Indhold

[Indledning 4](#_Toc389417658)

[Hvad er et spil 4](#_Toc389417659)

[Problemformulering 5](#_Toc389417660)

[Hovedspørgsmål 5](#_Toc389417661)

[Delspørgsmål 5](#_Toc389417662)

[Forundersøgelse 5](#_Toc389417663)

[Teknologianalyse 5](#_Toc389417664)

[Indledning 5](#_Toc389417665)

[Valg af spilmotor 6](#_Toc389417666)

[CryEngine 6](#_Toc389417667)

[Unity3D 7](#_Toc389417668)

[GameMaker 8](#_Toc389417669)

[Unreal 8](#_Toc389417670)

[Byg eget framework 9](#_Toc389417671)

[Opsummering 9](#_Toc389417672)

[Valg af udviklingsmiljø 10](#_Toc389417673)

[Valg af billedredigeringsprogram 10](#_Toc389417674)

[Lydeffekter og musik 11](#_Toc389417675)

[Versionsstyring 11](#_Toc389417676)

[Unity Team Server 11](#_Toc389417677)

[GitHub 11](#_Toc389417678)

[Bitbucket 11](#_Toc389417679)

[Dropbox 12](#_Toc389417680)

[Opsummering 12](#_Toc389417681)

[Markedsanalyse 12](#_Toc389417682)

[Indledning 12](#_Toc389417683)

[Analyse af spilmarkedet kontra Anden Underholdning 13](#_Toc389417684)

[Analyse af spilplatform 15](#_Toc389417685)

[Kvalitetsforventninger 15](#_Toc389417686)

[Opsummering 16](#_Toc389417687)

[Segmentanalyse 17](#_Toc389417688)

[Salgsplatforme – AppStore mv. 17](#_Toc389417689)

[Målgruppeanalyse 18](#_Toc389417690)

[Demografi – Apple og Android 19](#_Toc389417691)

[Opsummering 22](#_Toc389417692)

[Virksomhedsopstart 22](#_Toc389417693)

[Virksomhedens interne ressourcer 22](#_Toc389417694)

[Håndgribelige ressourcer: 22](#_Toc389417695)

[Uhåndgribelige ressourcer 23](#_Toc389417696)

[Virksomhedens evner 23](#_Toc389417697)

[Kernekompetence 23](#_Toc389417698)

[Kerneydelser 23](#_Toc389417699)

[Virksomhedens eksterne miljø 23](#_Toc389417700)

[Opstartsbudget 25](#_Toc389417701)

[Finansiering 26](#_Toc389417702)

[Virksomhedsform 26](#_Toc389417703)

[Indtægtsmetoder 27](#_Toc389417704)

[Direkte Salg 27](#_Toc389417705)

[Reklamer 27](#_Toc389417706)

[Free-To-Play / Mikrotransaktioner 27](#_Toc389417707)

[Abonnementsordning 27](#_Toc389417708)

[Kontrakt 28](#_Toc389417709)

[Merchandise 28](#_Toc389417710)

[Opsummering 28](#_Toc389417711)

[SWOT 29](#_Toc389417712)

[Strategiplan 30](#_Toc389417713)

[Målsætning & Handlingsplan 30](#_Toc389417714)

[Projektvalg 31](#_Toc389417715)

[Projektbudget 32](#_Toc389417716)

[Projektafgrænsing <> Udviklingsfokus 33](#_Toc389417717)

[Projekt fokus 33](#_Toc389417718)

[Implementering 34](#_Toc389417719)

[Design 34](#_Toc389417720)

[Idegenerering??? 34](#_Toc389417721)

[Udviklingsproces 34](#_Toc389417722)

[Scrum 34](#_Toc389417723)

[Extreme Programming 35](#_Toc389417724)

[Introduktion til Komponentbaseret udvikling 36](#_Toc389417725)

[Beskriv i korte træk forskellene imellem objektorienteret udvikling, og komponentbaseret udvikling. 36](#_Toc389417726)

[Hvorfor benytter vi os af komponentbaseret udvikling frem for OOP? 36](#_Toc389417727)

[Grafiske designhensyn 36](#_Toc389417728)

[Begrænsninger og optimering 36](#_Toc389417729)

[GPU - Graphics Processing Unit 38](#_Toc389417730)

[Optimering af lys 40](#_Toc389417731)

[3D-modeller 41](#_Toc389417732)

[Skærmstørrelser 41](#_Toc389417733)

[Analyse af skærmstørrelser 41](#_Toc389417734)

[Design af brugergrænseflader 43](#_Toc389417735)

[Udvikling 43](#_Toc389417736)

[Indledning 43](#_Toc389417737)

[Introduktion til brugergrænsefladen i Unity 43](#_Toc389417738)

[Scene 43](#_Toc389417739)

[Game 44](#_Toc389417740)

[Project 44](#_Toc389417741)

[Hierarchy 44](#_Toc389417742)

[Inspector 44](#_Toc389417743)

[Animator 44](#_Toc389417744)

[Spiludvikling med Unity 44](#_Toc389417745)

[GameObject 45](#_Toc389417746)

[Component 45](#_Toc389417747)

[Prefab 45](#_Toc389417748)

[GUI Udvikling med Unity 45](#_Toc389417749)

[Mojo implementation 47](#_Toc389417750)

[Import af humanoid Mecanim model 47](#_Toc389417751)

[Oprettelse af Mojo som spilbar figur 49](#_Toc389417752)

[Opsætning af animator controller-komponenten 50](#_Toc389417753)

[Tilføjelse af yderligere komponenter 53](#_Toc389417754)

[Rigidbody komponenten 53](#_Toc389417755)

[Capsule Collider, fysik-komponenten 54](#_Toc389417756)

[Capsule Collider, Trigger-komponenten 55](#_Toc389417757)

[PlayerScript-komponenten 55](#_Toc389417758)

[Gennemgang af PlayerScript.cs 56](#_Toc389417759)

[Opsummering 59](#_Toc389417760)

[AI 60](#_Toc389417761)

[Procedural udvikling 60](#_Toc389417762)

[Proces 60](#_Toc389417763)

[Sprint 1 60](#_Toc389417764)

[Sprintmøde 1 60](#_Toc389417765)

[Retrospective 61](#_Toc389417766)

[Retrospective – Sprint 1 61](#_Toc389417767)

[Sprint 2 61](#_Toc389417768)

[Sprintmøde 2 61](#_Toc389417769)

[Retrospective – Sprint 2 62](#_Toc389417770)

[Sprint 3 62](#_Toc389417771)

[Sprintmøde 3 62](#_Toc389417772)

[Retrospective – Sprint 3 62](#_Toc389417773)

[Perspektivering 63](#_Toc389417774)

[Konklusion 63](#_Toc389417775)

[Bilag 63](#_Toc389417776)

[Ordforklaring 63](#_Toc389417777)

[Slamtekst taget ud af kontekst 64](#_Toc389417778)

# Indledning

Formålet med denne rapport er at undersøge, hvad der indgår i processen at udvikle et software produkt og udgive det. Rapporten forsøger at tage udgangspunkt i et virksomhedsperspektiv, hvor udviklingen og produktet skal danne grundlaget for en bæredygtig virksomhed.

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

Det næste er en markedsanalyse, hvis formål er at undersøge, hvordan man succesfuldt fører 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 forsøger afsnittet af redegøre for de styrker og svagheder ved virksomheden for at give et indblik i dennes bæredygtighed.

Den næste del af rapporten forsøger at redegøre for den proces, produktet er gennemgået. I denne del bliver der gjort redefor designfasen, hvor produktidéen mv. blev til. Endvidere gøres der rede for, 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 et 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 så 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, derfor skal der laves en markedsanalyse, så vi med en vis rimelighed kan konstatere at der rent faktisk findes en køber til vores produkt, og at det spil som vi producerer kan ramme et segment i den samlede køberskare.  
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 det kunne være relevant at gøre brug af.

For at kunne arbejde effektivt og målrettet skal der udvælges 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 der som IT-forundersøgelsen har afsløret i forbindelse med opstarten af Zen Rabbit Studios.

### Hovedspørgsmål

* Hvordan opstartes et bæredygtigt spilfirma.

### Delspørgsmål

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

# Forundersøgelse

# Teknologianalyse

## Indledning

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 vi 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, 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 |  |
| 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 |

Figur 1 - CryEngine features.

CryEngine udvikles af tyske CryTek, og har i udgangspunktet 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 ikke til at sige hvad en pro-licens helt præcist koster, da priserne varierer efter størrelsen på det firma der henvender sig. Der er dog tegn der tyder i retningen af, at CryTek skal have 20% af bruttofortjenesten ved licenser givet til indie-udviklere.  
Dog har CryTek for nyligt annonceret, 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 et 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 8000kr. (abb. 400kr/md)  Pro-iOS: +8000kr. (abb. +400kr/md)  Pro-Android: +8000kr. (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 |

Figur 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 pris på nul kroner og nul øre 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 der ikke så meget at sige til, det koster 8100 for en licens til den cyklus Unity er i, og den ejer man så til evig tid. Hvis man vil opgradere til den efterfølgende cyklus koster det det halve. Nu, og indtil Unity 5 lanceres kan man pre-order version 5, og få adgang til Unity 4 pro indtil da.  
Hvis man vil udvikle til smartdevices koster det yderligere 8100 for iOS pro og igen 8100 for Android Pro.   
Man får altså ikke adgang til f.eks. iOS pro features selvom man har Unity pro.   
Det er der mange meninger om, men det betyder at det hurtigt kan eskalere i prisen. Især fordi det der er tale om her gælder 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 4300kr. |
| 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. |

Figur 3 - GameMaker features

GameMaker 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 at 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 4300 kr, for at kunne eksportere til iOS og Android.

I lighed med Unity 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 |

Figur 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, har det kostet 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 alt for meget til en platform.  
Netop den type overvejelser er en vægtig grund for at udvikle sit eget system, om det så er et framework, eller en decideret spilmotor. 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 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 Microsoft´s 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 afgjort 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 vores kode. 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 vant til 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 vi 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 godt nok til vores umiddelbare behov, men mere professionelle produkter som f.eks. Maya og 3D Studio Max har flere og bedre funktioner end Blender, og derfor kan det på den 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 at 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 dem. I stedet agter vi at benytte os af 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 der kan integreres direkte i Unity. Det koster 2700 kr. pr person, og der er ikke nogen Sky-funktion inkluderet, hvilket vil sige 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 at man bruger et andet redskab til ”mellemregninger” af tungere typer filer, som f.eks. 3D-modeller.  
Strengt taget behøvede vi slet ikke betale for en GitHub konto, men gratis-kontierne er offentligt tilgængelige. Det er ikke noget problem i første omgang, men det er usmart 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 max 5 personer tilknyttet der skal bruge 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 os af en ressource 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 det i forhold til opstarten af virksomheden er det at skyde langt over målet. 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, 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, gratis, 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 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 blive taget op igen.

# Markedsanalyse

### Indledning

I dette afsnit vil vi analysere og danne os et overblik over underholdningsindustrien som hele, hvilket 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.

*“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 5 - Omsætning for den samlede underholdningsindustri, fordelt på områder.

Som det ses i figur 5 er spilindustrien en relativ lille del af underholdningsindustrien, men ser man på spilindustriens alder i forhold til de andres har den haft enorm fremgang på utrolig 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 6 - Udviklingen i omsætningen i spilindustrien.

[[1]](#footnote-1)

Som det ses af grafen i figur 6 har spilindustrien haft stor positiv udvikling, specielt de seneste år, og vi forventer ikke den aftager i fremtiden. På dette grundlag er det derfor naturligt at konkludere, at computerspilsbranchen klart bør overvejes at føre virksomhed i.

### Analyse af spilplatform

Specielt i spilindustrien er der en markant større forskel på de forskellige platforme sammenlignet med f.eks. TV-industrien, vor det ofte er irrelevant om produktet ses på mobiltelefonen, på computeren eller på tv’et. 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 grafen herunder:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 |

Figur 7 - Spilindtægt på verdensplan fordelt på platforme

Det, der kan læses ud fra Figur 7, er, at alle områder har stabil fremgang med undtagelse af håndholdte konsoller (Nintendo DS, PS VITA mv.), som forudsiges at miste markedsandel. 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. 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. Figur 7 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”.

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, og som derfor helt naturligt opbygger nogle højere krav til kvaliteten af det spil som han eller hun bruger sin tid på. Spil der henvender sig til denne type spiller kan være af en næsten uanet kompleksitet og scope.

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. Spil der produceres til casual-spillere er derfor typisk genkendelige 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 for hardcore- og casual-arketyperne, gælder det at der er mange gyldne middelveje; altså brugere som er lidt af begge dele. 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, og til de mobile enheder er det downloadet over 500 mio. gange. 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.

Det er absolut nødvendigt, at vores spil opfylder de forventningskrav som slutbrugeren har, og fordi casual-spillere stiller lavere krav til spillets kompleksitet, og 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 gøre det, må vi se på eksempler på spil som ligner det vi laver og som minimum sikre at vores produkt er af samme kvalitet.

### 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 virksomhedens fokus.

### Segmentanalyse

Når der er lagt fokus på en platform at udgive til, er der brug 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 genre af spil, som dominerer.

Figur 8 - Procentopdeling af spils popularitet fordelt på genre

Som ses ud af figur 8, 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%.

Hvis man udelukkende baserede sit valg af spil på tallene i figur 8, så burde man altså udvikle et Trivia-spil med sociale elementer for at ramme det bredest mulige segment.

### Salgsplatforme – AppStore mv.

Da markedet for mobile spil er i stærk fremgang er det relevant at undersøge, hvilke muligheder, der er, for at udgive sit produkt og ikke mindst optimering af salg. 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 genre 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. |

Figur 9 - Antal hentede apps, fordelt på salgsplatforme

Som ses i figur 9, er iPhone- og Android-markedet stærkt dominerende i antal downloadede apps og det vil derfor være naturligt at fokusere på disse som det primære mål for udgivelsen af produktet, da der er 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.

Af de fire markeder som er repræsenteret herover, er det umiddelbart Apple og Googles markedspladser som er mest valide. Vores primære valg af salgskanal bør være en af de to, men den endelige beslutning kommer også til at bero på andre faktorer, eksempelvis test hardware mv.

### Målgruppeanalyse

Når produktet er lagt fast på genre og spillestil er det vigtigt at redegøre for den målgruppe, man håber at nå, for at være i stand til at optimere markedsføringen.

For at få et overblik over det generelle billede af spilindustrien, kigges der på et nogle grundlæggende demografiske tal:

Den midterste figur viser aldersfordelingen af computerspillere verden over, og som det ses af figuren er spillerne relativt unge, da 80% er 35 år eller derunder. Det næste, som bør bemærkes, er det relativt lille segment af unge under 18. Da computerspil let kan fortolkes som en form for legetøj, er det derfor naturligt at forestille sig et ungt publikum. Men da gennemsnitsalderen for computerspillere er 26 år, bør man indstille sig på at rette sin markedsføring mod det modne publikum.

Dette leder til den sidste figur, som viser computerspillernes daglige beskæftigelse. Som det ses er 50% i et fuldtidsarbejde, sammenlagt 30% er enten fuld- eller deltidsstuderende, og kun 11% er arbejdsløse. Mens kun 2% er pensionerede, hvilket i øvrigt hænger fornuftigt sammen med aldersfordelingen.



Figur 10 - Køns- og aldersfordeling imellem spillere.

Figur 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, overordnet set, 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.

### Demografi – Apple og Android

Nu hvor det er besluttet at fokusere vor opmærksomhed på Apple og Google, er det interessant at se hvilken forskel der er i demografien blandt deres kunder.



Figur 11 - Demografiopdeling mellem Android og Iphone brugere.

Det første, og måske vigtigste tal man skal bide mærke i i figur 10, er at der er flere Android, end iPhone brugere. Det lå også i kortene i de tal som fremgår af figur 9, 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. Som det fremgår af figur XXX, så er der kunder uanset hvilket platform vi udgiver på.

Tabellen viser os dernæst, at der marginalt flere kvinder end mænd som har en iPhone. Fordelt på alder, er der især mange yngre 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 12 - Populære genrer fordelt på køn og alder

Figur 11 giver et indtryk af hvilken type spil man kan forvente at en given aldersgruppe, og køn, spiller. Genrerne er plottet ind 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 10, så er kønssammensætningen, i hvert fald, i Amerika, således at der 52% kvinder der ejer en smartphone, og 55% mænd. Baseret

Kigger man på figur 11, kan man læse at arkadespil og ”Runner”-spil lægger i det neutrale felt, hvor der ikke er specifikke kønsdominans. Dog hælder genrene mere til den kvindelige side frem for den mandlige. Samtidig lægger begge genre i den yngre del af denne figur, men stadig omkring gennemsnitsalderen for computerspillere. Dog er der et spænd imellem de to, hvor gennemsnitsalderen for arkadespillere er 10 år ældre end ”runner”-genren. Det er derfor vigtigt, at have en klar idé om, hvilken genre produktet tilhører når det skal designes.

### Opsummering

Når man efterhånden har fået 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 freemiumelementer.

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

Baseret på de oplysninger vi har fået om vores potentielle målgrupper, så konkluderer vi, at vi kan lave spil til mænd, kvinder, unge, voksne, og i nærmest alle typer genrer. 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

### 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 ifb 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, 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. 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 mere modne platforme såsom: 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 samme teknologiske fremskridt der gør det muligt for os at starte op, har gjort det samme for tusindvis af andre udviklere, og de har alle sammen mere erfaring med udvikling end os.   
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 lave et build 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** |

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 projekt.

Da vi agter at gå til Nordjysk Iværksætter Netværk 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 driftbudget.

På denne måde ender vi med et relativt lavt beløb som vi selv kan finansiere.

## Finansiering

Til virksomhedens opstart 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. Når vi kommer dertil er vi, jf. opstartsbudgettet, nødt til at finde nogle flere penge til opstarten.  
Vi kan her bruge vores egne penge, men af helt 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 organisk udvikling.

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.   
Derudover er der en meget lang række forskellige tilskud man kan søge, hvilket er et område som vi vil undersøgte ganske nøje.

## Virksomhedsform

**Zen Rabbit Studios** skal startes op med så lille en startkapital som muligt, hvilket udelukker virksomhedsformerne Aktieselskab (A/S) og Anpartsselskab(Aps) der ville kræve indskudsværdi for hhv. 500.000 og 50.000. Tilbage er der 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, 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 som det koster at konvertere til et Aps. Først herefter kan man hæve et udbytte i virksomheden.

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 spørgsmålet om virksomhedstypen er irrelevant.  
Altså er det hensigten at starte virksomheden som et IVS.

## Indtægtsmetoder

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. En genre hvor dette har været udbredt er i sværvægtsklassen af MMO-spil. World og Warcraft har f.eks. kørt med betaling for spillet selv, og så en månedlig abonnementspris derefter.

### 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 er 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.   
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 made 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. De mest kendte tæller bl.a. Electronic Arts, Activision og 2K.  
Mindre samarbejdspartnere end dem, kunne dog også gøre det. Der findes eksempler på små studier som laver små spil for firmaer til deres markedsføring.

### 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.  
Der 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.  
I vores tilfælde vil det sige enten direkte salg, gratis med reklamer eller gratis med mikrotransaktioner.  
  
På lidt længere sigt kunne det være rigtigt interessant at kigge på Free-To-Play modeller. For en udvikler er der nærmest kun positive sider ved den metode:  
1. Man kan give produktet til kunden gratis, hvilket formindsker kundens barriere for køb.  
2. Spillet kan blive ved med at tjene penge lang tid efter kunden har fået det mellem hænderne.  
3. Det er en strategi der kan benyttes til alle typer spil, og på de fleste platforme.  
   
  
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.

Udgangspunktet for virksomheden er, at der er to programmører og en grafiker til rådighed for virksomheden 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 giver os en solid platform hvorpå vi kan diskutere forskellige designløsninger, indtjeningsstrategier osv.  
Der er ingen reel organisationsstruktur i virksomheden, da udgangspunktet er at grundlæggerne er lige stillet, men med er der en grovopdeling af opgaver 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

Hvis man tager de ovenstående faktorer, og vurderinger i betragtning kan samle det i en SWOT-model som set i figur 13. 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 figur <opstartsbudget>.

Vi starter firmaet som en sidebeskæftigelse, derfor har vi ingen trusler i forhold til at gå fallit som følge af lønudgifter mv. Den eneste reelle trussel fra markedet, er at vores produkter 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 |

Figur 13 - SWOT model for Zen Rabbit Studios

### Strategiplan

Hvor er 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 porteføjle, 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.

Hvor er 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.

Hvad det for en position som virksomheden ønsker at indtage:

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.

### Målsætning & Handlingsplan

* Om 3 mdr. Skal der være en prototype klar.
* Om 4 mdr. Skal virksomheden have firmahjemmeside og pressemateriale klar.
* Om 6 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.

## Projektvalg

Det spil vi har udvalgt som vores første projekt er en hybrid imellem en Endless Runner og et almindeligt action 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 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 levelstruktur. Altså, vores baner er 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 duelsystem, hvor vores protagonist i bedste westernstil står ansigt til ansigt med de primære antagonister.

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 level op af generiske byggeklodser behøver vi ikke bruge store mængder tid på at lave leveldesign. 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 playtest.

Playtest 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 at 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 det en senere Android version vil være gratis for brugeren at hente, men de må til gengæld leve med en hvis grad af reklamer.

Angående IAP vil vi implementere en butik i spillet hvorfra vi for mindre beløb kan sælge nye kampagner, og alternative skins til protagonisten. Dette gælder både for iPhone, og Android udgaverne.

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.

### Projektbudget

Reklameindtægter er baseret på antal visninger, og beregnes ofte via eCPM.

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 stensikre 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 an efter hvilket netværk man er tilkoblet. Derfor kan man med fordel benytte flere forskellige reklamenetværk.

Et andet godt argument for at være tilknyttet flere reklamenetværk er, at man på den måde øger produktets fillrate.

Fillrate, 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. I figur 14 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 |
| 6500000 | 27 | 3 | 526500 | iOS |
| 6500000 | 27 | 3 | 526500 | Google Play |
|  |  |  | 1053000 | I alt |

Figur 14 - Det forventede antal hentninger der er påkrævet for at nå en million i omsætning ved brug af reklamer.

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 |
| 2000000 | 0,025 | 10 | 500000 | iOS |
| 2000000 | 0,025 | 10 | 500000 | Platform |
|  |  |  | 1000000 | I alt |

Figur 15 - Det foventede antal hentninger der er påkrævet for at nå en million i omsætning ved brug af IAP.

Da vi benytter os af en kombination af de to scenarier bliver det rent faktiske antal nødvendige brugere naturligvis lavere. Men da vores samlede potentielle brugerflade er et sted imellem 1,5-2 mia. mennesker, så er det ikke et urealistisk scenarie.

# Projektafgrænsing <> Udviklingsfokus

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 forhold til vores teknologianalyse vil vi kun beskæftige os med et smalt udsnit af de produkter der reelt er til rådighed, baggrunden for udvælgelsen af hvert enkelt produkt er beskrevet i det relevante kapitel.

I forhold til udviklingen af grafiske artefakter, og de teknologier der er forbundet med den udvikling, så er det et emne som vi kun ganske kort berører. Al endeligt materiale af den art bliver udviklet af den tredje virksomhedsgrundlægger, der er grafisk uddannet, og som i øvrigt ikke er involveret i udarbejdelsen af denne rapport.

I forbindelse med test findes der en lang række af bugtracker-værktøj som vi kunne vælge at gøre brug af. I forbindelse med dette projekt vil vi ignorere disse værktøjer, da vi ikke har et reelt behov for at gøre brug af nogen af dem.

### Projekt fokus

Det er af, af flere grunde, ikke muligt færdiggøre vore spil inden for projektets deadline. Det skyldes primært, at produktionen kræver en lang række forskellige aktiver 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 lyd, og at vi kun laver en enkelt spilbar ”verden” færdig.   
  
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 vi vil benytte os 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.

<alle afgrænsningerne ifh til forundersøgelsen skal stå i de relevante kapitler. Dette kapitel skal samle op, og redegøre for fokuspunkterne i produktet.

f.eks. Random spawns, noget om AI, Character Controller, sceneopsætning.

# Implementering

# Design

## Idegenerering???

## 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 og en høj grad af erfaring. 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 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.

### 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.

Den væsentligste ændring vi har foretaget, er at vi accepterer at en story kan oprettes og tages ind midt under et sprint. Det er er ellers en af de få regler som anses for fastlagte i Scrum, at det skal 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. 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 upræcis. 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 level 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 GDD, og TDD. Begge dele er dokumenter 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.

### Extreme Programming

I forhold til udviklingen af scripts, benytter vi os af XP næsten som det er beskrevet i manifestet. 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

### Beskriv i korte træk forskellene imellem objektorienteret udvikling, og komponentbaseret udvikling.

For det første er det vigtigt at slå fast, at man ved begge udviklingsparadigmer arbejder med objekter. Forskellen mellem de to ligger i den måde man organiserer objekterne på. I klassisk 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 at Mojo ***er et*** GameObjekt, og Mojo ***er en*** bevægelig figur etc. Fordi det giver logisk mening.

Hvis 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.

### Hvorfor benytter vi os af komponentbaseret udvikling frem for OOP?

Fordi Unity er bygget op omkring komponenter.

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 koncentrerer vi os fremadrettet kun om komponenter.

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 renderer, fordi de allerede er lavet for os. Vores opgave er at levere de scripts, som i sig selv er komponenter, som skal forbinde GameObjekter og komponenter med hinanden.   
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.

# Grafiske designhensyn

Når der udvikles til mobile platforme er der adskillige elementer, som skal tages højde for. Samtidig er det heller ikke altid muligt at udnytte den viden og de ressourcer, som er gældende til udvikling af eksempelvis computerspil. Udvikler man derfor til smartphones og tablets er det nødvendigt at undersøge og redegøre for de faldgruber, som et projekt kan bremses af.

## Begrænsninger og optimering

Den største forskel man ser, når man sammenligner fx computerspil eller konsolspil med mobile spil, er den computerkraft, som de forskellige har til rådighed. Hvorimod computere og konsoller har rig mulighed for at blive udrustet med kraftfulde komponenter, så er fokuspunktet i smartphones og tablets mobilitet herunder kompakthed. Dette stiller høje krav til hardwaren, da det fysisk begrænser pladsen til ydeevnen. Samtidig er temperaturen et element, som skal indberegnes, hvilket kræver nedkøling af hardwaren enten i form af blæsere eller vandkøling. Dette betyder, 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. Det er sammen med processoren oftest flaskehalsen i enhedens ydeevne. 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. En kompleks shader resulterer derfor i en høj fillrate i applikationen, og det anbefales at bruge en så simpelt som muligt shader.

Dog findes der andre områder, som kan optimeres på. Et er tekstur-detaljegraden, som kræver hukommelse af den mobile enhed. Kræver dette 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 e 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 teksturer 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. Her ses et eksempel på ingen brug af mipmaps imod brugen af mipmaps.



Figur 16 - tv. ingen mipmaps. th. med mipmaps.

Som det ses på billedet 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.

En anden måde, at optimere brugen af teksture på, er ved at benytte sig af det, som kaldet et tekstur atlas. Et tekstur atlas er en enorm billedfil, som indeholder en lang række, hvis ikke alle, teksture i en scene. Dette gør renderingen mere effektiv, da filen bliver håndteret som et enkelt enhed i udregningerne, hvilket især er effektivt, hvis spillet indeholder mange små objekter, som bruges ofte. Et typisk atlas ser sådan ud:



Figur - Eksempel på et texture atlas

Dette kræver dog lidt tilpasning, da UV-mappet skal hænge sammen med tilhørende model. Et UV-map er det, som definerer hvordan en tekstur skal pålægges modellen. Lægges alle teksturene derfor i en ny fil kan det være nødvendigt at sikre sig korrekt pålægning af disse.

Det er samtidig muligt for teksturene at overlappe ved brug af mipmaps, og det er derfor igen nødvendigt at sikre sig at underbillederne ikke bliver forurenet af de omkringliggende billeder.

Sidst er det værd at nævne LOD, som står for Level Of Detail. Med LOD forsøger man at mindske antallet af draw calls ved at simplificere objekterne i scenen ud fra afstanden til kameraet. Teknikken minder meget mipmaps, men dækker over modellerne i scenen. Et eksempel kan ses her:



Figur - Demonstration af LOD

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

Alt i alt er formålet at skabe et produkt, som kan håndteres af flest mulige enheder for at øge det potentielle marked.

### Optimering af lys

Et andet element, som i høj grad påvirker ydeevnen er de forskellige former for lys, som kan indbygges i et produkt. 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 vertexen. Pixel lys bliver herimod udregnet for hver enkelt pixel på skærmen. Forskellen ses herunder, vertex lys til venstre og pixel lys til højre:



Figur 19 - tv. Vertex lys. th. pixel lys

Som det ses giver pixellys langt højere kvalitets lys, men stiller også langt større krav til den mobile enhedsydeevne. Dog giver pixellys nogle muligheder, som ikke er tilgængelige med vertex lys. Med pixel lys er det 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. Light cookies er en funktion, som giver lyset et slags filter, hvor formålet er at give udvikleren mulighed for manuelt at bestemme lysets mønster.   
Realtime skygger er, som navnet indikerer, realtidsskygger, som opdaterer 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, hvorefter alle objekterne, som kan modtage skygger, renderes ind i skyggekortet. Denne proces er derfor mere krævende af grafikkortet end pixel lys i sig selv.

### 3D-modeller

Tredimensionelle modeller, som oftest er brugt i videospil, er krævende komponent i et produkt. Det er dermed også et element, hvor ydeevne burde være et fokuspunkt under udviklingen af disse. Når en 3D-model udvikles til den mobile platform, er der en række punkter, som er værd at undersøge.

Det første, som bør redegøres for, er antallet af polygoner, modellen skal bestå af. Antallet af polygoner bestemmer detaljegraden af figuren og dermed kvaliteten af det visuelle element, men jo højere antal polygoner, jo højere ydeevne er nødvendig. På en mobil platform 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. 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 stiller højere krav til ydeevnen.

Det næste mål for optimeringen er at bruge så få materialer som muligt. Et materiale i 3D-modelleringssammenhæng er en samling af teksture samt shaders mv. på modellen. Dog er det ikke typisk nødvendigt med mere end ét materiale, men der kan opstå situationer, hvor kvaliteten kan forbedres.

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 mobilapplikation. Dette kan sammenlignes med et computerspil typisk med imellem 15 og 50.

Det anbefales kun at bruge en enkelt ”skinned mesh rendere” 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.

## Skærmstørrelser

Når man skal udvikle en brugergrænseflade er der flere elementer, som skal tages højde for bl.a. skærmstørrelse og dimensionerne, altså forholdet mellem længden og bredden. Skærmstørrelsen og dimensionerne på kan variere betydeligt alt efter, hvilken platform, der udvikles til, herunder mobile enheder som smartphones og tablets, men også computerskærme, tv mv. I dette projekt, hvor produktet er henvendt til smartphones og tablets, er der også et væld af forskellige skærmstørrelser, som skal indregnes i 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 forventet. I bedste tilfælde er brugergrænsefladen ikke æstetisk nydelig, men dog anvendelig.

### Analyse af skærmstørrelser

Før udviklingen påbegyndes er det nødvendigt at vide præcist, hvilke skærmstørrelser er mest populære og derved gør sig gældende. Samtidig gør dette det også muligt at spare ressourcer, hvis der ikke behøves at udvikles og testes til småt udbredte skærmstørrelser.

Figur 20 - Udbredelse af skærmstørrelser i procent

Som kan ses ud fra denne graf, ligger størstedelen af de mobile enheder inden for relativt få forskellige skærmstørrelser, da ’Andre’ kun dækker over 16,8% af alle enheder.

Det, som dog er mere relevant at redegøre for, er dimensionerne på skærmen. Da det samme størrelsesforhold betyder, at det er muligt kun at nøjes med at skalere brugergrænsefladen, volder dette kun få problemer. Derimod er det ikke muligt kun at skulle skalere grænsefladen, hvis dimensionerne er anderledes.

Figur 21 - Udbredelse af skærmdimensioner i procent

Kigger man derfor på de typiske dimensioner på markedet, 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 muligt at lægge fokus her.

### Design af brugergrænseflader

Når der designes grænseflader, er der en god mængde teori og designregler, som støtter udviklingen af disse. I dette projekt forsøges der at tage nogle af disse designregler i brug. Menuen er en grænseflade og dermed gælder der en række designregler, som bør overvejes under opbygningen. De første designprincipper, som bør indlejres i menuen er gestaltlovene. 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.

# Udvikling

## Indledning

Tilgår

## 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, i hvert fald de følgende.

### Scene

Scene-vinduet er hvor man opstiller sin scene. 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 lavede et build.

### 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.

### 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 GameObjeck 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.

### 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

Hvad er de grundlæggende koncepter som vi arbejder med i unity?

Spiludvikling med Unity handler grundlæggende om at manipulere GameObjekter 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 scene (eller Hierarchy-vinduet), bliver der oprettet GameObjekt som kan indeholde det asset. 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 komponent er en ”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. 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. 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 Run-Time 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.

## 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 menusystemet. Systemet, som består af en række GUI-elementer, er bygget i Unity3D’s eget miljø vha. scripting. Miljøet minder meget HTML og ASP, men mangler noget funktionalitet i form af relative koordinater på skærmbilleder, som er under opsejling en ny opdatering.

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 stort brug af script-funktionerne Screen.Height og Screen.Width, som henviser til skærmens længde og bredde. Uanset hvilken skærmstørrelse der er tale om, kan man derfor 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 22 - Udsnit af MainMenu.cs

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’, som indeholder følgende parametre, tekst, højde og vidde. Teksten henviser til den tekst, som vises på selve knappen på i brugergrænsefladen, hvor i dette tilfælde 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. 12 er her valgt udelukkende ud fra, hvad der så passende ud fra udviklerens side. 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 dette projekt er det funktionen, ’GUILayout.BeginArea’, som opretter et område, der kan redigeres. Det bliver eksempelvis brugt i scriptet, ’LevelSelect’, hvor det enkelte områder, som bliver oprettet indeholder en række knapper.

Figur 23 - Level Select menuen.

Som det ses på figur 23, 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 af område og knapper. Koden for et enkelt af disse områder ses her:

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 24 - LevelSelect.cs

Der startes med at oprette et område, som bliver tildelt dynamiske variabler, for at sikre det dynamiske layout. Derefter bliver der påbegyndt et 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’, hvor efter endnu en knap oprettes, som nu ligger under den første vandrette række. Sidst afsluttes den lodrette række, hvorefter området afsluttes.

Der er altså blevet oprettet en række scener med tilhørende GUI, udviklet i Unity3D’s eget system. Systemet tager højde for skærmens størrelse og dimensioner, og det må dermed siges at opgaven er afsluttet.

## Mojo implementation

### 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”. Konkret betyder det, at man allerede i modelleringsprogrammet tilføjer et antal ”knogler” til modellen, på figur 1 er det vist hvordan knoglerne tager sig ud i Blender animationsprogrammet. I Blender er der også funktionalitet til at knytte knoglerne sammen med modellens mesh, altså dens væv.

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



Figur 25 - modelrigging i blender

Første trin for at kunne oprette vores protagonist 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 26, de tre øverste objekter er mesh-objekter og udgør tilsammen figurens udseende. Default\_Take er et animationsobjekt som Blender automatisk genererer ved eksport.



Figur 26 - Unity deler en importeret model op i GameObjects

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 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. Det 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 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 Blender giver mindre manuelt arbejde nu, da det øger muligheden for at Unity selv kan placere hver knogle i den korrekte position. På figur 27 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 27 - Cirklerne med stiplede linjer markerre knogler som er valgfrie at tilføje.

Tilbage til Import-indstillingerne, 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.

Reelt er den eneste *scale den eneste* værdi som der er behov for at indstille på denne side. Som beskrevet i **introduktionen til unity,** så er forventer fysikmotoren i Unity 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. I dette tilfælde passer en 0.5 opskalering til resten af game-verdenen.

### Oprettelse af Mojo som spilbar figur

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 passer, er det en god ide at lave et standard cube-objekt, til at sammenligne modellen med. 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 ganske vidst 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. (se figur 28)



Figur 28 - 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 GameObjektet, hvis det skulle blive nødvendigt.

### Opsætning af animator controller-komponenten

Da Mojo er oprettet som en Mecanim kompatibel rig, oprettes der automatisk en Animator-komponent på gameobjektet når han trækkes ind i scenen. Animator-komponenten (figur 29) 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. 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. 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.

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, slide og skyde. Controller-komponenten skal opsættes så der er overgange imellem de relevante tilstande.



Figur 29 - 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 29)



Figur 30 - 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 vi kommer til senere. Den eneste vi benytter 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 State indeholder en animation, der skal afspilles under en given omstændighed. Der er altid en animation, som skal være standard. I figur 30 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. *Speed* indikerer hvor hurtigt animationen skal afspilles relativt til den hastighed hvormed den er optaget.

*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. 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.



Figur 31 - Indstillingsmuligheder for en tilstand

Hvis spilleren hopper, skal der være en overgang til at indikere den handling. Derfor er der imellem Run og Jump oprettet en transition. På figur 32 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. 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 32 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 30 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. Med det, er den ene side af vores character controller færdig.



Figur 32 - Øv. transition fra Run til Jump. ndst. Transition fra Jump til Run.

## Tilføjelse af yderligere komponenter

Introducer det faktum at der skal flere komponenter til for at få det til at spille

### 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 en character controller-komponent som kommer med Unity. 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. Da vi bestemmer både vægt og kraft (også tyngdekraft), er standardværdien 1 blevet fastholdt, og så indrettes påvirkningskræfterne efter det.

*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 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.

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 33 - Mojo omgivet af Capsule Collider

Attributten *Is Trigger* angiver om Collideren (figur 34) 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 ifh. til det GameObject som den er tilknyttet. På figur 33 er den indstillet til at dække Mojo, og kan ses som et grønt grid over selve figuren.



Figur 34 - 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 GameObject, 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 GameObjekt, 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.

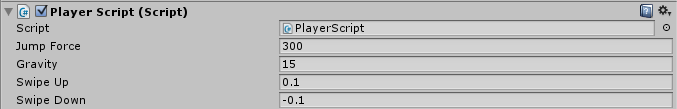
### PlayerScript-komponenten

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 ordentligt til udtryk.

Det første vi gør i Scriptet, er at erklære at UnityEngine skal anvendes, dernæst scriptets navn, og 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 35 - PlayerScript-komponenten. Mulighed for indstillinger uden adgang til koden

Som det fremgår af Figur. 35, kan man ændre i den kraft som objektet bliver påført når man hopper. Det er ligeledes muligt at ændre i tyngdekraften, samt 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() 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 der er ingen grund til at gøre det anderledes.

Figur 36 - PlayerScript Start()

void Start () {

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

GetComponent<T>() gennemsøger det objekt som scriptet er tilknyttet efter en komponent af den type som er angivet. 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 metoder: 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 gameobjekt som parameter. Da det ikke er gameobjektet vi skal bruge, benyttes GetComponent<T> direkte på det fundne gameobjekt.

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 denne metode. 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 37, sikrer at Mojo altid løber lige ud. De animationer der er tilknyttede er ikke hundrede procent i vatter, hvilket uden denne linie, der for hvert kald til metoden nulstiller Mojo´s position på Z-aksen. 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 mildest talt ikke et kønt syn. 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 37 - PlayerScript Update()

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. 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 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.

Figur 38 - SwipeControls() håndtere brugerinput

private void SwipeControls()

{

if (Input.touchCount > 0 && Input.GetTouch(0).phase == TouchPhase.Moved)

{

//Find bevægelsen fra sidste frame til nu

Vector2 deltaTouchPos = Input.GetTouch(0).deltaPosition;

//Hvis swipet er opadgående og spilleren har jordforbindelse

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());

}

}

}

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 39 sættes højden på fysik collideren til en fjerdedel af normal, ligesom dens centrum flyttes end 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 39 - 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.

Figur 40 - Coroutine eksempel

IEnumerator WaitForSlide()

{

//Debug.Log("waitforit");

yield return new WaitForSeconds(1.2f);

StandUp();

}

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 16 tilser dette.

Figur 41 - StandUp()

void StandUp()

{

\_psysCol.height = 1f;

\_psysCol.center = new Vector3(0, 0.5f, 0);

\_collisionCol.enabled = true;

}

## 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, og er samtidig den mindst krævende i form af GPU-kraft.

I forsøget på at holde den påkrævede GPU-kraft nede bruges der ikke realtime skygger, da disse er tunge for grafikkortet. Dette betyder ingen pixellys, som er beskrevet i afsnittet ’Optimering af lys’. Som det ses i figur 20, er lyset bygget så simpelt som muligt. Objektet gør ikke brug af *cookies*, *halo*, *flare* mv.

Figur - Directional Light komponent

*Render Mode*, som beskriver kvaliteten af renderingen af lys, står til ’Auto’, hvilket betyder den gør brug af Unitys indbyggede kvalitetsindstillinger. Det samme gælder for *Lightmapping*, som definerer hvorvidt der bruges realtime lys, baked lys eller der hentes fra kvalitetsindstillingerne. Da vi hverken anvender realtime lys eller baked lys, er det naturlige valg at definere det i kvalitetsindstillingerne.

*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.

## AI

Tilgår

## Procedural udvikling

Tilgår.

# Proces

## Sprint 1

### Sprintmøde 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. 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

I alt 23 story points ud af 25, som vores velocity tillod os.

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 højt i produkt backloggen, da de udgør den grundlæggende funktionalitet i produktet og samtidig gør den videre udvikling lettere.

I begyndelsen af Player-funktionaliteten var der brug for undersøgelse af emnet, da der var usikkerheder omkring implementeringen. Værktøjet som anvendes er et indbygget system i Unity, kaldet Mecanim, der stykker animationer sammen og som vi skulle bruge til spillerens bevægelser. Der var dog 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 implementeringen 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 at helt afslutte player-delen som kræver mere arbejde længere henne i processen. Vi har dog valgt at nedskrive disse stories ved deres respektive afslutninger, fordi produktionen af de animationer vi skal bruge til slutproduktet er stories for sig selv.

Udviklingen med banen gik lettere end forventet, og det var muligt at nedskrive hele Level-delen relativt hurtigt. Arbejdet bestod af nogle relative små spikes, omhandlende tilfældig generering af såvel banens platforme og baggrundende.

### Retrospective

Alt i alt er det første sprint vellykket, da det har været muligt at nedskrive alle stories i sprintbackloggen, selv med forventede 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 næste stories i rækken for at undgå eventuelle spikes og få skubbet funktionalitet ud. Vi kan med fordel revurdere de estimater vi gav vores stories i starten af sprintet, da mange af dem var langt hurtigere overstået, end forudset.

### Retrospective – Sprint 1

## Sprint 2

### Sprintmøde 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 optimere på vores character controller, som er den samling af komponenter der håndterer styringen af vores protagonist. Ugen startede ud med en kort opsætning af backloggen, samt uddelegering af arbejdsopgaver. Menusystemet, som skulle oprettes, skulle bestå af en brugergrænseflade med tilhørende funktionalitet, hvor det er muligt at klikke sig rundt imellem de forskellige menuer. I forbindelse med optimering af character controlleren betod af at modellere en 3D-figur i Blender for derefter at animere nogle grundlæggende besluttede vi, at lave en spike hvori vi ville lave en 3D model til brug for mecanim fra bunden, og undersøge hvordan man lavede tilhørende animationer, som kunne 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:

* Menu, hovedmenu – 5 point
* Menu, ’Vælg bane’ – 5 point
* Menu, ’tutorial’ – 2 point
* Menu, ’indstillinger’ – 2 point
* Menu, ’Credits’ – ½ point
* Menu, exit-knap – ½ point
* Spike, 3D-model – 3
* Spike, Animation – 4.5

Med 22,5 story points i alt var der lidt spillerum, hvis der skulle opstå noget uforudset. Opgaverne blev fordelt med menu-funktionaliteten til én udvikler og 3D-animeringen til en anden.

At skabe et menusystem viste sig at være en relativt simpel, men langsommelig proces. Det var ikke et problem at få en forståelse for Unity3D’s GUI-system, da det i høj grad mindede om ASP.NET samt HTML. Der blev hurtigt skabt en prototype, men da løsningen 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. Dette betød manuel opsætning af næsten samtlige knapper mv., som krævede et godt antal timers arbejde. Men det lykkedes i sidste ende at nå frem til et tilfredsstillende resultat, som gav mulighed for at trække nye opgaver ind i dette sprint. Her blev valgt at fokusere på følgende stories:

* Indsæt baggrundsmusik – ½ point
* Indsæt lydeffekter – 3 point

At indsætte lyd er en relativ let opgave, men da lyd ikke automatisk medfølger ved sceneskift, viste sig at være mere kompliceret end som så. Det lykkedes dog at få en umiddelbar løsning på plads, som nu har banet vej for at afslutte resten af opgaven, da den ikke er fuldt afsluttet endnu og dermed ikke nedskrevet.

### Retrospective – Sprint 2

Sprint 2 har på visse områder været en succes, i det at menu-opgaven kunne nedskrives helt og dermed muligt at inddrage nye stories. Samtidig gav det både mulighed for at fokusere på andre aspekter af projektet og optimere på eksisterende områder i produktet. Animationsopgaven har været en tung opgave, som har derfor opslugt megen tid uden at vi har fået noget konkret ud af det. Det er godt for teamet, at flere kender processen med at udvikle modeller og animationer. Men især i forbindelse med animationsarbejdet er der et ganske langt stykke vej fra at vide hvordan en animation optages, og så til at få et bare nogenlunde acceptabelt resultat ud af det. Model- og Animations-spikene blev løst for så vidt som at det handlede om at forstå udviklingen. Men vi har ikke benyttet os af de assets som blev produceret her.

## Sprint 3

### Sprintmøde 3

### Retrospective – Sprint 3

# Perspektivering

# Konklusion

# Bilag

### Ordforklaring

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, og altså ikke

indhold generelt.

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.

Fillrate, GPU – Tilgår.

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 – Folk der reparerer spil.

In App Purchases – Bruges sammen med mikrotransaktioner, og er en ”butik” i spillet hvor

brugeren kan købe virtuelle dele til spillet for rigtige penge.

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.

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 den færdige version 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.

# Slamtekst taget ud af kontekst

IT-forundersøgelsen klarlægger de styrker og svagheder som eksisterer ved opstarten af **Zen Rabbit Studio**. Dette opnåes ved dialog imellem iværksætterne, og strategisk analyse. Efter dette kapitel er der en klar forståelse af hvilken type virksomhed der er under opbygning, samt dens fremtidige muligheder. Ligeledes forelægger der en detaljeret handlingsplan for den første periode af virksomhedens eksistens, samt en mere løs vision for tiden derefter.  
Kapitlet er baseret på <KILDER> og ejernes vision.

Det er afgørende, at virksomheden hurtigst muligt får udgivet et spil, af en kvalitet som gør at det kan anvendes til at tiltrække investorer. Efter udgivelse på en platform skal det vurderes om der er basis for, at lave opdateringer til spillet, eller om det skal porteres til flere platforme. Hvorvidt vi skal det ene eller det andet, beror på om spillet er en succes eller fiasko. Såfremt der er værdi i videreudvikling på spillet, kan målet om to udgivne spil på et år fraviges.  
Uanset om spillet har været en kommerciel succes eller ej, er det vores udgangspunkt for at tage kontakt til potentielle investorer. Det er en proces som vi påbegynder umiddelbart efter udgivelsen. Udover at kunne referere til det spil vi allerede har lavet, er det også væsentligt, at vi har et pitch klart til det næste projekt.

Uanset om vi kan få hul igennem til en eller flere investorer, så er det væsentligt at vi så hurtigt som muligt får etableret en firmahjemmeside. Fordi spil er en global virksomhed, er det vigtigt at have et fast anker som vi kan henvise til hvor alle og enhver kan se at vi er et firma der hele tiden arbejder aktivt imod færdiggørelsen af det næste projekt.   
Samtidigt er det af største vigtighed at få påbegyndt opbyggelsen af et brand og et ry så hurtigt som det overhovedet kan lade sig gøre. Både for at kunne være genkendelige over for kunderne, men især for den interne branchegenkendelse. Som i så mange andre sammenhænge handler det om at have de rigtige kontakter, og for at få det skal vi ud og være synlige på så mange måder som vi kan.   
Dvs. Tilstedeværelse på sociale medier som f.eks. facebook, Twitter og Youtube, samt deltagelse på relevante internet fora.

1. http://vgsales.wikia.com/wiki/Video\_game\_industry [↑](#footnote-ref-1)