unity3d.com,« Unity3D, [Online]. Available: http://docs.unity3d.com/Manual/Shadows.html.

[56] »docs.unity3d.com,« Unity3D, [Online]. Available: https://docs.unity3d.com/Documentation/Manual/ModelingOptimizedCharacters.html.

[57] »stats.unity3d.com,« Unity3D, Maj 2014. [Online]. Available: http://stats.unity3d.com/mobile/display.html.

[58] »docs.unity3d.com,« Unity3D, [Online]. Available: http://docs.unity3d.com/Manual/SceneView.html.

[59] »docs.unity3d.com,« Unity3D, [Online]. Available: http://docs.unity3d.com/Manual/GameView.html.

[60] »docs.unity3d.com,« Unity3D, [Online]. Available: http://docs.unity3d.com/Manual/ProjectView.html.

[61] »docs.unity3d.com,« Unity3D, [Online]. Available: http://docs.unity3d.com/Manual/Inspector.html.

[62] »docs.unity3d.com,« Unity3D, [Online]. Available: http://docs.unity3d.com/Manual/GameObjects.html.

[63] »docs.unity3d.com,« Unity3D, [Online]. Available: http://docs.unity3d.com/Manual/Prefabs.html.

[64] »docs.unity3d.com,« Unity3D, [Online]. Available: http://docs.unity3d.com/Manual/InstantiatingPrefabs.html.

[65] »Unity3D.com,« Unity3D, [Online]. Available: http://unity3d.com/unity/animation.

[66] J. Slick, »About.com,« About, [Online]. Available: http://3d.about.com/od/Creating-3D-The-CG-Pipeline/a/What-Is-Rigging.htm.

[67] R. H. Creighton, Unity 3.x Game Development by Example: Beginner's Guide, Packt Publishing, 2011.

[68] »docs.unity3d.com,« Unity3D, [Online]. Available: http://docs.unity3d.com/Manual/FBXImporter-Model.html.

[69] »docs.unity3d.com,« Unity3D, [Online]. Available: http://docs.unity3d.com/Manual/RootMotion.html.

[70] »docs.unity3d.com,« Unity3D, [Online]. Available: http://docs.unity3d.com/ScriptReference/Animation-animatePhysics.html.

[71] »docs.unity3d.com,« Unity3D, [Online]. Available: http://docs.unity3d.com/Manual/class-Animator.html.

[72] »docs.unity3d.com,« Unity3D, [Online]. Available: http://docs.unity3d.com/Manual/class-State.html.

[73] »docs.unity3d.com,« Unity3D, [Online]. Available: http://docs.unity3d.com/Manual/InverseKinematics.html.

[74] »forum.unity3d.com,« Unity3D, 15 Oktober 2012. [Online]. Available: http://forum.unity3d.com/threads/154816-Mecanim-Transition-Inspector-Checkboxes.

[75] »docs.unity3d.com,« Unity3D, [Online]. Available: http://docs.unity3d.com/430/Documentation/Components/class-Rigidbody.html.

[76] »docs.unity3d.com,« Unity3D, [Online]. Available: http://docs.unity3d.com/ScriptReference/GameObject.GetComponent.html.

[77] »Answers - Unity3D,« [Online]. Available: http://answers.unity3d.com/questions/7743/where-can-i-find-music-or-sound-effects-for-my-gam.html.

[78] »docs.unity.com,« Unity3D, [Online]. Available: https://docs.unity3d.com/Documentation/Manual/ModelingOptimizedCharacters.html.

[79] A. Orlova, »azoft.com,« Azoft, 20 Februar 2014. [Online]. Available: http://blog.azoft.com/mobile-development-app-trends/.