Prosjektbeskrivelse
Steam Buccaneers: Search for the Seven Cogs

Dangerzone - Programmerere

Bacheloroppgave

Bård Harald Røssehaug | John Olav Haraldstad | Lars Solli Hagen

Innholdsfortegnelse

Introduksjon

Vår problemstilling er: Hvordan utvikle mekaniker som gir spilleren en følelse av å være en pirat i verdensrommet, samt lage et utfordrende univers

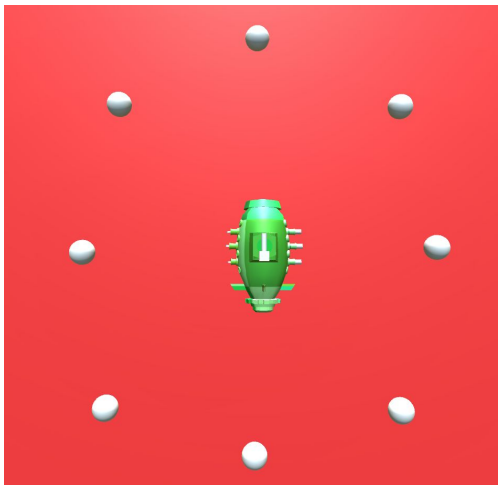
Gameplay

Vi skal i dette avsnittet forklare og argumentere for valg vi tok under utviklingen av gameplay. Dette innebærer da elementene AI, Gravitasjon, spillerkontroll, oppgraderinger og kampsystemet.

AI - MARINE

Første iterasjon

Utviklingen av AI foregikk over flere iterasjoner, av forskjellige årsaker. I starten ble det brukt Unity sitt eget NavMesh Agent system til pathfinding. NavMesh Agent er en komponent du setter på et objekt i scenen, og agenten vil gå mot punktet, eller objektet, du sier er dens mål. I første omgang gikk agenten direkte mot player, men det ble raskt oppdaget at dette gav et veldig annerledes resultat enn vi ønsket, da dette førte til at ai'en kjørte rett inn i playeren ved alle mulige anledninger. En løsning på dette var å danne en ring av spheres rundt player, hvor da agenten ville gå mot disse istedenfor. Disse vil henvises til som "spillerballer".



Ballene er de hvite sirklene som omringer spillerens skip. Agenten ville kjøre mot den ballen som var nærmest ai'ens skip, noe som gav en simulering av at ai'en kjørte etter spilleren mens den svingte og kjørte fremover. Dersom spilleren sto i ro ville agenten kjøre til den ballen som var nærmest skipet, stoppe opp, for så å rotere til sidekanonene pekte mot player.

Dette virket som en god løsning, men problemet vi møtte på nå var at ai'en ikke kjørte som en båt. Unity har ikke laget agentene for at de skal passe på kjøretøy, som forårsaket at skipet til ai'en roterte rundt på en meget urealistisk måte. Vi ønsket at den skulle rotere som et skip i havet, ved å måtte kjøre fremover mens den svinger, eller ved å rotere mens den står i ro.

Løsningen ble å lage et eget objekt for agenten, og et eget objekt for ai-skipet. Det er dette som definerer andre iterasjon av ai'en.

Ai-skipet vil rotere mot agenten dersom den ikke har denne direkte forran seg, og kjører fremover så lenge de ikke kolliderer. Nå kunne vi legge inn regler på hvor raskt skipet kunne rotere seg, som både gav oppførselen vi ønsket, samt ai'en ble balansert mot spilleren ettersom den ikke lenger hadde overlegne navigasjons muligheter.
En ulempe med denne løsningen var derimot at dersom spilleren roterte mens ai'en var nesten helt inntil en ball kunne den rotere i retninger vi ikke kunne kontrollere. Dersom agenten sto inntil en ball, men ai-skipet var akkurat for langt unna til å klare å treffe den, ville det ende i at ai-skipet konstant kjørte rundt i ring.
Dette problemet ble midlertidig løst ved å plassere enn sphere forran skipet til ai'en, og gjøre at agenten kjørte til den spillerballen som var nærmest denne, heller enn den som var nærmest skipet. Dette gjorde også at agenten kontinuerlig ville bytte mellom spillerballene, ettersom det regelmessig ville være en ny som var nærmest detectoren.

Andre iterasjon
Det ble gjort to forsøk på å skrive AI fra bunnen av, og resultatene ble ekstremt ulike. Den første versjonen feilet fullstendig. Før den ble slettet bestod den av over 200 if-tester, hvor da testene var av typen "hvor er spiller i forhold til AI" og "Hva er rotasjonen til spiller i forhold til AI". Her ble det kalkulert den relative x og z posisjonen spilleren hadde i forhold til AI, slik at koden kunne si "dersom spillers relative z posisjon er større enn 0.5 og dens relative x posisjon er større eller lik 5, da skal AI sjekke forskjellen i rotasjonen mellom seg selv og spiller.". Når den så hadde regnet ut forskjellen skulle den ta ulike valg basert på hvor store forskjellene var, og hvilken rotasjonsretning de hadde i forhold til hverandre.
Vi regnet ut hvor mange tester det ville bli dersom vi skulle ha en reaksjon for hver 45grad i en rotasjon, og hver relative x og z posisjon fra -0.5 til 0.5, med en økning på 0.1 ved hver test, og vi kom da frem til at det ville bli mellom 12000 til 16000 tester. Med andre ord ikke en god løsning.

Den andre versjonen er kraftig inspirert av Sid Meier's Pirates. AI'en de brukte i dette spillet var veldig enkel, men den fungerte veldig bra. Reglene den følger er å alltid måtte kjøre fremover, og roterer slik at du har kanonene vendt mot spilleren.
Vi adapterte dette inn i vårt spill og sørget for at vår AI nesten gjør det samme. Vi ønsket at den skulle forsøke å holde seg på en minimumsavstand fra spilleren, men samtidig ikke være for langt unna. Dette ble enkelt løst ved å regne ut avstanden mellom de to objektene. Derretter kastet vi en RayCast

fra høyre og venstre side av AI'en, som da fungerte som sensorene for dens kanoner. Ved å regne ut den relative x posisjonen til spilleren kan AI enkelt se om den er på høyre eller venstre side. Negativ verdi betyr at spilleren er på venstre side, positiv verdi betyr at spilleren er på høyre side. Derretter sjekker AI den relative z posisjonen. Negativ betyr at den har den bak seg, positiv betyr at den har den foran seg. (Legge inn vedlegg)Når den har gjort disse fire testene kan vi si til AI at den enten skal rotere mot venstre, eller høyre.
Det ble også lagt inn at AI'en vil spawne et godt stykke unna spilleren. Dette var for å forhindre at spilleren skulle se den plutselig dukke opp i spillet, og samtidig fjerne behovet for faste spawnpoints. AI'en har en relativt stor makshastighet i forhold til spilleren frem til den kommer innenfor minimumsdistansen den skal ha mellom seg selv og spilleren, da får den samme hastighet som spilleren.

Tredje iterasjon
Marinen fikk nå automatisk tildelt helse og level på våpen basert på avstanden spilleren har til spillets startpunkt. Startpunktet er determinert med en helt vanlig kube som er plassert i scene, og SpawnAI scriptet bruker denne for å regne ut hvor stor avstanden er mellom spiller og startposisjonen. Denne har blitt tildelt navnet "origin".

Kanonenes level blir satt i "setSCannonLevel" funksjonen i SpawnAI. Variabelen "temp" holder på verdien tilsvarende 10% av avstanden mellom objektene, så er avstanden 100 vil temp være 10. Ved å bruke 10% har marinen en balansert endring på antall oppgraderte våpen gjennom hele nivået til spillet.
Det kjøres så en loop seks ganger, tilsvarende alle de seks kanonene på skipet. Hver gang opprettes et tilfeldig tall mellom 0 til 100 som brukes i testen for å se om en kannon skal bli level 2. Dersom dette tallet er mindre enn "temp" vil kanonen bli level 2. Det opprettes så et nytt tall mellom 0 til 100, og det sjekkes om dette er mindre enn 50% av verdien til temp igjen, med andre ord dobbelt så liten sjanse for å få høyere tall. Er tallet høyere blir kanonen level 3.
Etter denne loopen sjekkes avstanden mellom spiller og origin. Her skjer det bestemte tester som sjekker om marinen har fått oppgradert nok kanoner til et minimums level i forhold til avstanden mellom spiller og origin. For eksempel skal det være minst 2 level 2 kanoner når avstanden er over 200meter. Stemmer ikke dette settes en tilfeldig kanon til level 2, slik at kravet er nådd.

Helsen til marinen tilsvarer 1% av avstanden mellom spiller og origin. Dersom avstanden er 100m vil da helsen til marinen være 1. Det er satt inn en if test rett under som sjekker om helsen til marinen er under 20, og dersom dette stemmer settes helsen til 20. Det gjør marinen til en enkel, men ikke for

enkel, fiende de første minuttene i spillet slik at spilleren kan bli kjent og komfortabel med kontrollene.

Fjerde iterasjon

Den fjerde versjonen av Marinens AI inkluderte en rekke spennende endringer. Nå spawnes det opp til 10 mariner som kjører rundt på kartet samtidig. Når en marine opprettes vil den samtidig få et punkt i verden den skal kjøre mot. Hvilket punkt er ikke så nøye, ei heller om den kjører mot eller bort fra spilleren, da det viktigste var å befolke universet på en måte som lar spilleren se andre skip når kameraet er zoomet ut.

Når en av marinene kommer nær nok spilleren vil alle andre enn den ene marinen slettes fra scenen. Marinen som lever vil så gå inn i en kamp-modus og begynne sin jakt på å drepe spilleren. Når marinen dør vil SpawnAI scriptet begynne å befolke universet igjen, og hele syklusen restartes. Den originale ideen var å ha flere mariner i kamp mot spilleren samtidig, men det ble for vanskelig å sørge for at marinene ungikk hverandre i tillegg til å angripe spilleren på en effektiv måte.

For å forhindre dødtid ble det opprettet en timer som tikker oppover med 1 hvert sekund, med andre ord en vanlig stoppeklokke. Når denne når en bestemt verdi, i vårt tilfell 45 sekunder, vil den sist opprettede marinen gå i kampmodus og angripe spilleren. Da vil de andre marinene slettes, slik som ellers når en marine går til angrep, og timeren vil restartes. Timeren restartes hver gang en marine går til kamp eller dersom bossen lever for å forhindre å starte en unødvendig kamp.

AI - BOSS

Bossen i spillet ble utviklet raskt og effektivt ved å bruke koden som allerede var utviklet til marinens AI. Siden marine og boss deler script måtte vi innføre noen få tester i kodene for å forsikre oss om at bestemte ting skjer dersom det er bossen som lever, men ellers undergikk de få endringer.

I SpawnAI sjekkes det om spilleren er nærmere enn X distance fra punktet hvor bossen skal spawne, og dersom det stemmer skal boss spawne. Om ikke skal det spawne en normal marine. I AIMaster er det så en bool som heter “isBoss”, og denne er bare sann dersom det er bossen som spawnes. Denne brukes for å forhindre at bossen skal slette alle andre skip fra scene, da vi ønsket at spilleren skal kunne se at skipene flykter når han/hun nærmer seg bossens spawn punkt.

Bossen i spillet legger ut bomber bak seg hvert 5. sekund, og disse sprenger dersom et skip eller en kanonkule treffer dem. Når de sprenger opprettes en array av colliders som sjekker om det er noen andre collidere innenfor en angitt radius, og disse vil legges til i arrayen. Derretter kjøres det en loop gjennom arrayen, hvor det finnes alle RigidBody components. Disse får så påført seg en kraft, og distansen til bomben determinerer kraften. Lenger avstand tilsvarer lavere kraft.

Sammen med dette vil alle AIMove script pauses i 1 sekund. Dette er for å gi spilleren en følelse av å miste kontroll etter å ha blitt skadet av en bombe, men det skjer også med alle ai'er (marine og boss) som påvirkes av bomben. Dette ble gjort både fordi det er logisk at også bossen skal påvirkes, samt det ble ubalansert at bossen kunne kjøre videre mens spilleren var et hjelpeløst mål.

Gravitasjon

Når vi planla og ha ekte gravitasjon med i spillet regnet vi med at dette ville bli en ganske vanskelig ting å lage og å få til å fungere helt riktig. Derfor ble det planlagt at det ville bli brukt 2 uker for å ferdigstille denne koden. Dette viste seg å være galt. Etter litt søking på nettet fant vi et script som laget gravitasjon på tingen vi festet scriptet til. Ikke bare åpnet dette opp med to uker med ekstra tid, men gjorde at utviklingen av AI og spillerkontroller kunne bedre tilpasses til gravitasjonen.

3 faglige og akademiske kilder (ikke APA stil enda):

Project Leadership - Step by step (Svein Arne Jessen 2010)

The Art of Game Design - A book of lenses. second edition (Jesse Schell 2015)

The hows and whys of level design. second edition (sjoerd "Hourences" de jong 2008)

Player controller

Noe av det viktigste, om ikke det viktigste i et spill, er det å få selve karakteren du styrer til å føles bra ut, se bra ut, i hvert fall i et 3-persons synspunkt, og fungere feilfritt.

Player controllen tok utgangspunktet i den tidlige prototypen for et control-scheme utviklet høsten 2015, et simpelt script hvor spilleren bare ble flyttet fremover ut i fra dens lokale x-retning, og roterte rundt sin egen z-akse. Prototypen hadde også en kanon på taket, som alltid hadde sin rotasjon vendt mot musepekeren, og stasjonære kanoner på begge sider som kun skyter rett frem fra skipets høyre eller venstre side.

Level Design

I boken “The hows and whys of level design” av Sjoerd De Jong, har han komponert en sjekkliste (s.26) over nøkkelelementene som et level vil bli bygget rundt. Nedenfor er det prøvd å fylle ut denne listen med den informasjonen nødvendig i planleggingen. Listen fungerte godt som et verktøy for å bygge levelet, for det ga god oversikt over hva som trengs, og planleggingen.

(1- time)
For selve planleggingen av levelet har jeg totalt 7 dager bestående av 8 timers arbeidsdager

(2-tools)
Unity er game enginen som vil bli brukt til å lage designet, videre så vil en designer lage teksturer og modeller i Photoshop og Maya.

(3-Limitations)
sånn sett så har vi ingen klare limitations annet enn tid. En liten hickup er at kunstneren som skal lage tekstur til planetene aldri har gjort det før, og jeg har ikke mye erfaring med å bygge en level før annet enn en som er basert på et virkelig kart av Galapagos. Jeg har også designet brikker som var brukt i et brettspill laget tidligere i utdanningen sammen med Bård og John.

(4-requirements)
Det vil være butikker rundt i systemet som må tas i betraktning når vi designer levelet, så levelet må designes slik at planetene rundt lett kan skape en viss følelse av gjennkjennelse så du vet hvor du er, og hvor de andre butikkene er., i tillegg så skal vi ha ulike soner hvor fiender med ulik styrke skal opprettes, vi skal også ha en sone hvor det er plassert en boss.

(5-purpose)
Det er et singleplayer spill med random spawns og spesifikke butikker rundt om kring i solsystemet. Det er også plassering for en boss. Man skal også ha en viss følelse av oppdagelse i spillet.

(6-gameplay)
Man styrer et romskip rundt, og har muligheten for tilfeldige encounters, i tillegg til at man skal sloss mot en boss som end game, så det må være nok plass til å kunne bevege seg rundt. I tillegg så er også fysikk ganske viktig, i og med at vi har med at planeter har gravitasjonskraft, og trekker både spillere og AI, samt kulene deres mot seg om man kommer nære nok. “inertia” i rommet er også en viktig faktor, at man “sklir” når man kjører rundt, så det å ha nok plass til å skli rundt, men fortsatt ha god nok plass til å ha en responstid er viktig. I tillegg så vil det kanskje være store distanser mellom objekter, så man kjører fort. Vi har også lekt med ideen om å ha at planetene går i baner, veldig sakte, så levelet vil konstant endre seg.

(7-Theme)
Sci-fi/space tema som er cartoonish, så universet må på mange måter reflektere dette.

“One designer may write everything down i a design document wile another (...) plans it out in their head” s. 25

Jeg er enig med det sistnevnte, hvor jeg heller i stor grad planlegger hvordan det skal se ut i hodet, i hovedsak utseende på hvordan det skal se ut. Dette skaper store problemer ettersom det er jeg som designer verden rundt gameplay og story, og ikke de kunstneriske aspektene. Det gikk derfor opp for meg hvor viktig det var å så fort som mulig finne ulike bilder, eller formidlet ideene mine på papir, til de som faktisk tok seg av det kunsteriske, som å teksturere og modellere planetene, og bakgrunnene som ville være på plan i de ulike sone-inndelingene, slik at alle har en pekepinn på hvordan levelet skal se ut.

Så fort jeg hadde fått visjonen min på plass i hodet, fikk jeg den ned på papir hvor jeg laget flere forskjellige oppsett av hvordan verden skulle se ut, hvor ting skulle stå plassert, ulike soner og videre. Deretter gikk jeg videre til å finne bilder av hvordan jeg ser for meg at de ulike planetene og bakgrunnene ser ut, og passer sammen, et såkalt moodboard. For å sitere Sunniva, som sitter med hovedansvaret for å faktisk lage planetene :
“dette er veldig hjelpsomt, når jeg skal lage planeter :D”

Noe av det store ved designet av dette levelet, er å få det til å virke som et dynamisk solsystem som faktisk lever. Vi ønsket at planetene skulle rotere rundt sola i systemet, og at spilleren også kunne reise rundt hele solen for å oppdage innholdet i spillet. I praksis var dette en stor utfordring i forhold til flyten i levelet, samt det ville bli for mye tomrom i levelet i denne omgang, uten nok innhold, og for store distanser å reise.

(se LevelDesign4 eller 5)

Løsningen ble heller i denne omgang å ha levelet begrenset til et lite område, hvor planetene fortsatt vil rotere rundt solen, men ikke hele veien rundt. Her vil heller planetene havne i toppen av grensen, utenfor synsrekkevidde, og rotere ned mot den nedre grensen, når de er ute av synsrekkevidde her, vil de bli satt på toppen igjen, og begynne å rotere nedover.

Implementering av GUI

Før GUI kunne implementeres fant vi ut at det ville være lurt og ha en måte og lagre data på. Vi fant ut at å bruke “Language Specific Serialization” til å lagre data, fungerte godt i vårt tilfelle. Vi lærte dette via Unity sine tutorials.
Vi utviklet også en måte å gå fra spillscene til shopscene og fortsatt holde på den viktigste spiller dataen. For å gjøre GUI enklest mulig å jobbe med, bruker vi bare minst mulig scripts for knappene. Dette gjorde vi fordi det ble alt for mange script, hvis vi skulle hatt et unikt for hver knapp. Måten vi gjorde dette på var at vi koblet scriptet til et empty gameobject og brukte Unity

sin innebygde knappefunksjon. Vi refererte til det ene scriptet på alle knappene, og det fungerte akkurat som vi ville.

Quest GUI

Tutorial

Planleggingen av tutorial var gjort av vår level-designer Lars, vår animatør Synnøve og kodeansvarlig John Olav. Grunnen til at det var spessielt disse tre som planla tutorial var fordi hver enkelt hadde en del som skulle inn. Lars måtte se til at tutorialet ville være bra gameplaymessig og at det ville være interessant for spilleren å spille. Synnøve har med alt av dialog å gjøre og måtte se til at karakterene i spillet ville bli godt presentert. John Olav var med for å se til at tingene Lars planla kunne gjøres i koden for så å implementere.
Planleggingen gikk fint og en oppbygging ble skrevet.

Start

- *Plot: Protagonist buys ship. Gets instructions from shopkeeper.*
    - *Dialog-system. Talk*
- *Player learns controls*
    - *First stage: steering.*
        - *Dialog tells player how to control*
        - *Next stage trigger by player pressing W, A, and D. Shooting turned of.*
    - *Second stage: shooting*
        - *Dialog tells player how to shoot*
        - *Next stage trigger after player shooting 10 all together.*
    - *Last stage: Enemy*
        - *Enemy spawns. Rest of game pauses.*
        - *Dialog tells player what to do and what happens.*
        - *Everything unfreezes and player uses controls to kill enemy.*
        - *Next stage trigger when enemy is killed and loot is picked up.*
- *Player learns shop*
    - *First stage*
        - *Dialog: Explaination on what shop does and how to enter*
        - *Player drives into shop and change scene*
    - *Last stage*
        - *Dialog: Meet shopkeeper and explaination on how to use store.*
        - *Player fixes ship for damage*
        - *Player can leave shop and exit tutorial*
        - *Player spawnes in real game world*

Vi jobbet videre med dette og noen små forandringer var gjort. Som for eksempel at tutorialet fortsatte når spilleren ønsket, istedenfor at den fortsatt etter at player hadde trykket på knappene

som styrte. Dette gjorde vi fordi vi så at alle trenger ulik tid på å venne seg til kontrollerne. Da er det greit å kunne utføre tutorial i sitt eget tempo.
Etter å ha fått dialogen til karakterene måtte dette legges inn i koden på en god måte. Her brukte vi en array som holdt alt av dialog, så hver gang spilleren progresserer, så vil koden fortsette igjennom koden. Ikke en veldig dynamisk eller avansert metode, men det fungerte fint for det lille tutorialet vi har. Alle andre deler av tutorialet som vi trengte hadde vi laget fra før. For eksempel spawning av Marine, pause funksjon og loot-drop. Tutorialet var mest å utføre allerede laget funksjoner i riktig rekkefølge enn å skrive nye funksjoner.

Iterasjoner av planlegging (Gantt)
Våre planer for hva vi skal gjøre, og når, har undergått endringer kontinuerlig gjennom arbeidsprosessen dette semesteret. I denne delen skal jeg forklare de ulike versjonene vi har hatt, og hvorfor vi har foretatt de ulike endringene.

Vår gantt er fargetkodet, som da betyr at når en rute har for eksempel fargen gul, da skal John Olav utføre den oppgaven den dagen. I radene nedover kolonne A ser vi oppgavene som skal utføres. Ved siden av i kolonne er rutene ved siden av hver oppgave farget, som da forklarer hvem som har hovedansvar for hver enkelt oppgave.
For hver dag som går skal hvert medlem skrive ned antall timer de jobbet den dagen. Dette er for at alle medlemmene i gruppen skal kunne se hvordan de andre ligger an, og om de gjør arbeidet som forventes av dem. Det er selvfølgelig mulig at noen jukser på timene de har jobbet, men det er likevel en bedre ordning enn ingen notering av timer i det hele tatt.
Raden i toppen av gantt viser datoer. 08.01 betyr da 8. januar. Dette er for at vi enkelt skal kunne se hvilken dato de ulike oppgavene skal utføres. Har en dato rosa ruter under seg er det obligatorisk oppmøte, da vi skal slå sammen prosjektet vårt i Git og se gjennom gantt for mulige endringer som må utføres. Grå ruter er ment for å skrive på rapporten, og svarte er deadlines for innlevering til veiledere.

I den første versjonen av gantt (se vedlegg 1.1) hadde vi kartlagt alt som måtte utføres av arbeid, hvem som skulle gjøre det og når. Det er også inkludert at medlemmene skal skrive inn X dersom de ikke har jobbet med dagens oppgave, F dersom de har jobbet med og fullført hele oppgaven og A+rad dersom de har jobbet med noe annet enn oppgaven de er satt opp til. Dette var ment å skulle brukes dersom en oppgave ble ferdig før tiden, for eksempel gravitasjon. Ble gravitasjon ferdig noen dager før tiden, skulle John bruke A+rad, for eksempel A30, for å henvise til oppgaven han jobbet med istedenfor.

Det er et synlig hopp i John Olav sine arbeidsoppgaver. Dette skyldes dårlig fordeling av arbeidsoppgaver. Vi valgte i første omgang å ikke fylle ut dette tomfeltet, da vi valgte å tro at han kunne fungere som en "support" de to ukene, altså veksle mellom å assistere Bård og Lars i deres oppgaver.
Vi ønsket ikke å gi ham flere arbeidsoppgaver, da dette ville føre til at han måtte ta noen av de andre programmerernes oppgaver, og da ville de stå med tomrom i arbeidsplanen. Vi antok også at vi ville støte på problemer underveis som førte til at tomrommet ikke ville bli noe problem, og mer eller mindre fikse seg selv.

Vi valgte å ikke notere ned helgene i gantt i 1. iterasjon, da foreleserne gav klare signaler om at de forventet at alle studenter jobbet hver dag. De forventet samtidig 37.5 timer i uken, noe som betydde 6 timer med arbeid hver dag. Vi kom frem til at et medlem kan jobbe så mye eller lite de vil, så lenge de blir ferdig med oppgaven i tide. Dette betydde at dersom noen ble ferdig 2 dager før tiden kunne de ta seg 2 dager fri.

Det tok ikke lang tid før vi følte behovet for å implementere helgene inn i gantt. I andre iterasjon av gantt (se vedlegg 1.2) har det nå kommet grønne kolonner, som da betyr helg, altså lørdag og søndag. Vi forventet fra da av at medlemmene jobbet 8 timer om dagen, eller 37.5 timer i uken. Ved å innføre helgene og gjøre pilotproduksjonen om til en jobb føler man seg mindre presset til å måtte jobbe i helgene også, noe som gjør at medlemmene kan hvile seg ut i to dager for så å prestere bedre de neste 5. Jobber derimot et medlem mye mindre enn forventet og ikke klarer å fullføre oppgaven i tide, er det forventet at medlemmet må jobbe inn tapt tid i helgen. Denne løsningen har vist seg å være veldig god for oss, og vi har holdt på dette systemet ut semesteret.

Medlemmer kan fremdeles jobbe for eksempel 10 timer om dagen, fire dager i uka, for så å ta en ekstra fridag. Vi forventer bare at de skal ha jobbet minimumskravet og fullfører oppgavene sine.

Det er også innført blå ruter. Blå ruter betyr obligatorisk fremføring forran klassen. Det er bare to av disse, men det er likevel greit å få dem skrevet ned slik at ser dem i god tid før presentasjonen.

Det ble heller ikke gjort noen forsøk på å reparere gantt for andre iterasjon, men det hadde allerede oppstått problemer. John fullførte gravitasjonen og lagre/loade mye tidligere enn antatt, så det ble nødvendig å gi ham nye oppgaver å jobbe med. Løsningen ble å sette ham på GUI, noe

som har fungert veldig bra. Han fikk arbeid å utføre, og Bård, som vi kan se i vedlegg 1.3, fikk 2 ekstra på å forbedre AI'en.

Til tredje iterasjon av gantt (se vedlegg 1.3) ryddet vi opp i arbeidsfordelingen, arbeidstider og endret til å vise hvordan arbeidet faktisk har foregått. Hovedsakelig betyr dette at vi har fått ryddet opp i rotet med John sine oppgaver. Som vi kan se har vi nå redusert gravitasjon til bare 1 dag, ettersom det var så raskt det gikk. Det er også lagt til hvilke dager han jobbet med hva frem til 27. januar, datoen vi endret til tredje versjonen av gantt.
Nå viser det bedre hva han skal gjøre, og når, samt vi har fått laget en jevn arbeidsfordeling frem til alle oppgaver er utført. Lars har fått noen ekstra dager på leveldesign, da han så dette som en nødvendighet, og det er lagt til hvor lenge ekstra Bård skal jobbe med AI.

Som vi kan se er det noen trange uker hvor det bare blir en eller to dager til å arbeide på spillet, men vi har valgt å ikke endre på dette da det fremdeles vil fungere som en vanlig arbeidsuke. Ja, det blir bare 2 dager til å jobbe med spillet i slutten av januar, men til gjengjeld er det 3 dager som går bort til forberedelse for innlevering. Før den første fremføringen er det nødvendig å sette sammen det vi har og forberede seg til presentasjonen, så derfor blir uken seende ut som den gjør. Hadde vi endret på ukene for å alltid få 5 sammenhengende dager med arbeid på prosjektet ville det overstyrt helgene og dagene vi uoffisielt har fri, noe som igjen ville ført til trettere medlemmer og lavere effektivitet.

Fjerde iterasjon
Som vi kan se i Vedlegg 1.4 ble det lagt til noen få nye punkter og ryddet opp i timefordelingene. Noen medlemmer lå foran planene, noen litt bak, så da endret vi gantt til å vise de nye realistiske arbeidstidene som krevde for å fullfør en oppgave.
Oppgavene som ble lagt til var "Marine AI - unngå shop, boss, planeter" og "Marine AI - spawne på patruljepunkt, patruljere, oppdage player". Grunnet endringer i planene for hvordan vi ville ha ai'en i spillet til å oppføre seg ble det nødvendig å innføre disse to endringene til gantt.
Oppgavene ble skrevet inn like etter bossen til spillet ble fullført da det ikke var mer, i forhold til planene, for Bård å gjøre.

Arbeidsmåte

Det var viktig for gruppen at medlemmene sa seg enige i bestemte arbeidsmåter, da dette var med på å skape tillit og organisering. Organisert, som i at alle vet hva som forventes

av hverandre, og hver enkelt medlem vet hva som forventes av dem, og tillit kommer av at alle gjør nettopp det organiseringen er ment for.

GitHub

Gjennom hele utviklingen av spillet har vi tatt i bruk GitHub. Dette lot oss på en forholdsvis enkel og effektiv måte samle arbeidene våres i et felles prosjekt, samt se på koden til de andre medlemmene for å finne inspirasjon eller hjelp. GitHub ble også brukt sammen med designergruppen, da dette lot oss få alle elementene ferdig implementert i Unity.

Det var under de ukentlige møtene at vi merget sammen de ulike branchene til medlemmen inn i Master. Vi valgte å gjøre det i felleskap, ettersom det var relativt stor fare for konflikter. Ved de aller fleste anledningene kunne vi enkelt finne ut årsaken til en konflikt, men det skjedde også at det oppsto fatale feil som førte til at vi kunne sitte i flere timer og krangle med GitHub for å komme rundt problemet. Vi har ved flere anledninger måtte lage nye branches og slette de gamle, da feilen ikke så ut til å kunne repareres. Når vi slettet en branch kopierte vi først hele prosjektmappen den innehold, og kopierte så dette inn i den nye branchen. Vi vet fremdeles ikke hvorfor dette fikset problemet, men vi har ikke undersøkt det nærmere.

37.5 timer i uken

Pilotprosjektet blir behandlet som en fast jobb, og det forventes da at hvert medlem jobber ca 38 timer i uken. Som nevnt i “Iterasjoner av Gantt” ovenfor behandler vi produksjonen som en fulltidsjobb, med da 5 dager jobbing og 2 dager fri i uken. De fleste medlemmene jobber med andre ord 8 timer om dagen, 5 dager i uken.

Unntak kan selvfølgelig forekomme. Noen ganger jobber man kanskje med noe som var forventet å ta veldig lang tid å fullføre, men plutselig kommer man på en kode som løste problemet på en mye enklere og mer effektiv måte enn planlagt. Har man da jobbet 6 timer den dagen går man gjerne hjem heller enn å begynne på noe nytt med de resterende 2 timene.

Et eksempel kan være AI. Det siste som gjenstår er å sørge for at AI spawner i nærheten av spilleren, og at den så går direkte mot spilleren. Når denne koden er ferdig, er AI antatt å være ferdig. Det er torsdag, og Bård skal være ferdig i løpet av fredagen. Han har jobbet 6 timer den torsdagen, og har så fullført hele oppgaven. Det er da akseptabelt at Bård drar hjem istedenfor å starte på mandagens oppgave, dersom han ønsker dette. Når det så blir fredag skal Bård starte på mandagens oppgave.

Samlet

Hver uke var det en kamp om å sikre seg et grupperom for hver dag fra mandag til fredag. De aller fleste medlemmene ønsket å møte opp på skolen for å jobbe i felleskap, av flere årsaker. Den største var at man lettere klarte å holde seg konsentrert om man satt sammen med andre som jobbet. Det er også morsommere å sitte sammen med dem man lager spill med, enn å sitte hjemme alene. Ved å sitte sammen er det også lettere å samarbeide, som er til stor fordel for oss programmerere. Kanskje har en av oss løst en liknende problemstilling en annen befinner seg i, og kan da assistere denne.

Ukentlige møter

Hver fredag var det obligatorisk oppmøte for alle medlemmer i gruppen. Under fredagens møter gikk vi gjennom hva hvert enkelt medlem har gjort siden sist, hvordan de ligger an i forhold til planen (Gantt) og hvilke eventuelle endringer i Gantt som må foretas. Vi merget også sammen prosjektet i Git på fredagene.
Møtene var viktige for gruppen. De sørget for at medlemmene gjorde det som var forventet av dem til angitt tid, var en mulighet for alle å møtes og være sosiale, planlegge fremover og gi en følelse av kontroll.

Spilltestene

Under spilltestene møttes vi på skolen for å møte de andre gruppene i et samarbeid om å teste hverandres spill. Her deltok medlemmene fra de ulike gruppene, og gav hverandre konstruktiv tilbakemelding på spill, ide, og konsept. Vår gruppe noterte ned det meste som ble sagt, og utførte nødvendige balanseringer og bugfixing etterpå. Designvalg ble så utført i fellesskap med gruppen basert på tilbakemeldingene vi fikk under spilltestene.

Se “Vedlegg 2” mappen for referat fra de ulike spilltestene.