Høgskolen i Sørøst-Norge

Fakultet for Allmenvitenskaplige fag
Institutt for Økonomi og Informatikk

-

6117 Bachelorprosjekt

Immortal Guardians TD

Gruppe medlemmer

Studentnummer	Fornavn	Mellomnavn	Etternavn	Oppgave
151690	Phuc	Cao	Tran	Back-end Programmerer
152946	Feda		Saric	Back-end Programmerer
151536	Christer	Gumø	Løberg	Grafisk Design / musikk og lyd

Innhold

1. Introduksjon

Vi har tatt inspirasjon fra rapporten “Utviklingsprosjektet Trolls and Gods” angående oppsettet på rapporten vår. [28]

1.1 Innledning

Oppgaven som vi tok for oss etter en god del diskusjon var å lage et strategispill som kombinerte Tower Defense og Hack’n’Slash sjangerne. Vi bestemte oss for å lage spillet med spillmotoren Unity og programmeringsspråket C# fordi Feda og Phuc hadde Unity/C# som spesialpensum høst 2017.

For at bachelorprosjektet skulle være gjennomførbart så bestemte vi oss for at målet skulle være å lage en first-playable, en demoversjon fordi et komplett spill ikke var realistisk gjennomførbart for oss gitt tiden vi har på å realisere dette.
Vi brainstormet om hva vi ville demoen skulle inneholde av funksjonalitet slik at alle forstod hva som skulle lages.

Siden dette er et prosjekt der arbeidet går ut på systemutvikling så valgte vi en metode som passet for prosjektet vårt og oss som nybegynnere i spillutvikling. En metode som gikk ut på kontrollerbar kaos og hyppig evaluering av funksjonalitet.

Verktøyet og språket C# som skulle brukes til programmeringen var det kjennskap til fra før av, mens Christer skaffet seg et tegnebrett hvor han måtte lære seg programmet som fulgte med.

1.2 Idéen

Det begynte med at Fedja og Phuc i høsten 2017 satt og diskuterte om hva de ville gjøre som bacheloroppgave. Etter en del diskusjon fant de ut av at de hadde lyst til å lage ett spill, men det var først i Januar 2018 at gruppen samlet seg for å diskutere idéer angående hva slags sjanger spillet skulle være.

Vi gikk gjennom en del idéer fra forskjellige typer sjangere og bestemte oss for et spill som vi tenkte kunne passe innenfor tidsrammen vår. Vi endte opp med Tower Defense strategi som hovedsjangeren vår. Men dette følte vi ikke var unikt nok så vi gikk gjennom en del andre sjangre som vi følte passet å kombinere sammen med hovedsjangeren vår; etter en del diskusjon ble vi enige om å legge til hack'n'slash.
Inspirasjon vår kommer fra Warcraft 3 (lagd av Activison Blizzard) sin klassiske Tower Defense custom maps og hack'n'slash spill som Hero Siege og Diablo 3.

2. Administrasjon

2.1 Oppdragsgiver, roller og ansvarsfordeling

Dette er vårt eget prosjekt så vi er oppdragsgiverne.
Gruppen vår består av tre back-end programmerere. Vi manglet en grafisk designer så det vi bestemte oss for å gjøre er å sette Christer på denne oppgaven fordi han var den som var best til å tegne og hadde interesse i dette området.
Siden gruppen vår er så liten så bestemte vi oss for å bruke Random.org til å bestemme prosjektleder. Dette førte til at Feda Saric ble valgt som Prosjektleder. I gruppen vår så hadde ikke prosjektleder noe som helst ansvar ettersom gruppen var så liten og vi så det som unødvendig litt ut i prosjektet.

2.2 Prosjektlokale, utviklermiljø og annen programvare

Vi fant et grupperom på skolen i andre etasje der vi jobbet i ca en måned før vi fant ut av at det var grupperom med TV'er i tredje etasje.
Vi bestemte oss for å flytte oss opp på et av grupperommene i tredje etasje (4-312 til 4-316), 3 ganger i uka fordi det er en TV vi kan koble PC-ene våres opp til for visuell presentasjon for hele gruppen.

På skolen så gjorde vi alt fra diskusjon av spill logikk til rapid prototyping av spillet for å teste ut idéene våres.
Programvarene som vi bestemte oss for å bruke var Unity Engine med C# (Phuc og Feda brukte dem i spesialpensum høsten 2017), Visual Studio men byttet senere til Jetbrains Rider som tekst editor, Github for versjonskontroll og Clip Studio Paint og GIMP for grafisk design.

2.3 Arbeidsmetode

Vi gikk gjennom videoer for spillutvikling og tok inspirasjon fra forskjellige utviklere, spesielt Mark Cerny.
De viktigste punktene vi fikk med oss var at et spill var umulig å defineres fullstendig på starten og at vi måtte utvikle raskt og teste våre idéer for å se hva som funket og hva som ikke funket.
Så vår arbeidsmetode gikk ut på Rapid-prototyping av idéer med Kanban der vi la til og fjernet ting hele tiden under utviklingsprosessen.
Vi brukte Trello.com for vår Kanban tavle.

2.4 Publisitet og sosiale media

Vi har lagd en nettside med hjelp av Wordpress der vi har en forside, kontakt info, download og logg.

Forsiden presenterer gruppen og prosjektet, kontakt info har et skjema der folk kan sende inn feedback/spørsmål, Download er der vi publiserer spillet vårt, Logg oppdaterer vi med ting vi har gjort etter hvert møte med gruppa.

Vi har også en Facebook side der vi har invitert våre testere slik at de kan oppdateres på fremgangen i spillet samt få notifikasjoner på når nye versjoner av spillet er ute.

2.5 Fremdriftsplan

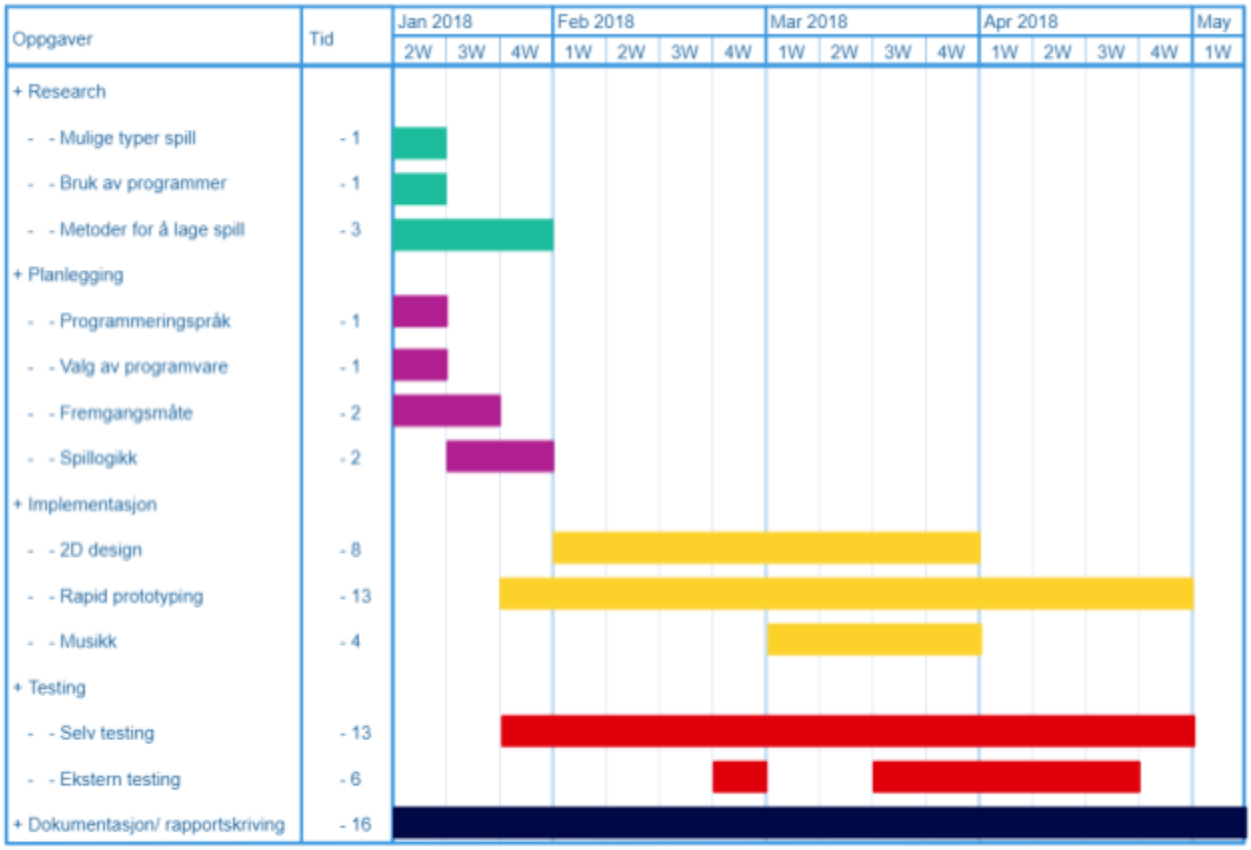


Fig 1, 2.5 Fremdriftsplan

Denne fremdriftsplanen ble bestemt i prosjektbeskrivelsen som vi leverte 02.02.2018.

2.6 Kritisk vurdering og tilbakeblikk

Vi hadde mange ambisiøse idéer for spillet vårt men fant fort ut av det ikke ville være nok tid å implementere alt ettersom mye av det elementære til spillet ble forandret på. Vi hadde en viss anelse på hva vi ville gjøre og brukte veldig lang tid på å teste alt før vi ble fornøyde. Dette førte til at mye ble kuttet ut.

Vi hadde store limitasjoner når det kom til idéene våres på grunn av mangel på grafisk design og animasjoner. Vi tok i bruk mye 2D grafikk fra Unity Asset Store samtidig som Christer tegnet selv, men tegningene ble veldig simple så vi måtte kutte ned på mange av idéene som krevde mer unik grafikk. På den grafiske siden ble ikke vinkelen

perspektivet skulle sees fra fastsatt, og burde vært noe som ble fastsatt i starten av prosjektet.

3. Utviklingsmetode og verktøy

3.1 Planlegging

Valg av utviklingsmetodikk var ganske lett for oss ettersom vi alle hadde hatt smidige metoder faget i semesteret før, og ved hjelp av dette faget så gikk vi i retningen til en hybrid av Scrum og Kanban. Grunnen til dette var at vi visste at vi trengte sprinter men vi ville heller ikke være så tidsbegrenset på grunn av vi virkelig ikke kunne estimere tiden på arbeidet som måtte gjøres. Dette førte til at estimeringene ble veldig grove og unøyaktige i begynnelsen men ble mye bedre utover i arbeidet.
Vi brukte Idéen til Cerny om spillutvikling som gikk ut på kontrollerbar kaos, ikke fokusere på forhåndsplanlegging men istedet på testing av idéer [26]. Og det var akkurat slik det var naturlig for oss å arbeide.

Den første uken bestod av brainstorming av idéer, brukerhistorier til hva slags funksjonalitet vi ville ha med i spillet og grunnleggende funksjonalitet hvor vi plasserte disse i en kanban tavle på Trello.
Etter å ha ført på en god del kort på tavlen vår så var det på tide å kategorisere kortene etter hvor kritisk funksjonaliteten var for spillet og avhengigheter mellom funksjonene.
På grunn av at vi ikke har mye tidligere erfaring med spillutvikling så bestemte vi oss for å ikke ha nøye estimeringer på tiden som skulle brukes, dette førte til at vi kombinerte Scrum og Kanban, der vi hadde fokus på oppgaven i stedet for å ha fokus på tidsrammen til sprinten som vi estimerte.

3.2 Kanban tavlen

Vi valgte å ha ta i bruk en digital løsning på grunn av at vi ikke hadde et fast rom å jobbe i og Trello var den beste løsningen for dette, ikke minst gratis. En stor fordel med Trello var at vi kunne lage sjekkpunkter på kortene slik at vi kunne samle en del kort under ett overordna kort og ha en sjekkliste som viser hvor mye som har blitt gjort. Dette førte til en mer oversiktlig løsning i forhold til om alle arbeidsoppgavene hadde vært et eget kanban kort, og man kan også se avhengigheter mye klarere på denne måten.

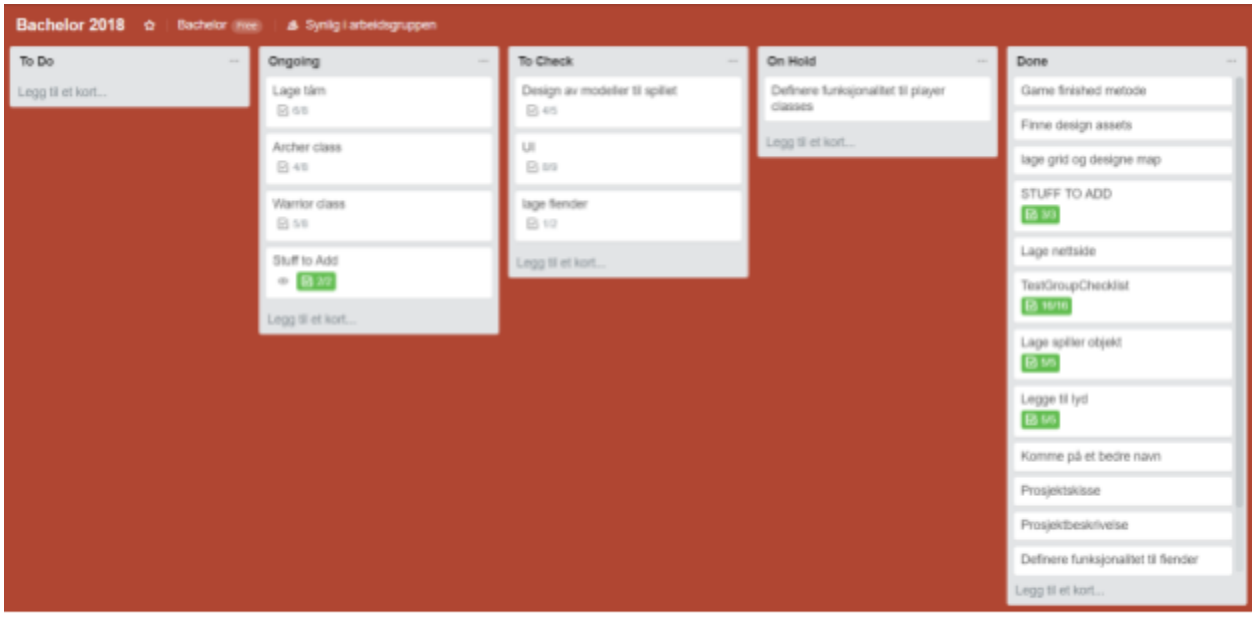


Fig 1, 3.2 Kanban Tavle

3.2.1 To Do

I denne kolonnen så finner man oppgavene som må gjøres, rangert i rekkefølge etter hvor viktig oppgaven er med tanke på avhengighet. I begynnelsen av prosjektet så ville man ha funnet de mest elementære oppgavene på toppen, eksempel på dette er spiller karakter, enkel bane å traversere på osv. Rekkefølgen var bestemt slik at vi begynner veldig smått og deretter bygger oppå den forrige fullførte oppgaven, altså som å bygge med lego klosser.

3.2.2 Ongoing
I denne kolonne så finner man oppgavene er i fokus akkurat nå. Her så bestemte vi oss for å avgrense hvor mange oppgaver som vi hadde lov til å holde på med slik at hver enkelt person ikke ble druknet i arbeid. For Feda og Phuc så var det alltid en større felles oppgave og eventuelt en mindre oppgave ved siden av. For Christer så var det bare en oppgaven om gangen.

3.2.3 To check
Her finner man arbeidsoppgaver som man skal sjekke en siste gang i felles før den enten blir plassert i “Done”, “On hold” eller tilbake til “Ongoing”.

3.2.4 On hold
I denne kolonnen så vil oppgaver som vi ikke er fornøyde med etter “To check” bli plassert dersom vi mener at det er bedre å fortsette med nye arbeidsoppgaver i stedet for å fullføre disse akkurat nå.

3.2.5 Done
Her blir alle de oppgavene som vi mener er helt ferdige plassert.

3.3 Parprogrammering
Etter at idéene våre var definert og vi hadde valgt hvilke metodikker vi ville bruke (Rapid-prototyping), så satt Phuc og Fedja seg ned og kodet mye sammen til å begynne med. Ettersom de begge er nybegynnere så var parprogrammering den beste måten å begynne å jobbe på.

3.3.1 Prinsippet
Hovedprinsippet bak denne teknikken er å luke ut så mange feil som mulig og samtidig la begge partene fordype seg i koden som blir skrevet. Dette hjelper med å holde styr på logikken bak koden og få det optimalisert i ettertid. Et viktig punkt når det kommer til

parprogrammering er god kommunikasjon mellom sjåføren og observatøren. Observatøren må være engasjert i koden og diskutere om sine egne tankeganger når det kommer til implementasjon. [27]

3.3.2 Definisjon

Parprogrammering er en ganske kjent utviklingsmetode som går ut på at to programmerere sitter sammen og samarbeider. Betegnelsen sjåfør og observatør brukes vanligvis til rollene.
Sjåførens oppgave er å sitte med tastaturet og skrive kode, mens observatøren hele tiden observerer arbeidet som sjåføren gjør. Observatørens hovedoppgave er å se etter feil og komme med forbedringer.[27]

3.3.3 Fordeler ved parprogrammering

Bedre kvalitet

I prosjektet vårt så målet vi kvalitet ut ifra antall feil i koden. Det å ha en observatør som hele tiden er med på å se etter feil under implementasjonen hjelper til at et stort antall feil blir oppdaget i nåtid istedet for i etterkant. Selv så merket Feda og Phuc at det var mye lettere å finne og forebygge feil når det var to personer som jobbet sammen.[27]

Økt produktivitet

Her måler vi produktivitet ut ifra hvor mye effektiv kode som er skrevet på gitt tid. Antall kodelinjer er derfor ikke en indikator på at det er produktivt. Når vi jobbet hver for oss senere i prosjektet så skrev hver av oss en del kode, men da vi gikk gjennom arbeidet vårt i felles så la vi fort merke til at koden kunne forbedres. Det var altså lettere å skrive kort og konsis kode når vi jobbet som et par istedet for individuelt.[27]

3.3.4 Var det en metode egnet for oss ?

Parprogrammering funket veldig bra for oss ettersom vi har gjort dette mye før og er komfortable med hverandre. Det hjalp oss som uerfarne kodere å minske så mye feil som mulig tidlig i implementasjonen. Vi fikk lært av hverandre og ble mye bedre på å oppdage feil som den andre overså. Det eneste negative var at dette kostet oss opptil dobbelt så mye tid begge ble brukt til å jobbe på den samme arbeidsoppgaven. Vi hadde jo tross alt bare to uerfarne programmerere i prosjektet.

3.4 Github

Git har vært et veldig viktig verktøy for å kunne holde alle medlemmene oppdatert på utviklingen av spillet vårt. Tidlig i prosjektet så bestemte vi oss for å ta i bruk GitHub og GitHub Desktop, men på grunn av misforståelser så tok vi det ikke i bruk før 2-3 uker etter prosjektstart pga at vi hadde lest at det var veldig mange problemer når man brukte GitHub og Unity sammen. Heldigvis så prøvde vi å laste opp til GitHub og fant ingen store problemer slik som vi hadde lest om.

Git er et verktøy for versjonskontroll og kode styring. Git følger med på hva slags forandringer en eller flere personer gjør på kildekoden og kan vise hvem som har gjort hvilke endringer. Dette gjør Git til en utrolig bra verktøy der flere personer samarbeider om å skrive på den samme koden ettersom det er i stand til å løse konflikter som er et resultat av forskjellig kode på forskjellige PCer. Det er noen situasjoner som Git ikke klarer å løse, og dermed må man manuelt løse problemet.

GitHub er en nettbasert tjeneste. Denne tjenesten er bygget oppå versjonskontroll verktøyet Git. Det er her man laster opp en kopi av Git repository'et sitt slik at man kan synkronisere med andre.

GitHub Desktop er en applikasjon som man kan installere på datamaskinen slik at den kan synkronisere lokal kildekode opp imot det man har på GitHub serveren.

GitHub fungere ved at man først lager et repository på GitHub.com. Repository er vanligvis bruk til å organisere et prosjekt. Når den er laget så vil det opprettes en **master**-versjon og en lokal versjon på datamaskinen. For å ikke ødelegge master-versjonen når man laster opp endringene sine så er det anbefalt å lage en "**Branch**", altså en kopi av master-versjonen som du kan laste opp endringer til og gjøre tester på før man til sist gjør disse endringene på master-versjonen.
Hvis det oppstår en bug eller fører til at prosjektet krasjer så er det mulig å gjøre en såkalt "uncommit" som tilbakestiller siste opplastning. Det gode er at ikke bare siste versjon blir husket, men alle tidligere versjoner også, noe som er hovedgrunnen til at så mange folk idag tar i bruk dette verktøyet. [9]

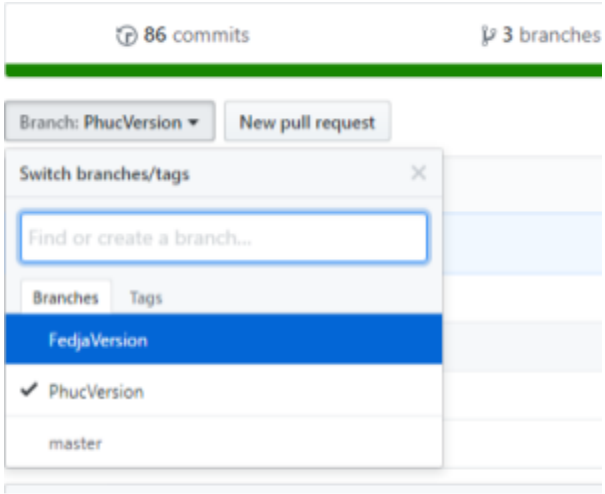


Fig 1, 3.4 Gihub

En viktig del av Git er ".gitignore" filen. Her så skriver bruker inn hvilke filer som Git skal utelate fra prosjektet. Det vil si at disse filene ikke blir lastet opp til versjonen som er på nettet.
Det er vanligvis at filer som er datamaskin spesifikke blir lagt til her. Filer slik som meta-filer, prosjekt filer o.l. Disse filene vil jo være forskjellig på hver enkelt datamaskin

og er lite hensiktsmessig å sende disse mellom hverandre hvis de likevel ikke kan brukes og blir overskrevet av de andre.

Ettersom ingen av oss i gruppa hadde brukt Git før så forårsaket det et lite problem i begynnelsen av prosjektet vårt ettersom vi ikke hadde lagt til en ".gitignore" fil .
Unity lager midlertidige filer spesifikke til editoren, brukeren og datamaskinen , noe som skapte problemer for de andre når de synkroniserte prosjektet etter at det hadde blitt gjort en 'commit'. Dette ble fort løst ved å spesifisere alle disse midlertidige filene som Unity lager inne i gitignore filen slik at GitHub ignorerte disse når den skulle laste opp prosjektet til skyen.
Utenom det lille problemet vi hadde i starten så var det ingen utfordringer ved å bruke GitHub.

Verdt å nevne er at Github har en del standard ".gitignore" filer for de forskjellige programmeringsspråkene og verktøyene som gjør det enkelt å velge riktig fil til prosjektet. Disse standard filene kan finnes her. Vi tok i bruk den som er laget spesifikt for Unity.

```
[Ll]ibrary/
[Tt]emp/
[Oo]bj/
[Bb]uild/
[Bb]uilds/
Assets/AssetStoreTools*

# Visual Studio 2015 cache directory
/.vs/

# Autogenerated VS/MD/Consulo solution and project files
ExportedObj/
.consulo/
*.csproj
*.unityproj
*.sln
*.suo
*.tmp
*.user
*.userprefs
*.pidb
*.booproj
*.svd
*.pdb

# Unity3D generated meta files
*.pidb.meta
```

Fig 2, 3.4 Gitignore

3.5 Objektorientert programmering med Unity

3.5.1 Hvordan programmere i Unity?

Unity er et verktøy som er komponentbasert. I Unity så vil en komponent alltid bestå av mange andre komponenter. “GameObject” er base klassen for alle entiteter i Unity scenen. Under et GameObject så kan man feste mange forskjellige andre komponenter som gir dette objektet ytterligere funksjonalitet og design. Noen eksempler på slike komponenter er kollisjonsbokser, bilder, animatør og scripts.[7]

Når det skal opprettes et skript i C# som skal kobles til et GameObject så må det eksplisitt skrives at det skal arve fra klassen “MonoBehaviour”. Dette er klassen som alle skript i Unity stammer fra. Lages skriptet i JavaScript så vil det automatisk arve fra MonoBehaviour uten at man eksplisitt skriver det.[7]

I klassen MonoBehaviour ligger det en haug med metoder som kan overskrives, hvor de viktigste er “Awake()”, “Start()”, “Update()”, “LateUpdate()”, “FixedUpdate()”.
Dette er metoder som bestemmer hvordan skriptet skal oppføre seg i de forskjellige tilstandene i Unity motoren.

Awake() - Blir kalt på bare en gang rett etter at objekter har blitt initialisert og rett før Start(), brukes til å initialisere eventuelle variabler eller tilstander før spillet starter. Vil bli kjørt uavhengig om scriptet er aktivert eller ikke [2]

Start() - Blir kalt på en gang når skriptet når skriptet starter opp. Vil alltid skje før noen av oppdaterings skriptene [2]

Update() - Kjører hver gang en ny ramme (frame) blir tegnet. Dette skjer flere ganger i sekundet, hvor det vanligste er 60 ganger i sekundet. Vi har ikke lagt på noen restriksjon i spillet vårt så den vil kjøre så fort som datamaskinen klarer [2]

LateUpdate() - Kjøres hver gang en ny ramme blir tegnet, men vil bli utført etter [2]

Update() - Dette er nyttig for ting som kamera bevegelse, hvor objektet posisjon vil endre seg i Update() og kamera vil da kunne oppdatere seg i samme ramme, istedet for i rammen etter [2]

FixedUpdate() - Kjøres hvert x antall frames og brukes hovedsakelig til håndtering av fysikk oppgaver [2]

Unity editor er derfor laget på en slik måte at man definerer GameObjekter og plasserer disse ut i en 3D eller som i vårt tilfelle en 2D scene

3.5.2 Hvorfor valgte vi C# over Unityscript (JavaScript) som programmerings språk?

C# er et objektorientert språk samtidig som språket har veldig lik syntaks sammenlignet med Java, og ettersom vi 3 hadde objektorientert programmering (Java) i 3 semester så var dette en stor faktor til hvorfor vi valgte C# og ikke Unityscript.
JavaScript (Unityscript) kan ikke sammenlignes med JavaScript (ECMAScript) ettersom de er 2 vidt forskjellige språk. Det ga da ingen mening å velge et språk som er bygget for Unity og bare brukes i Unity ovenfor C# når begge har så å si samme funksjonalitet og ytelse.
C# er veldig godt dokumentert på Unity sine sider og de fleste tutorials både på hjemmesiden deres og på internettet er skrevet i C#.
Ifølge en blog-post fra Unity fra august 2017 så er det bare 0.8% av de som bruker Unity 5.6 som bare bruker UnityScript som programmeringsspråk mens 85.4% bruker C# uten en eneste UnityScript fil i prosjektet deres. Siden nesten ingen bruker UnityScript så valgte Unity å fase ut språket. [3]

Siden språket allerede er på vei ut døra så var det naturlig nok ikke noen grunn til å velge noe annet enn C# som programmeringspråk til prosjektet vårt.

3.6 Unity API dokumentasjon og tutorials

Både C# og Unity har en ganske så omfattende dokumentasjon som er lett tilgjengelig for alle. Det var akkurat denne dokumentasjonen som gjorde Unity så enkelt å bruke. Metoder, klasser, funksjonalitet i motoren og hvordan editoren fungerer står godt forklart med eksempler i C#.

En *tutorial* er en opplæringsvideo i hvordan man går frem og tankegangen når man skal lage en spesifikk funksjon i Unity.
Ettersom Unity økosystemet er så stort så finnes det en haug med tutorials laget både av folk som jobber hos Unity (via hjemmesiden) og av andre brukere. Utfordringen i dette er å kunne ta inn det som forklares og implementere det inn i sin egen kode på en god måte.

På grunn av at vi er nybegynnere når det kommer til hvordan vi skal lage de forskjellige funksjonalitetene til spillet vårt så brukte vi en del tid på å se gjennom tutorials som handlet om Tower Defense sjangeren. Dette var for å få kunnskap om hvordan et slikt spill er bygget opp og konseptene rundt programmeringen i et slikt spill.
Disse videoene var veldig nyttige og vi følte vi lærte en god del som vi hele tiden hadde i bakhodet mens vi programmerte spillet vårt.
Du vil finne disse videoene under referanser i slutten av dette dokumentet.

3.7 Grafisk design med Clip Studio Paint og GIMP

Til grafisk design ble Clip Studio Paint og GIMP(GNU Image Manipulation Program) brukt for å lage grafikken vi trengte. Clip Studio Paint var et program som fulgte med tegnebrettet Christer kjøpte, og var ikke gratis programvare på den tiden. GIMP på den andre siden var gratis, men passet ikke helt til å illustrere med.

3.7.1 Tegnebrett

Tegnebrett er et trykkfølsomt brett som kobles til PC'en med USB, og gir en input på samme måte som en mus ville gjort. For å tegne med brettet bruker man en spesialegnet penn. Mange av brettene kan også måle hvor hardt man trykke ned på brettet med pennen, men ikke alle programmer vil ta hensyn til det og behandler det som et vanlig musetrykk. Pennen og brettet hadde også enkelte knapper man kan binde til knapper eller kombinasjoner til knapper på tastaturet. Spesielt nyttig var ctrl + z for å angre på feil.

Brettet som ble kjøpt var "Wacom Intuos Comic Black Pen & Touch Small" og det fulgte med en software bundle med programmer som kunne laste ned. Brettet kunne også måle penntrykket.

Fig 3.7.1 Modellen av tegnebrettet som ble brukt.

3.7.2 GIMP

Først ble GIMP prøvd ut for å sjekke om det kunne brukes til å tegne med, men GIMP hadde problemer med å oppdage penntrykket, og i tillegg bevegde markøren i programmet seg færre ganger per sekund enn musepilen utenfor programmet, så det ble vanskelig å vite hvor markøren ville bevege seg. GIMP var også et mer all-around program og spesialiserer seg ikke på mye på tegning alene, og kanskje mer på redigering av bilder. På grunn av dette skiftet Christer over til Clip Studio Paint, siden det ikke hadde disse problemene, og var mer beregnet for å tegne med uansett.

GIMP var fortsatt noe nyttig siden det viste koordinater på bildet, og hvis man velger en seksjon av bildet kan man se hvor høy og lang den seksjonen er. Hvis man bevegde en del av bildet kunne man også se hvor mange pixler delen var fra startposisjonen. GIMP ble derfor brukt til å posisjonere ting som User Interface.

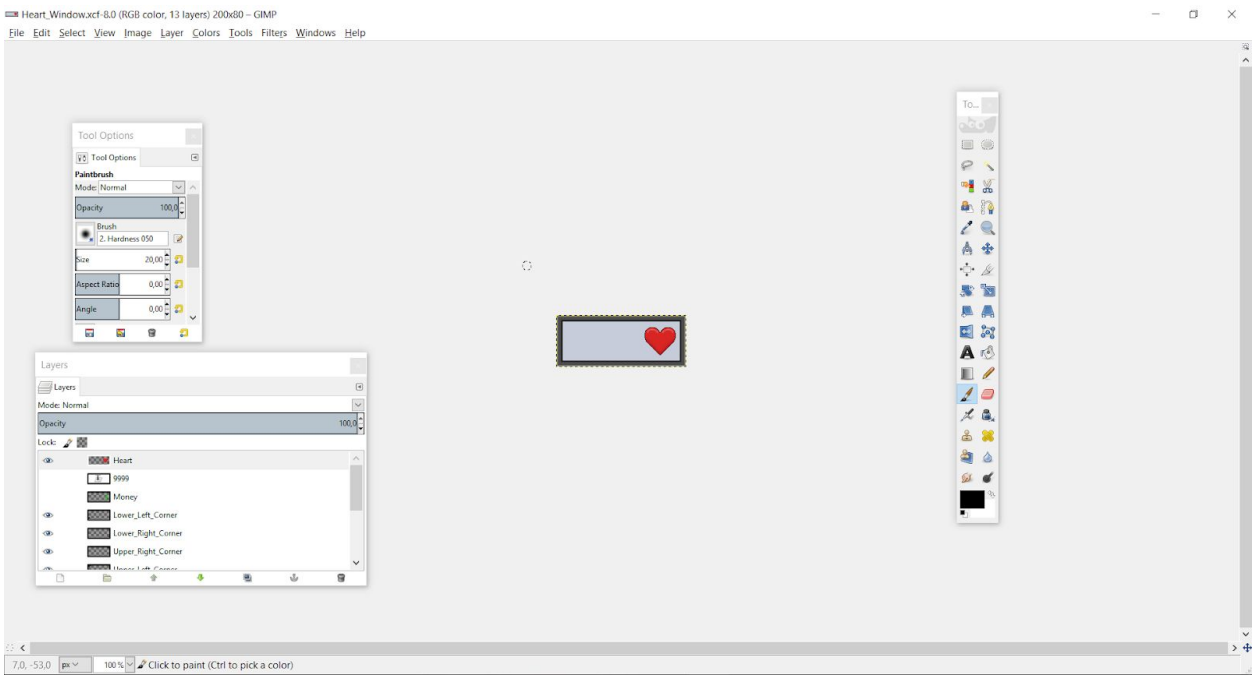


Fig 3.7.2 Bilde av GIMP i bruk.

3.7.3 Clip Studio Paint

Clip Studio Paint var programmet som ble mest brukt til til å tegne med siden det fulgte med tegnebrettet og støttet brettets funksjoner bedre enn GIMP. Programmet var mer beregnet for å lage selvstendige illustrasjoner enn flere bilder som skulle passe sammen, men støttet ikke på noen store problemer der. Et problem med programmet var at det var vanskelig å måle posisjoner og lengder.

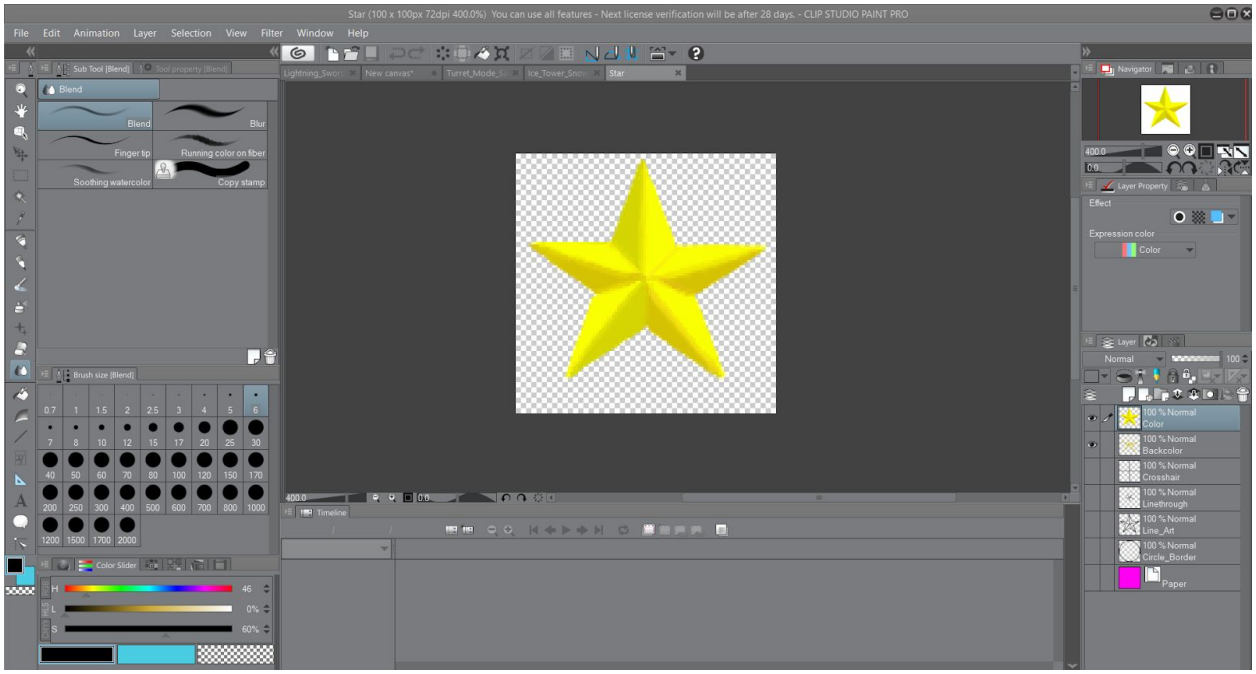


Fig 3.7.3 Bilde av Clip Studio Paint i bruk.

En nyttig funksjon Clip Studio Paint hadde var muligheten til å lage animasjoner. Det gjorde at vi hadde mer kreativ frihet til å lage ”abilities” og tårn siden de ofte trenger animasjoner. Man kunne også endre på hvor mange bilder i sekundet man ville spille av animasjonen. Spesielt nyttig var Onion Skin i animasjon, med den ser man en silhuett av det neste og forrige bilde, så man får en bedre oversikt over posisjonen og flyten av animasjonen.

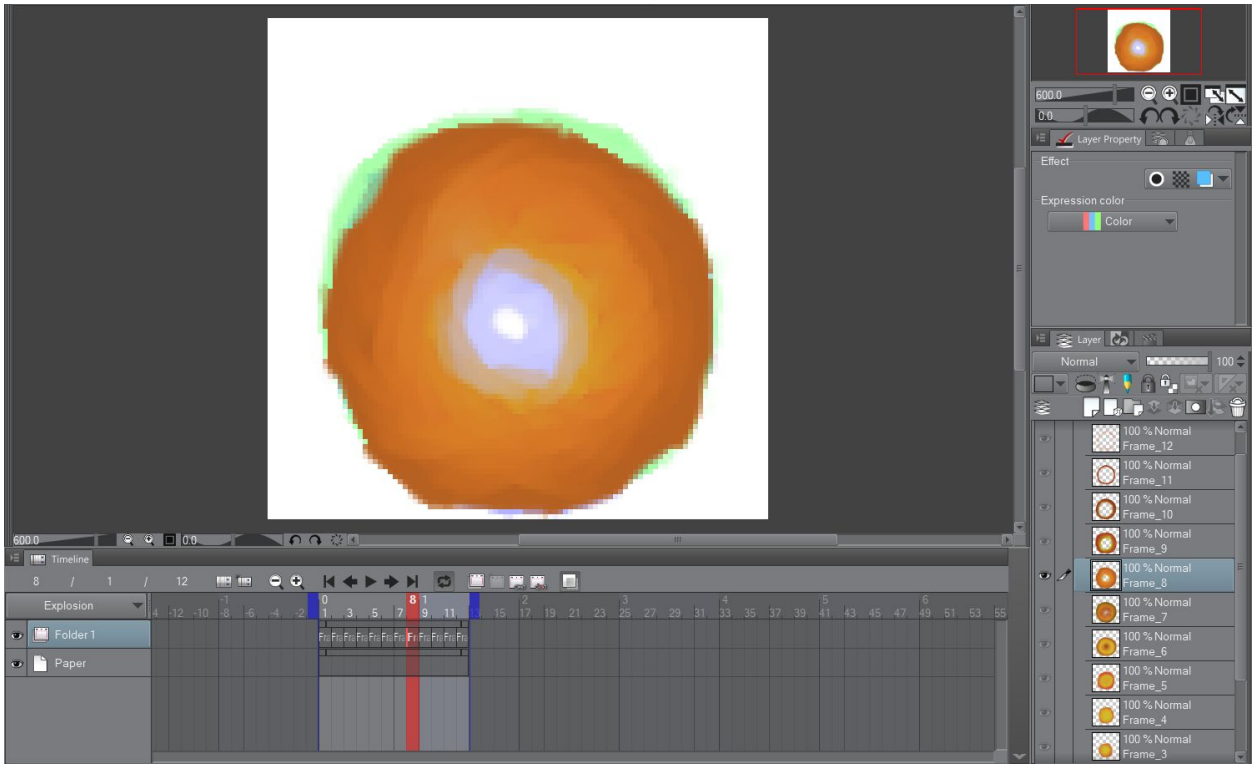


Fig 3.7.4 Bilde av animasjon i Clip Studio Paint med Onion Skin. Den grønne silhuetten ytterst viser hvordan neste bilde ser ut, mens den blå i midten viser det forrige.

4. Systemdokumentasjon

I dette avsnittet så vil vi forklare oppbyggingen bak spillet og grunnen til de forskjellige valgene vi har tatt.

4.1 Oppbyggingen av det logiske i spillet

Etter at sjangeren hadde blitt bestemt, så var det på tide å diskutere oppbyggingen av spillet og hva som skulle være med. Det viktigste vi kom fram til var spiller karakteren, tårn, fiender og målet med spillet som vi skal gå dypere inn i.
Hvis spilleren må tenke gjennom hva han/hun må gjøre for å spille spillet, så er det ikke intuitivt nok. Så vi har lagt mye vekt på at det spillet skulle være intuitivt med en enkel og responsiv kontroller sånn at spilleren kan føle en god flyt gjennom hele spillet.

4.1.1 Meny

Når spillet startes, så vil presenteres med en simpel meny der vi har noen knapper med design fra Unity Asset Store og vår egen logo.
“Play” knapp for å starte spillet, “Settings” knapp for endre oppløsning, fullskjerm og lyd og en “Quit” knapp for å slutte spillet
Det er alltids en “Back” knapp oppe i venstre hjørne som vil ta spilleren tilbake til forrige skjerm.
Når spilleren trykker på play så vil spilleren komme til en skjerm som forklarer litt av hva spillet går ut på og målet med spillet.
Next knappen oppe til høyre vil føre spilleren videre til en forklaring av tårnene som kan velges mellom og litt informasjon om dem.
Neste side vil forklare litt av brukergrensesnittet for spilleren og hva de forskjellige tingene gjør.
Neste side vil spilleren kunne velge sin klasse og samtidig bli presentert informasjon om klassene som kan velges.Etter at spilleren har bestemt seg, så vil spillet starte.

4.1.2 Spiller karakteren

Til å begynne med så hadde vi diskutert og bestemt oss fram til 3 unike klasser som spilleren kunne velge mellom. Under produksjonen av spillet så fant vi fort ut av at vi ble holdt tilbake når det kom til grafisk design og animasjoner. Vi fant forskjellige assets fra Unity Asset Store for design og animasjon men det ble veldig generisk.
Etter mye diskusjon og leting så bestemte vi oss for å gå for det generiske fra Asset Store og heller fokusere på back-end koding og funksjonalitet. Etterhvert så kuttet vi ned fra 3 forskjellige klasser til 2 for å holde oss innenfor tidsrammen.

Når det kommer til kontrollen for spilleren, så gikk vi med kontrollene folk er mest vant med som er W,A,S og D for bevegelse. Vi sørget for at dette var ekstremt responsivt og ikke ville bli holdt tilbake av animasjoner for å få god flyt i spillet. Når det kommer til spillerens spesielle krefter, som bestemmes av hvilken klasse som blir valgt, så vil kontrollene være venstreklikk, høyreklikk og 1-4 for de forskjellige angrepene. Disse kontrollene er ganske typisk for hack and slash sjangeren og vil være veldig intuitivt.

4.1.3 Tårn

Vi hadde samme problemet med tårn som vi hadde med spiller karakteren med design og animasjoner. Men siden tårnene står stille med prosjektiler som beveger seg, så krevde det ikke for mye animasjoner så vi bestemte oss for å tegne våre egne tårn som du kan se i sluttresultatet. De 3 tårnene vi fikk tid til å lage var vanlig Pil tårn, Kanontårn og Is tårn.

Et av de største problemene vi hadde når det kom til tårn var om spilleren skulle få lov til å bygge tårn i slåss fasen der fienden angriper. Vi brukte veldig mye tid på å teste og få dette til å funke når vi burde ha tenkt på om dette var nødvendig for spillet. Vi fikk dette til å funke i spillet vårt men fant fort ut av at hvis spilleren kunne bygge under sloss fasen, så ville det bli lagt mindre vekt på å faktisk bruker spiller karakteren. Får å balansere det sånn at spilleren bruker like mye tårn som spiller karakteren så delte vi det inn i 2 faser, sloss fase og byggefase.

Etter en grundig gjennomgangen av fienden sin Pathfinding, så fant vi en metode som hindret spilleren i å bygge tårn som ville blokkere veien for fienden helt. Dette løste et av de større problemene for design i spillet vårt.

4.1.4 Fienden

Vi bestemte oss for at det eneste fienden skal gjøre i spillet vårt er å gå fra punkt A til punkt B. Dette gjorde det ganske så enkelt for oss å lage logikken for spillet siden det eneste fienden trenger er Pathfinding som vil styre hvor fienden går og liv med indikator som viser hvor mye liv fienden har igjen. Vi fant ikke noe grafisk design for fiender som vi var fornøyde med så vi bestemte oss for å bruke vanlige geometriske figurer for fienden.

Til å begynne med så lagde vi fienden på en tungvint måte der vi lagde et og et objekt og definerte hva slags fiende det var i koden. Vi begynte med dette på grunn av rapid prototyping konseptet vårt der vi ville få ting til å funke på en banal måte for testing. Etter at alt funket som vi ville så så vi inn på forskjellige metoder for å lage fiender som lett kan forandres på. Det er her vi fant en smart design metode fra Unity som heter Scriptable Objects

```
[CreateAssetMenu(fileName = "New Enemy", menuName = "Enemy")]
public class Enemy : ScriptableObject
{
    public Sprite Art;
    public Color Color;

    public int Health;
    public int Money;
    public int Defaultspeed;
    public bool Boss;
}
```

Fig 1, 4.1.4 Enemy Scriptable Object

Her ser du hvordan vi har kodet vårt Scriptable Object med public variabler som definerer de forskjellige verdiene for fienden vår.

På toppen av koden så har vi lagt til [CreateAssetMenu] som vil gjøre det veldig lett for oss å lage nye fiender. Alt som må gjøres er å høyreklikke på prosjekt vinduet og velge create, så kan du se på toppen at Enemy ligger der.

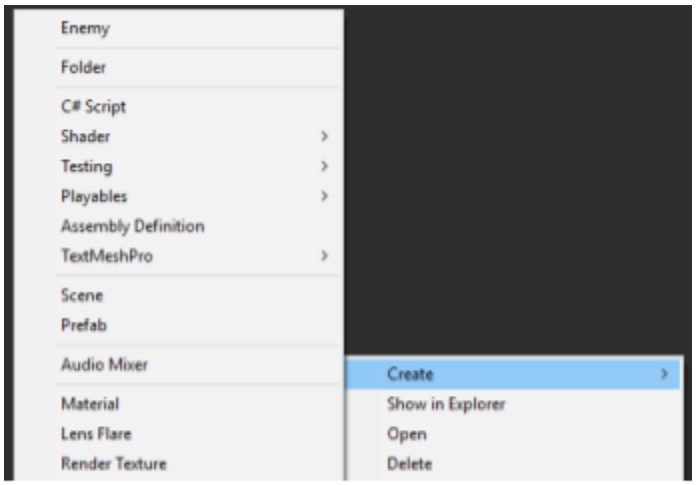


Fig 2, 4.1.4 Asset Menu

Siden det er public variabler for fienden så kan alt defineres i inspector vinduet som vist under.

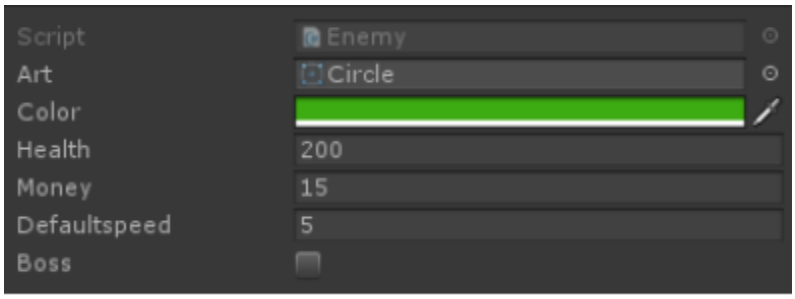


Fig 3, 4.1.4 Enemy Object i Inspector

Alt vi trenger å gjøre da er å instansiere fienden med et fiende objekt for å bestemme verdiene som vist under.

```
public void InitializeStats(Enemy enemy)
{
    _startHp = enemy.Health;
    _money = enemy.Money;
    _defaultSpeed = enemy.Defaultspeed;
    GetComponent<AIPath>().maxSpeed = _defaultSpeed;
    transform.GetChild(0).gameObject.GetComponent<SpriteRenderer>().sprite = enemy.Art;
    transform.GetChild(0).gameObject.GetComponent<SpriteRenderer>().color = enemy.Color;

    _boss = enemy.Boss;

    _hp = _startHp;
}
```

Fig 4, 4.1.4 Initialize Stats

I teorien så høres fienden vår ganske lett ut å implementere, men vi undervurderte hvor lang tid det faktisk tok å implementere det viktigste for fienden. Nemlig Pathfinding som vi skal gå dypere inn i på neste punkt.

4.1.5 Managers

En viktig del av programmet vårt er data management. Det er koder som må kjøres men som ikke tilhører et spesifikt objekt og skal brukes gjennom hele programmet. Det er her Managers kommer inn for å styre alt som trengs. Vi har GameManager som styrer alt fra museklikk til tilstanden til spillet. TowerManager som styrer bygging av tårn. UIManager som styrer alt det grafiske brukergrensesnittet til spillet. WaveManager som styrer hvilken runde spilleren er på og hvilken fiender som skal komme. AudioManager som styrer alt av lyd i spillet.

4.1.6 Systemarkitektur

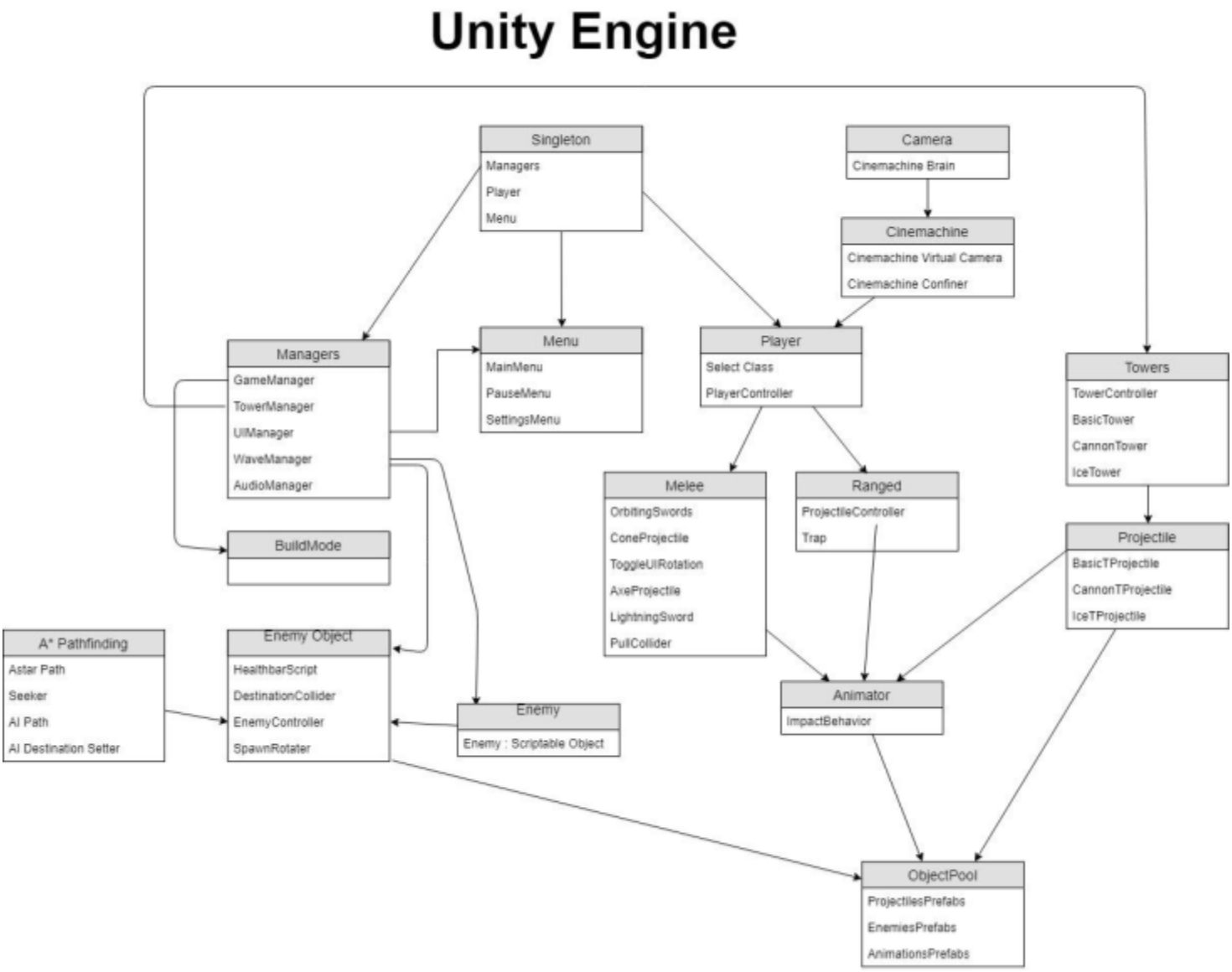


Fig 1, 4.1.6 Systemarkitektur

På bilde over så kan man se hvordan koden vår henger sammen og hva som kjører hva.

4.2 A* Pathfinding

Konseptet er greit nok å forstå men å implementere koden på en god og rask måte er veldig vrien. Så det vi bestemte oss for å gjøre var å lete rundt på nettet etter et eksempel på en slik algoritme brukt i Unity sammenheng ettersom vi definert ikke var de første til å bruke A* i et spill.

Heldigvis så fant vi Aron Granberg, som er en spillutvikler og jobber aktivt på et A* pathfinding prosjekt. Dette er en Unity ressurs med både betalt og gratis versjon der lisensen er en Unity Asset Store lisens, som vil si at den kan brukes til kommersielt bruk.

Koden hans oppfylte alle våres krav og mer.

På vårt prosjekt så er det bare fienden som bruker A* pathfinding for å finne nærmeste vei til mål.

Nedenfor vil vi forklare hva A* er, hvordan det fungerer og hvordan man bruker ressursen laget av Aron Granberg. [12][13]

4.2.1 Hva er A* pathfinding?

A* Search Algorithm er et konsept som først ble beskrevet i 1968 av Peter Hart, Nils Nilsson og Bertram Raphael fra Standford Research Institute (som nå er SRI International).

Denne algoritmen er en videreutviklet versjon av Edsger Dijkstra 1959 Algoritme med bedre ytelse ved bruk av heuristisk metoder for å veilede søket sitt.

Heuristikk, innenfor IT, er en teknikk for å finne seg fram til nærmeste løsning på minst mulig tid. Svaret du får er ikke perfekt men det er nærme nok perfekt og bruker mindre tid enn andre metoder som prøver å finne det perfekte svaret.

A* bruker en vektet graf, noder med forskjellige verdier og to lister (åpen og lukket liste) for å finne seg fram til sluttnoden (målet).

Den kalkulerer kostnad for traversering og regner seg fram til den nærmeste veien til målet. [12][13]

4.2.2 Kalkulering av kostnad for traversering

Formel: $F_{kostnad} = G_{kostnad} + H_{Kostnad}$

$G_{kostnad}$ = Distansen fra start noden
$H_{kostnad}$ (heuristikk) = Distansen fra slutt noden
$F_{kostnad}$ = Summen av G og H

Fig 1, 4.2.2 Node

Her er et eksempel på hvordan formelen er representert i en node

Vektet Graf

Her har du en vektet graf med start node A, omringet av noder med kostnads verdien sin.
Kostnaden for horisontalt og vertikalt traversering ble regnet som 1 og de brukte pytagoras for å regne ut den diagonale verdien.
$H^2 = x^2 * y^2$ der x og y er 1.
$H = \sqrt{1^2 + 1^2} \approx 1.4$
Deretter ganges alt med 10 for å unngå komma tall. [13]

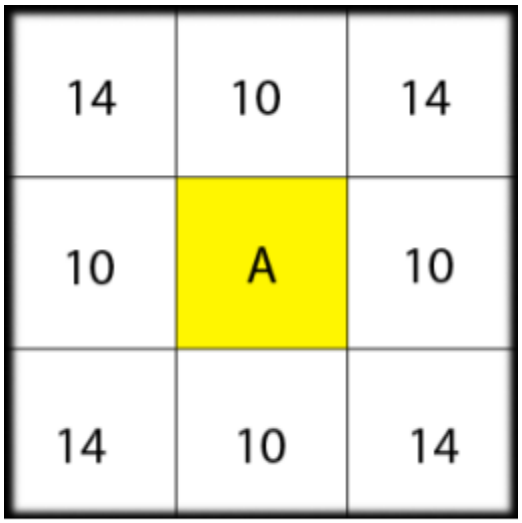


Fig 2, 4.2.2 Vektet Graf

4.2.3 Konkret eksempel

På bildet så kan man se start node A og slutt node B. Start noden vil bli lagt inn i den åpne listen

Det A* vil begynne med er å kalkulere G, H og F verdiene til alle nodene rundt start noden og legge dem til ÅPEN listen (grønne noder ligger i åpen liste).

Deretter vil den finne noden(e) med lavest F verdi fra den åpne listen og legge den i en lukket liste. Noden som blir lagt inn i lukket liste vil alltid ha en referanse til parent noden den kom fra.

Hvis to noder har samme F verdi så vil den se på H verdien for å sammenligne hvilken noder som er nærmest mål (B) og legge den inn i en lukket liste.

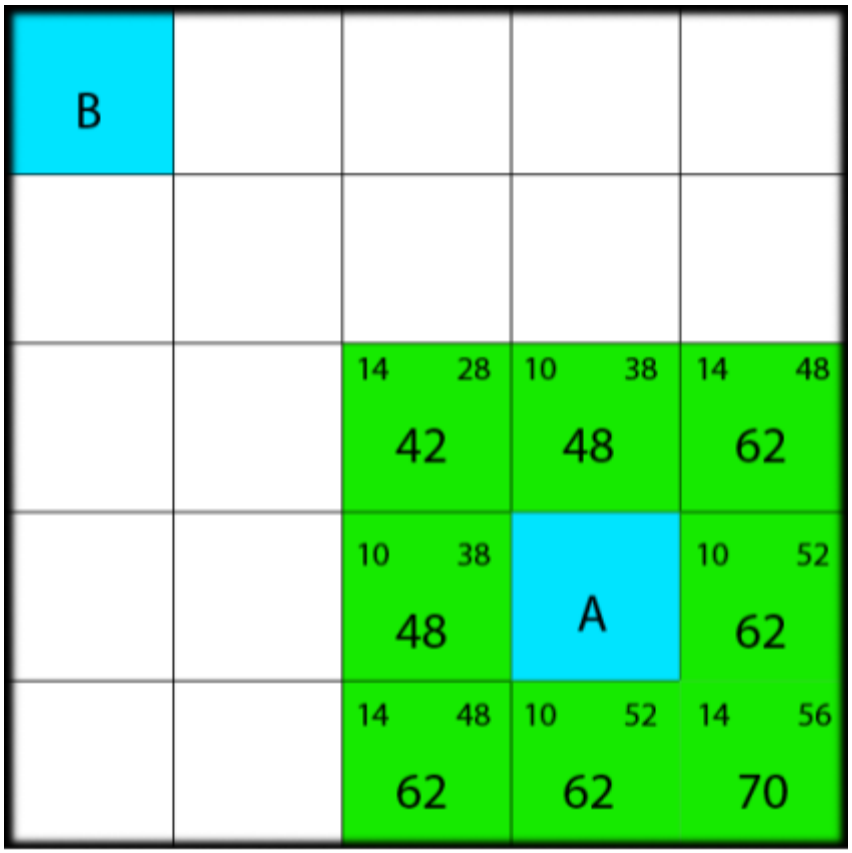


Fig 1, 4.2.3 Konkret Eksempel 1. iterasjon

Hvis 2 noder har akkurat samme verdier med lavest F verdi, så vil begge bli lagt inn i den lukkede listen.

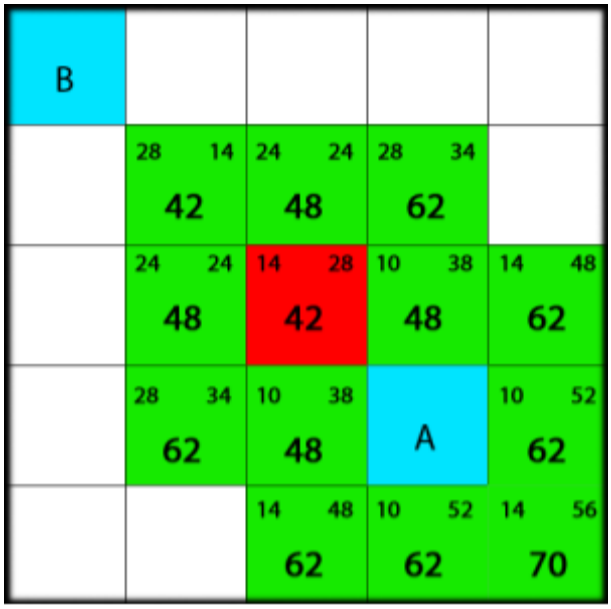


Fig 2, 4.2.3 Konkret Eksempel 2. iterasjon

Etter alle nodene med lavest verdi har blitt lagt inn i listen, så vil algoritmen gå gjennom den lukkede listen (røde noder) og kalkulere kostnads verdiene for alle naboene rundt og legge dem til den åpnende listen.

Så vil koden gjøre det samme som forrige iterasjon og gå gjennom den åpnende listen og finne noden med lavest F verdi og legge den til en lukket liste

Når H verdien til en node er 0 eller slutt noden har blitt funnet som vil si at algoritmen har nådd målet sitt, så vil den gå tilbake og se på sine forrige noder den gikk gjennom for å komme seg til mål.

Nå har algoritmen en liste med noder som representerer den nærmeste veien fra start noden til slutt noden og vil sende den tilbake til bruker. [12] - [15]

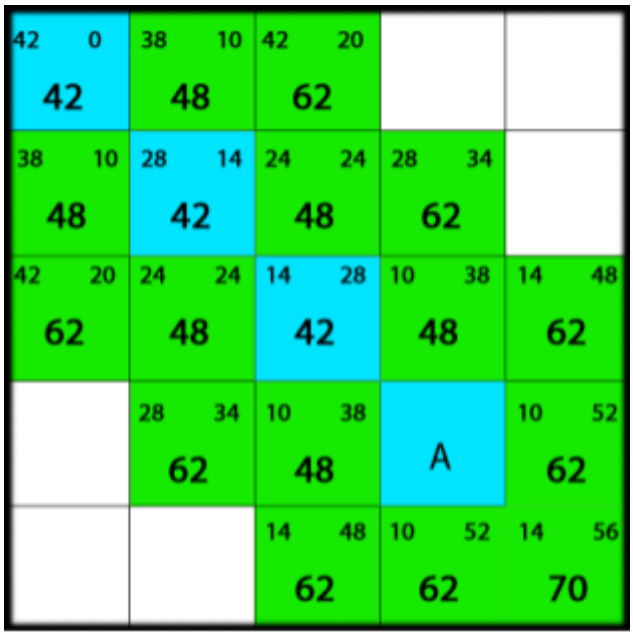


Fig 3, 4.2.3 Konkret Eksempel 3. Iterasjon

Her er en link til et gif med eksempel på åssen A* behandler pathfinding
https://imgur.com/gallery/vC71luv

4.2.4 Pseudo Kode

```
ÅPEN          // Liste med nodene som skal evalueres
LUKKET        // Liste med nodene som har blitt evaluert
legge start noden til ÅPEN

loop
  current = node i ÅPEN med laveste F_Kostnad
  fjerne current fra ÅPEN
  legge current til LUKKET

  hvis current er slutt node ELLER h_kostand er 0      // en vei har blitt funnet til mål
    return

  foreach nabo av current node
    hvis nabo ikke er traversabar ELLER nabo er i LUKKET
      hoppe til neste nabo

    hvis ny vei til nabo er kortere ELLER nabo ikke er i ÅPEN
      set f_kostnad til nabo          // ved å bruke g_kostnad og h_kostnad
      set parent av nabo til current
      hvis nabo ikke er i ÅPEN
        legg nabo til ÅPEN
```

4.2.5 Implementasjonen av A* Pathfinding Project

Det mest sentrale skriptet i A* Pathfinding Project er ‘astarpath.cs’, det oppfører seg som et sentralt knutepunkt for alt annet. Det er denne klassen som håndterer alt som har med pathfinding systemet, kalkulerer alle stier og lagrer all informasjon. Man legger dette skriptet inn som en komponent på et tomt objekt som man kaller “A*”. Inne på dette objektets inspektør vil man da finne en seksjon for AstarPath hvor man kan opprette grafer og gjøre endringer på innstillinger. Denne klassen er en ‘Singleton’ klasse, som betyr at det bare skal eksistere et instans av denne klassen i scenen. [12] Singleton blir forklart mer utdypende i 4.3.

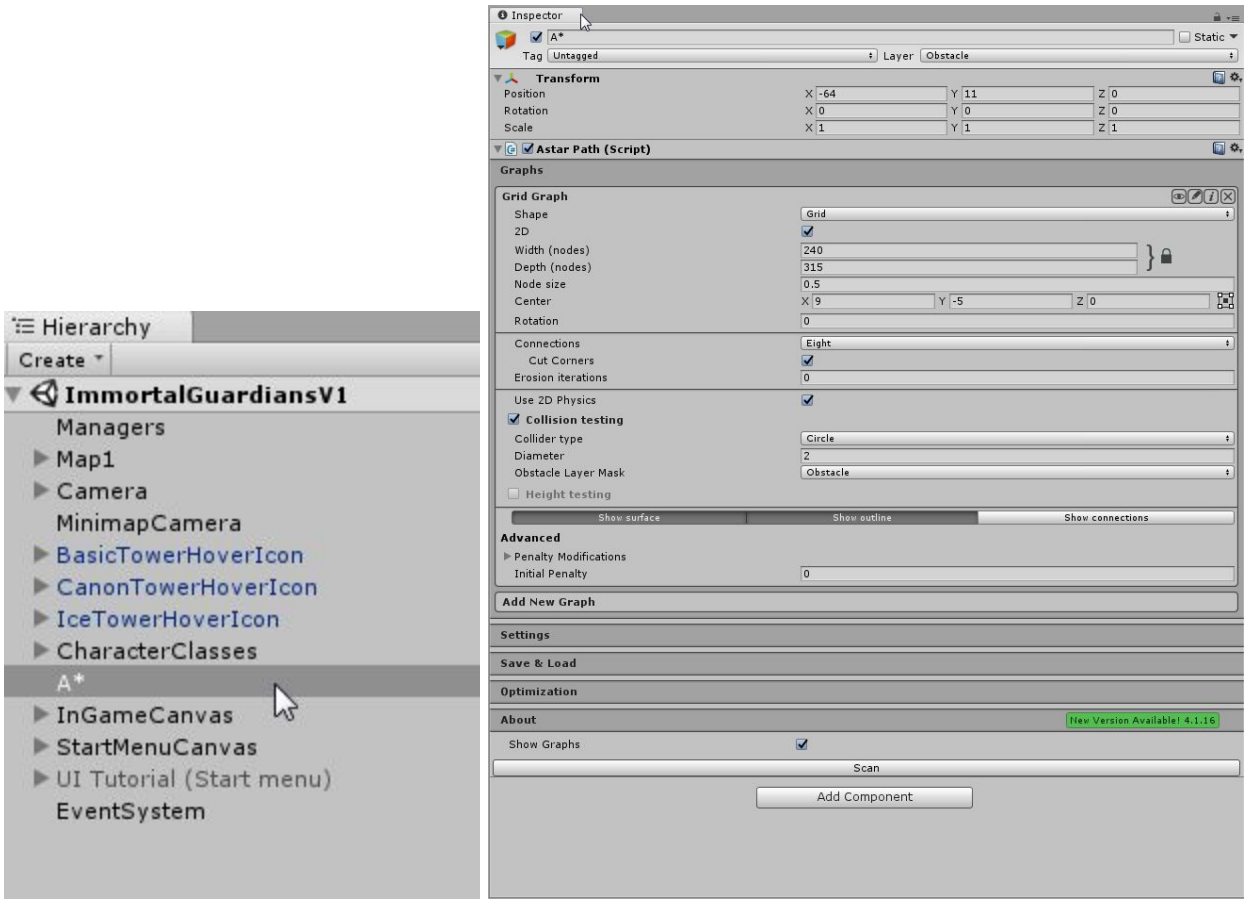


Fig.1 og Fig.2 4.2.5 Objektet i hierarkiet og inspektøren til objektet etter at ‘astarpath.cs’ er lagt til

Den nest viktigste komponenten er ‘Seeker.cs’, denne komponenten blir lagt til på hvert GameObjekt som skal bruke Pathfinding, som i vårt tilfelle er fiendene.
Denne komponenten håndterer alle kall gjort angående stien til det objektet den tilhører, den håndterer også etterbehandling av stien ved bruk av modifikatorer. Den er ikke absolutt nødvendig men den gjør jobben mye enklere. [12]

For å få AI (fienden) til å bevege seg så har han inkludert en del bevegelses skript i denne pakken (eksempel AIPath, RichAI, AILerp). Vi bruker ‘AIPath’ som er standard skriptet.
Dette skriptet vil prøve å bevege seg til en gitt destinasjon. Måten vi satt destinasjon på var å opprette et tomt objekt (usynlig) med ‘AIDestinationSetter’ komponenten, noe som gjør at alle med ‘AIPath’ vil følge etter dette objektet og bare plasserte det hvor vi ville destinasjonen skulle være.
Ovenfor snakket vi om modifikatorer, dette er små skript som man kan legge til på GameObjekter som har Seeker komponenten slik at den modifiserer stien enten etter at man har kalt på stien. Nedenfor vil du kunne se eksempel på dette. [12]

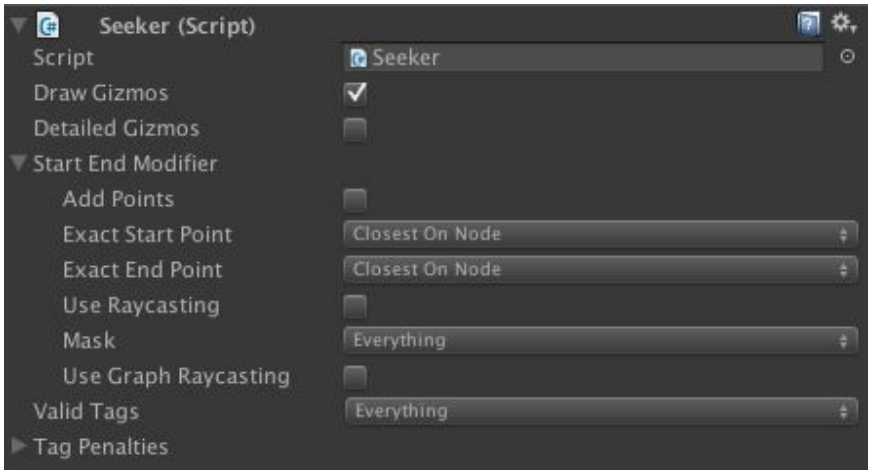


Fig.2 4.2.4 Seeker komponenten

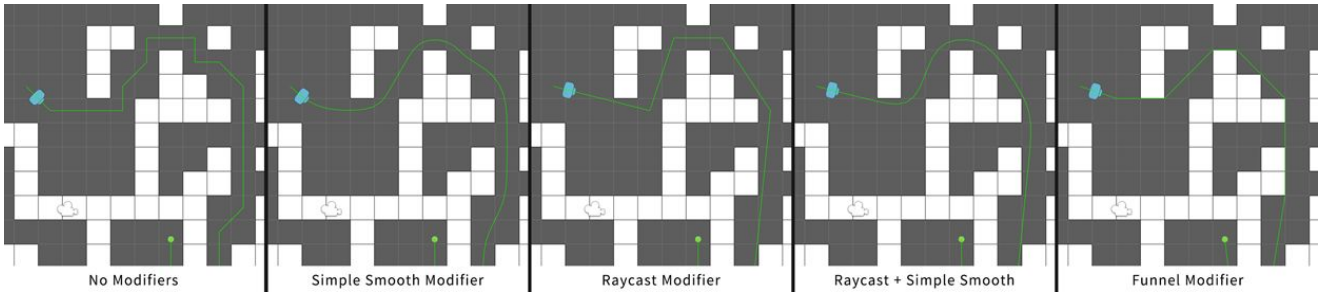


Fig.3 4.2.4 Hvordan stien til objektet vil se ut med de forskjellige modifikator skriptene

4.3 Singleton

4.3.1 Hva er Singleton?

‘Singleton Pattern' er et design mønster som begrenser initialisering av klassen til kun et objekt, dvs det bare er et instans av klassen. Dette er veldig nyttig når man bruker et objekt for å koordinere handlinger gjennom hele programmet. Som i vårt tilfelle blir brukt i våre Managers som håndterer hele spillet. [17]

Singleton design mønster løser problemer som

- å sikre at det er bare en instans av klassen
- Hvordan kan klassen aksesseres på en lett måte
- Hvordan klassen kan kontrollere sin initiasjon

4.3.2 Implementasjon

```
// An abstract class used to access scripts

public abstract class Singleton<T> : MonoBehaviour where T : MonoBehaviour{

    private static T _instance;

    public static T Instance
    {
        get
        {
            if (_instance == null)
            {
                _instance = FindObjectOfType<T>();
            }

            return _instance;
        }
    }
}
```

Ingen grunn for pseudo kode her siden koden er så kort

Fig 1, 4.3.2 Singleton Implementasjon

Her oppretter vi en Singleton klasse med typen T som er en Any Type notasjon og må implementere MonoBehaviour som er Unity sin base klasse og T må også implementere MonoBehaviour for at det fortsatt skal funke som en vanlig Unity klasse.

Med denne koden så kan du implementere en klasse under Singleton hierarkiet og bruke dens get metode for å få tak i klassen og aksessere dens public variabler og metoder

4.4 Object Pool

4.4.1 Hva er Object Pool?

Object pool teknikken er en kjent metode for å redusere ressursbruken til et program. Metoden går ut på å gjenbruke objekter som allerede er instansiert for å hindre kostnaden ved å instansiere/ødelegge objekter. Instansiere betyr å opprette et objekt. [17]

4.4.2 Fordeler

Når man jobber med store mengder objekter av samme type med kort levetid så kan det være veldig dyrt å kjøre for spillet hvis man instansierer og ødelegger dem veldig ofte på kort tid.

Object pool teknikken lager et sett med objekter som kan bli gjenbrukt.

Når et objekt trengs, så vil metoden sende en forespørsel til Object pool listen.

Hvis det allerede finnes et objekt som er instansiert men ikke er aktivt, så vil Object pool metoden aktivere objektet og sende den til metoden som gjorde kallet. Dette vil unngå ekstra ressursbruk som skjer når man instansierer et nytt objekt.

Hvis objektet ikke ligger i listen enda eller alle er opptatt (aktivert), så vil det bli instansiert et nytt objekt som blir lagt inn i listen over objekter i pool'et, deretter blir det aktivert før det sendes videre til metoden som gjorde kallet slik at den kan tas i bruk.

Etter at metoden er ferdig med objektet så vil metoden deaktivere og tilbakestille objektet slik at Object pool listen ser at objektet ikke lenger er i bruk og kan derfor gjenbruke det igjen.

4.4.3 Ulemper

I noen tilfeller, så vil pooling av objekter ta opp veldig mye minne hvis man instansierer for mange objekter uten å deaktivere dem. Så man må være obs på hvor det er nyttig å bruke object pool og hvor det ikke er nyttig.

Når man bruker Object Pool så er det viktig at man husker å resette objektet når det sendes tilbake til object pool sånn at det ikke er lagret noen verdier fra forrige bruk.

4.4.4 Pseudo Kode

```
OBJEKT LISTE        // Liste over objekter programmet kan bruke
POOL LISTE          // Liste over objekter som har blitt instansiert og lagt inn i pool'en
Public GetObject    // Metode for å få tak i et objekt
  Foreach objekt i POOL LISTE   // Sjekke om objektet finnes og kan gjenbrukes
    Hvis objekt er forskjellig fra objektet programmet spør om ELLER objekt er aktivt
      Hopp videre
    Hvis ikke
      Aktiver og returner objekt

  Foreach objekt i OBJEKT LISTE         // Sjekke hvilket objekt som skal instansieres
    Hvis objekt er forskjellig fra objektet programmet spør om
      Hopp videre
    Hvis ikke
      Instansier et nytt objekt av typen brukeren spør om
      Legge objekt inn i POOL LISTE
      Returner nytt objekt

Public void ReleaseObject           // Generisk metode for å deaktivere et objekt
  Deaktiver objekt
  Tilbakestill objekt
```

4.4.5 Implementasjon

Siden det bare er et instans av Object Pool så kan den legges under Singleton hierarkiet og bli brukt fra hvor som helst.

I Unity så bruker vi en String for å sammenligne objekter siden alle objekter har et navn som vi kan lett få tak i ved å bruke GameObject.name.

Når et objekt blir instansiert så vil navnet bli OriginaltNavn (clone), og for å fjerne clone delen så setter vi bare objektets nye navn til original navnet.

Vi bruker Object Pool for alt med kort levetid i programmet vårt.

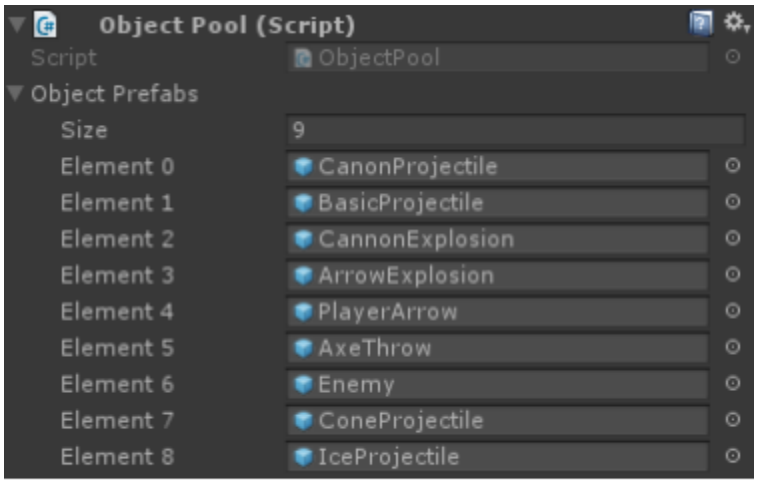


Fig 1, 4.4.5 Object Pool liste

Alle prosjektiler og eksplosjon animasjoner av spilleren og tårn er lagt inn i Object Poolen siden de har en veldig kort levetid og blir gjenbrukt veldig ofte.

Vi har “Enemy” i pool’et også men den blir ikke brukt fordi det var problemer med Aron Granberg sin Pathfinding når Enemy objektet ble brukt om igjen selv om det står i dokumentasjonen at dette var fikset. Noe vi må ta opp med Aron på forumet hans der han er veldig aktiv.

Dette har ikke så mye å si for hvordan spillet vårt er akkurat nå, men vi kan se at problemer oppstår hvis vi oppretter altfor mange fiender.

4.5 Kamera funksjonalitet med Cinemachine

4.5.1 Hva er Cinemachine?

Cinemachine er et rutinert kamera system brukt i Unity og er gratis fra Unity sin Asset Store. Det er et kraftig verktøy som har blitt jobbet på i over 20 år og som gjør det veldig lett for brukeren å lage scener som kan sammenlignes med film.

Cinemachine har mange forskjellige funksjoner, men det vi bruker Cinemachine til er en funksjon som heter Follow Mode.

Follow Mode er et kamera som vil følge etter spilleren som i spillet vårt, men forskjellen med Cinemachine i motsetning til vanlig kamera, er at den glatter bevegelsen til kameraet sånn at det ikke ser så snappy ut når man beveger spilleren.
Spilleren vil bevege seg først og kameraet vil akselerere på en glatt måte mens den følger spilleren.

Med Cinemachine så kan man definere en såkalt ‘Confiner’ som vil begrense hvor langt kameraet kan bevege seg. Som i vårt spill så vil kameraet stoppe å bevege seg når den kommer nær en vegg. [6]

4.5.2 Implementasjon

For å bruke Cinemachine i prosjektet, så må man legge på en Cinemachine Virtual Camera (CM vcam1) som et barn (child) av kamera objektet som vist under.

Fig 1, 4.6.2 Kamera i Hierarki

Kamera objektet må ha et Cinemachine Brain script lagt til som indikeres med et rødt ikon til høyre for objektet i hierarkiet som vist på bildet over.

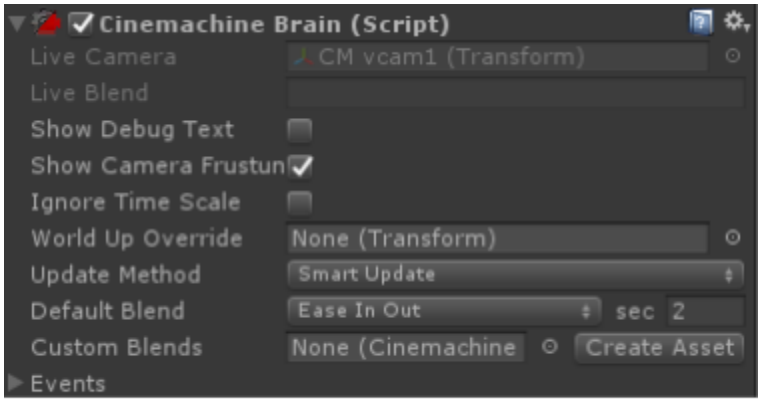


Fig 2, 4.6.2 Cinemachine Brain

Cinemachine Brain brukes til å definere hvilket kamera Cinemachine Virtual Camera skal bruke.

I selve Virtual Camera scriptet så har den en Follow metode der du definerer hvilket objekt kameraet skal følge etter som vist under

Fig 3, 4.6.2 Cinemachine Virtual Camera

Her definerer du også hvor langt unna kameraet skal ligge i forhold til objektet under ‘Lens’.

Man kan legge på en Confiner, som vi har gjort, der vi definerer det begrensede område med en Polygon Collider som vi kaller for “CameraCollider”.

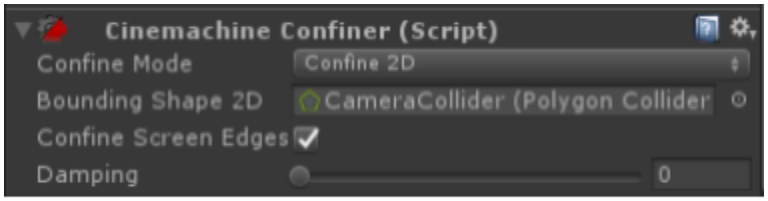
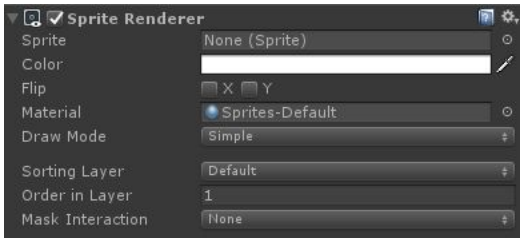


Fig 4, 4.6.2 Cinemachine Confiner

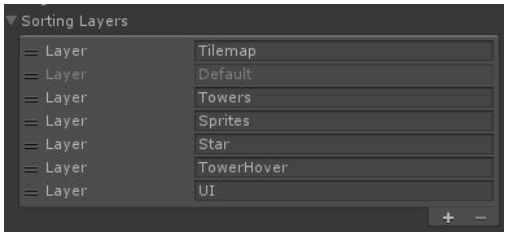
4.6 Det grafiske

4.6.1 Sprite

Sprite-klassen holder på 2-dimensjonale bilder. SpriteRenderer-klassen står for å presentere Sprite-objekter og brukes av GameObjects for å vise bilder. SpriteRenderen kan også endre på presentasjonen av bildet, slik som å endre farge, speile bildet, og forandre størrelse.



En veldig viktig funksjon i Sprite Renderer er Sorting Layer og Order in Layer. Dette er det som bestemmer hvilket bilde (sprite) som skal vises i forgrunnen og hvilket som skal vises i bakgrunnen.



Rekkefølgen som vises på skjermen er UI -> TowerHover -> Star -> Sprites -> Towers -> Default -> Tilemap. Så det tar prioritet fra bunnen til toppen.
Den andre viktige delen er hvilken Order in Layer verdi den har, som vi har bestemt manuelt på mange objekter men gjør det kodemessig for tårn.
Når vi bygger et tårn så tar vi y-akse verdien til mus posisjonen og ganger den med -1 for å bytte fortegn. Dette vil sørge for at tårnene blir vist riktig på skjermen.

En annen flott ting med Unity er at det støtter Photoshop filer(.psd) som vanlig .png bilder, så man kan lett redigere bilder uten å måtte eksportere dem som .png/.jpg bilder.

4.6.2 Tilemap funksjonen

Unity sin Tilemap funksjon, som kom ut med versjon 2017.2 den 12. oktober 2017, er en relativt ny funksjon for tegning med Sprites i Unity. Siden dette var så nytt så hadde vi problemer med å tilegne oss kunnskapen rundt det siden det var lite lære materiale ute på nettet. Etter at vi hadde lekt rundt med funksjonen og funnet ut av konseptet bak Tile Mapping, så skjønte vi at dette er et veldig kraftig verktøy som videre hjalp oss mye når det kom til å designe banen vår.

Tilemap gjør det enklere og raskere å lage 2D baner ved å ta i bruk sprites og et grid som følger med tilemap-et. Funksjonen består av en rekke andre systemer som jobber sammen og er listet opp under.

- **Tile** Assets
  Grid GameObjects
- The **Tilemap Palette**
- Custom **Brushes**

For å bruke Tilemap i Unity så høyreklikker man i hierarkiet og velger 2D Object -> Tilemap. Dette vil opprette et GameObject med Grid komponenten og et child object Tilemap.

Deretter så oppretter man et nytt Tilemap Palette der man kan dra inn bilder for å gjøre de om til tiles.
En tile er en asset som refererer til en Sprite. Når vi har Tiles så er det så enkelt som å velge en Brush, velge en Tile og tegne i vei på scenen. [4]

4.6.3 Third party sprites

Vi bestemte tidlig i prosjektet at det meste av det grafiske skulle hentes fra forskjellige kilder mens en liten del skulle tegnes selv. Det var fordi bare Christer hadde kunnskap om grafisk design og den kunnskapen var ganske så grunnleggende.

5. Konklusjon

5.1 Rammene

Et spillprosjekt kan vare i flere år før det blir ferdig så det vi prøvde var å sette rammene slik at dette var noe vi kunne gjøre ferdig innenfor tidsrammen vi hadde. Dette er beskrevet i prosjektbeskrivelsen. Der beskrev vi hvilke resultater vi forventet og hvordan vi skulle oppnå dem.

En estimering av tid for prosjektet ble aldri gjort fordi vi egentlig ikke hadde noen anelse om hvor lang tid et slikt prosjekt trenger. Det vi i stedet gjorde var å bestemme hvor mye vi skulle jobbe i felles og hver for seg per uke utifra arbeidsomfanget til et 15 poengs fag.

Vi holdt oss til arbeidsdagene mandag, onsdag og fredag ganske så godt utenom påskeferien hvor alle dro hjem og vi ikke kunne møtes. Det var også noen avvik der vi jobbet tirsdag, torsdag og lørdag på grunn av konflikter i timeplanen. Tidspunktet for oppmøte ble forandret på tre ganger. Mengden vi skulle jobbe ble ikke forandret, som var minst 4 timer per arbeidsdag.

5.2 Arbeid og utviklingsmetode

Arbeidsmetoden var ganske så viktig med tanke på hvor fleksibelt vi måtte være fordi et spill aldri er fullstendig definert i begynnelsen. I spillutvikling så vil mange endringer skje noe som også skjedde i vårt prosjekt. Uten smidig og Cerny metodikken så ville vi aldri ha klart å kommet like langt i prosjektet.

5.2.1 Kanban

Vi fulgte kanban regler hvor vi satt mindre vekt på hvor lang tid vi skulle bruke på oppgavene og mer vekt på at oppgavene ble ferdige. Kanban tavlen ble nyttig fordi vi hadde en god oversikt over oppgavene som måtte gjøres og avhengighetene til hver av dem. Dette gjorde det lett for oss å starte smått og bygge på det forrige arbeidet. Det ga også muligheten for å gjøre endringer, slette eller legge til nye oppgaver uten å måtte starte på nytt med planleggingen.

5.2.2 Cerny

Denne kaotiske arbeidsmetoden funket for oss men den krevde ekstra mye motivasjon og disiplin. Det var noen dager som gikk bortkastet der motivasjonen var for lav og andre dager der vi jobbet fra soloppgang til langt ut på kvelden.
Uten fastsatte mål for hver arbeidssesjon så var det veldig mye avsporing fra det vi satt oss ned for å arbeide med. Dette førte til noen sesjoner der vi slet hjernene våres ut med ideer men ingenting ble gjort ferdig og vi måtte gi opp for å fortsette en annen dag.

Alt i alt så kom vi oss henholdsvis i mål med det vi hadde lyst til å implementere, men på en kaotisk måte. [26]

5.2.3 Tilegning av kunnskap

Vi hadde en del kunnskap om Unity og C# fra før siden Phuc og Feda hadde det som spesialpensum i semesteret før bacheloroppgaven. Dette førte til at kunne hoppe rett på arbeidsoppgavene.

Når det kom til funksjonalitet som vi ville implementere men uten en god forståelse for hvordan så søkte vi mye på Youtube og Unity hjemmesiden etter tutorials. Vi fikk lært oss veldig mye etter å ha sett gjennom en god del videoer hvor det ble forklart trinnvis og konseptet bak hvorfor det ble gjort på den måten.
Det som også var et godt sted var forumet til Unity. Der fant vi så å si alt av løsninger til de samme problemene som andre brukere hadde støtt på.

Noen få endringer ble gjort i løpet av prosjektet med tanke på funksjonalitet på grunn av ikke god nok kunnskap. Men ellers så fikk vi implementert det meste av det vi så for oss i starten av prosjektet. Vi har nå en ganske så god forståelse for Unity spillmotoren med tanke på 2D.

5.3 Erfaring og utbytte

5.3.1 Phuc

Jeg har lært mye om workflowen i Unity og åssen jeg skal takle forskjellige problemer som oppstår. Lært mye om bruken av C# sammen med Unity og åssen få kodene til å henge sammen ved bruk av forskjellige metoder. Fått mye kunnskap om generell spillprogrammering og do's and don'ts. Alt i alt en veldig lærerikt erfaring når man må takle alle problemene selv.

5.3.2 Fedja

Jeg har lært en del mer om bruken av C# i Unity samtidig som jeg har en bedre forståelse av hvordan ting henger sammen i Unity spillmotoren. Prosjektet var veldig utfordrende, dette fikk meg til å bli mye bedre til å søke og tenke meg frem til løsninger ved hjelp av de forskjellige ressursene vi hadde tilgjengelig. Den viktigste erfaringen jeg fikk er å jobbe i en gruppe på et prosjekt av denne størrelsen.

5.3.3 Christer

Jeg lærte mer om digital tegning og tegning generelt. Spesielt nyttig var det å måtte tegne noe med farger istedenfor blyanten eller kullstiftens gråtoner. Jeg lærte også mer om å tegne uten en klar referanse. Prosjektet førte til at jeg lærte meg å bruke nye programmer og interessant teknologi til noe som jeg hadde bare brukt fysiske tegneredskaper til før. Utbyttet var kanskje ikke så stort for meg siden jeg ikke brukte så mye tid som jeg burde ha gjort.

5.4 Veien Videre

Når det kommer til spillutvikling så er det to viktige deler som må tas hensyn til før prosjektet starter. Det ene er å lage spillet, det andre er å selge det. Ideen vår kom fra nostalgi fra tidligere spill fra barndommen vår som i dag er en veldig niche idé.

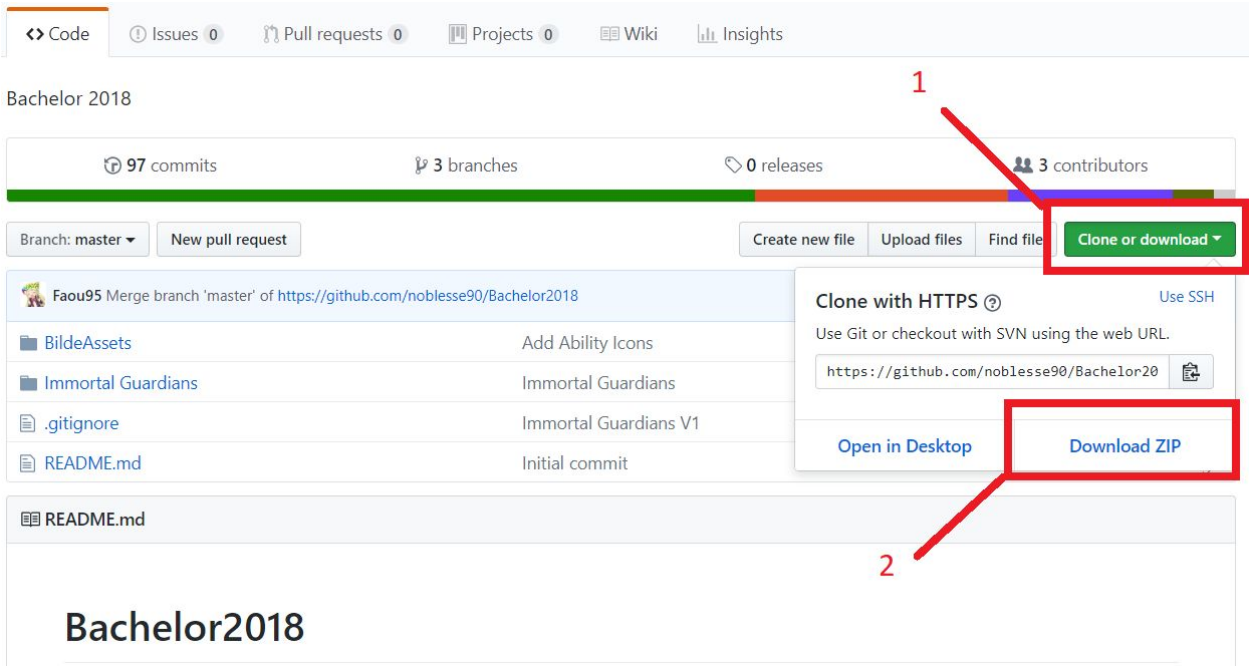
Gruppen vil skilles etter prosjektet men tenkte at utviklingen av spillet kunne fortsettes hvis det ikke var for at vi ikke kan se noen fremtid for dette spillet i dagens spillmarked. Når vi begynte på dette prosjektet så var det et lidenskaps prosjekt med erfaring som mål og ingen ideer om å faktisk selge spillet.

Så vi har bestemt oss for at dette prosjektet er ferdig etter bacheloroppgaven er innlevert, men erfaringen fra dette er noe vi alle kan ta med oss videre. [26]

6. Referanser

Prosjektfil:

Gå inn på lenken og trykk på “Clone or download” og deretter “Download ZIP”.
Lenke: https://github.com/noblesse90/Bachelor2018



Dokumenter

Prosjektskisse:

PDF: Vedlegg.zip/Prosjektskisse.pdf

Prosjektbeskrivelse:

PDF: Vedlegg.zip/Prosjektbeskrivelse.pdf

Milepæl 2:

PDF: Vedlegg.zip/Milepæl 2.pdf

Milepæl 3:

PDF: Vedlegg.zip/Milepæl 3.pdf

Hvordan laste ned og kjøre spillet:
PDF: Hvordan laste ned og kjøre spillet.pdf

Se prosjektlogg på nettsiden vår:
https://immortalguardiantd.wordpress.com/progression/

Unity

[1] Technologies, U. (n.d.). MonoBehaviour. Retrieved from https://docs.unity3d.com/ScriptReference/MonoBehaviour.html

[2] Technologies, U. (n.d.). Unity API. Retrieved from https://docs.unity3d.com/ScriptReference/

[3] UnityScript's long ride off into the sunset – Unity Blog. (2017, August 11). Retrieved from https://blogs.unity3d.com/2017/08/11/unityscripts-long-ride-off-into-the-sunset/

[4] Technologies, U. (n.d.). Intro to 2D World building w/ Tilemap. Retrieved from https://unity3d.com/learn/tutorials/topics/2d-game-creation/intro-2d-world-building-w-tilemap

[5] Technologies, U. (n.d.). Using Cinemachine: Getting Started. Retrieved from https://unity3d.com/learn/tutorials/topics/animation/using-cinemachine-getting-started

[6] Technologies, U. (n.d.). Cinemachine Asset. Retrieved from https://assetstore.unity.com/packages/essentials/cinemachine-79898

[7] Technologies, U. (n.d.). Unity User Manual (2018.1). Retrieved from https://docs.unity3d.com/Manual/

Github

[8] GitHub. (2018, May 04). Retrieved from https://en.wikipedia.org/wiki/GitHub

[9] Hello World - Lets get started with GitHub. (n.d.). Retrieved from https://guides.github.com/activities/hello-world/

Pathfinding

[10] A* Pathfinding Project. (n.d.). Retrieved from https://arongranberg.com/astar/

[11] Granberg, A. (n.d.). Get Started With The A* Pathfinding Project. Retrieved from https://arongranberg.com/astar/docs_3.8/index.php

[12] A* search algorithm. (2018, April 30). Retrieved May 4, 2018, from https://en.wikipedia.org/wiki/A*_search_algorithm

[13] Aron G. (2017, april 27). A* Pathfinding (E01: algorithm explanation) [Video file]. Retrieved from https://www.youtube.com/watch?v=-L-WgKMFuhE

[14] Aron G. (2017, desember 5). Get Started Tutorial - A* Pathfinding Project (4.1+) [Video file]. Retrieved from https://www.youtube.com/watch?v=5QT5Czfe0YE

[15]Aron G. (2017, april 27). 2D Tutorial - A* Pathfinding Project [Video file]. Retrieved from https://www.youtube.com/watch?v=i1Lo_WI_YOQ

Object Pool

[16] Object pool pattern. (2018, May 02). Retrieved from https://en.wikipedia.org/wiki/Object_pool_pattern

Youtube Unity tutorials

[17] inScope Studios. (2016, june 6). Tower Defense [Video file playlist]. Retrieved from https://www.youtube.com/playlist?list=PLX-uZVK_0K_4uNwvKian1bscP9mVvOp1M

[18] Brackeys. (2017, desember 6). SETTINGS MENU in Unity [Video file]. Retrieved from https://www.youtube.com/watch?v=YOaYQrN1oYQ

[19] Brackeys. (2017, november 29). START MENU in Unity [Video file]. Retrieved from https://www.youtube.com/watch?v=zc8ac_qUXQY

[20] Brackeys. (2018, january 3). SCRIPTABLE OBJECTS in Unity [Video file]. Retrieved from https://www.youtube.com/watch?v=aPXvoWVabPY

[21] Brackeys. (2017, desember 20). PAUSE MENU in Unity [Video file]. Retrieved from https://www.youtube.com/watch?v=JivuXdrIHK0

[22] Brackeys. (2017, september 24). How to make a Minimap in Unity [Video file]. Retrieved from https://www.youtube.com/watch?v=28JTTXqMvOU

[23] Brackeys. (2017, august 13). How to use GitHub with Unity [Video file]. Retrieved from https://www.youtube.com/watch?v=qpXxcvS-g3g

[24] Brackeys. (2016, november 27). How to make a Tower Defense Game (E23 HEALTH BARS) - Unity Tutorial [Video file]. Retrieved from https://www.youtube.com/watch?v=UKs1qO8w7qc

Singleton Pattern

[25] Singleton pattern. (2018, May 02). Retrieved from

https://en.wikipedia.org/wiki/Singleton_pattern

Third Party Sprite Assets

Tiles

- LOWLYPOLY. (n.d.). Hand Painted Grass Texture. Retrieved from https://assetstore.unity.com/packages/2d/textures-materials/floors/hand-painted-grass-texture-78552

- A3D. (n.d.). Five Seamless Tileable Ground Textures. Retrieved from https://assetstore.unity.com/packages/2d/textures-materials/floors/five-seamless-tileable-ground-textures-57060

- LOWLYPOLY. (n.d.). Hand Painted Stone Texture. Retrieved from https://assetstore.unity.com/packages/2d/textures-materials/floors/hand-painted-stone-texture-73949

- MAFUBASH. (n.d.). Tileable Pack 01. Retrieved from https://assetstore.unity.com/packages/2d/textures-materials/floors/tileable-pack-01-49278

- 3DFANCY. (n.d.). Stone Floor Texture Tile. Retrieved from https://assetstore.unity.com/packages/2d/textures-materials/roads/stone-floor-texture-tile-18683

GUI

- BLACK HAMMER. (n.d.). Fantasy Wooden GUI : Free. Retrieved from https://assetstore.unity.com/packages/2d/gui/fantasy-wooden-gui-free-103811

- ZERANO. (n.d.). Fantasy UI. Retrieved from https://assetstore.unity.com/packages/2d/gui/fantasy-ui-17019

- MANAKHOV, N. (n.d.). Icons set. Retrieved from https://assetstore.unity.com/packages/2d/gui/icons/icons-set-58217

- INDIE. (n.d.). RPG Fantasy cursors pack IND-UI-3. Retrieved from https://assetstore.unity.com/packages/2d/gui/icons/rpg-fantasy-cursors-pack-ind-ui-3-78578

- Technologies, U. (n.d.). TextMesh Pro. Retrieved from https://assetstore.unity.com/packages/essentials/beta-projects/textmesh-pro-84126

Character

- THOMAS, D. (n.d.). 2D Customizable Character - Male. Retrieved from https://assetstore.unity.com/packages/2d/characters/2d-customizable-character-male-101695

- REXARD. (n.d.). RPG inventory icons. Retrieved from https://assetstore.unity.com/packages/2d/gui/icons/rpg-inventory-icons-56687

- GRANT, L. (n.d.). 20 Evolving Fantasy RPG Weapons. Retrieved from https://assetstore.unity.com/packages/2d/textures-materials/20-evolving-fantasy-rpg-weapons-61204

- BEATHEART CREATIVE STUDIO. (n.d.). Animated 2D trap. Retrieved from https://assetstore.unity.com/packages/2d/textures-materials/animated-2d-trap-7571

Town

- THOMAS, D. (n.d.). 2D Hand Painted - Town Tileset. Retrieved from https://assetstore.unity.com/packages/2d/environments/2d-hand-painted-town-tileset-67346

Kenny.nl's Assets
Kenney • Assets. (n.d.). Retrieved from http://kenney.nl/assets
Disse assets pakkene ble lastet ned:

- Top-down Tanks Redux
- Medieval RTS
- Tower Defense
- Shooter
- Roguelike caves and dungeons
- Roguelike RPG pack
- Background1
- Background2

Andre

[26] Academy of Interactive Arts & Sciences. (2012, June 21). D.I.C.E. Summit 2002 - Mark Cerny. Retrieved from https://www.youtube.com/watch?v=QOAW9ioWAvE

[27] Pair Programming: Does It Really Work? (2018, February 14). Retrieved from https://www.agilealliance.org/glossary/pairing/

[28] Bratsberg, R., Mølstre, H., Wang, M. P., Helmersen, H. T., & Brinckmann, B. (2017, May 24). Utviklingsprosjektet Trolls and Gods [Bachelor project].

7.Kildekode

Du kan se og åpne de forskjellige skriptene ved å følge denne linken her eller ved å laste ned prosjektfilen til Unity via GitHub her. Det er mindre jobb å se koden gjennom GitHub, alt du trenger å gjøre for å se hva som står i de forskjellige skriptene er å åpne filene som har .cs notasjonen.