BACHELOROPPGAVE

Hypercube - multiplayer spill med Unreal Engine og dedikert server (Amazon Web Services Gamelift)

Marius Irgens

Dataingeniør
Data- og informasjonsteknologi kull 2018
Veiledere: Sven-Olai Høyland (HVL) og Aleksander Pedersen (UiT)
27. mai 2021

Rapportens tittel: Hypercube - multiplayer spill med Unreal Engine og dedikert server (Amazon Web Services Gamelift)	***Dato:*** 27. mai 2021
Forfatter: Marius Irgens	***Antall sider u/vedlegg:*** 50
	Antall sider vedlegg: -
Studieretning: Dataingeniør	***Antall disketter/CD-er:*** -
Kontaktperson ved studieretning: Sven-Olai Høyland	***Gradering:***
Merknader:	

Oppdragsgiver: Egendefinert oppgave	***Oppdragsgivers referanse:*** -
Oppdragsgivers kontaktperson: -	***Telefon:*** 98622671

Sammendrag:

Hypercube er et internettbasert flerspillerspill som låner designkonsepter fra esport og Battle Royale sjangeren. I Hypercube spiller man som en svevende droid. Åtte spillere plasseres på en kube bygget opp av 2400 ruter. Når en spiller beveger seg over en rute får hen energi til taktiske ferdigheter, men ruten går også i oppløsning. Dette fører til at spillerene får bedre og bedre evner, men samtidig til at området man kan bevege seg på blir mindre og mindre. Det er ikke mulig å stå stille. Det er mulig å bevege seg på alle sidene av kuben, som har gravitasjon inn mot sentrum. Hvis man faller inn i midten er man ute av spillet, og siste stående droide vinner.

Spillet er laget i Unreal Engine og bruker Amazon Web Services Gamelift for replikering mellom klienter.

Det ligger en video på utviklerens Youtube kanal: *https://youtu.be/oEGysBWCUsY*

Stikkord:

Spill	Dedikert server (AWS)	Multiplayer

Høgskulen på Vestlandet, Fakultet for ingeniør- og natuvitskap
Postadresse: Postboks 7030, 5020 BERGEN Besøksadresse: Inndalsveien 28, Bergen
Tlf. 55 58 75 00 Fax 55 58 77 90 E-post: post@hvl.no Hjemmeside: http://www.hvl.no

Innholdsfortegnelse

Det ligger en video på prosjektets Youtube kanal: *https://youtu.be/oEGysBWCUsY*
det er fint om du ser den før du leser videre.

Innledning

Motivasjon og mål

"Den globale spillindustrien er en milliardindustri som har opplevd sterk vekst de siste årene. Den norske spillindustrien har også vokst, men veksten har vært beskjeden sammenlignet med både det globale markedet og med våre naboland" *(Oslo Economics 2018 s.5)*.
Norge er et kulturrikt land med ellers sterke bidrag til den globale felleskulturarven. Dataspill har gått historisk fra å være en sjenert kuriositet til prominent kultivator i det globale åndslivet. Som nasjon har vi mye å gi, også her; verden trenger flere norske aktører i spillmarkedet.

"If you are working on something that you really care about, you don't have to be pushed. The vision pulls you." – Steve Jobs.

Enter studio er et oppstartsselskap som brenner for å fremme norsk deltagelse i det internasjonale spillbildet, såvel som å heve dataspillets status i det norske åndslivet. Studioet håper på å kunne opparbeide seg interaktiv ekspertise gjennom gjentatte prosjekter. Denne ekspertisen vil deretter være tilgjengelig i det norske samfunnet og alltid kunne bidra til å heve den norske standarden i sektoren. Studioet håper at dette prosjektet kan være starten på en slik reise i gjentatte prosjekter. De tekniske hovedtemaene i denne oppgaven (interaktiv grafikk og nettverksreplikasjon) vil dessuten studeres videre i en eventuell masteroppgave.

Kontekst

"Work consists of whatever a body is obliged to do, and… Play consists of whatever a body is not obliged to do." - Mark Twain, The adventures of Tom Sawyer.

Mykvarer kan ansees å være digitale verktøy som hjelper brukeren å utføre et stykke arbeid, som tidligere har vært enten ekstremt tidkrevende, eller rett og slett umulig. Dataspill skiller seg fra denne beskrivelsen ved at dets mål ikke er å hjelpe brukeren å arbeide. Nærmere det motsatte: Å hjelpe brukeren å avkoble. Dette er grunnlaget for å kategorisere dataspill som kultur, selv om kultur ikke alltid har en avkoblende faktor. Denne kulturelle sammenkoblingen gir spill en såkalt x-faktor: et kunstnerisk aspekt som er vanskelig å definere. Hva er gøy? Hva er kult?

Man kan se på det generelle markedet for å prøve å danne seg et bilde av hva brukeren ser etter i et dataspill. Her må man selvsagt ha realistiske forventninger ut fra hvilke tekniske begrensninger selskapet besitter - hvis man er et nystartet enkeltmannsselskap med en tidsramme på 4 måneder kan man ikke ha de samme forventningene til sluttresultatet som et veletablert konsern som jobber med et prosjekt i årevis med mange ansatte; man må prøve

å ekstrahere essensen av spillene på markedet, og deretter se på hvordan man kan tilsette disse essensene i sitt eget prosjekt.

Det finnes noen fellesnevnere i dagens spillnæring som er værdt å merke seg:

- Flerspillerfunksjonalitet (over internett), såkalt social gaming er en viktig faktor i dagens spillmarked: “According to ESA data, 50 percent of gamers consider online play capability a factor into their purchasing decision.” *(Broadbandsearch 2020)*.
- Esport (kompetativ spilling) er populært: “Esports is making a notable mark on the industry, with NewZoo noting that in 2017, 22 percent of US male miennials watched esports.” *(Broadbandsearch 2020).*
- Battle Royale sjangeren er en av de største og viktigste spillsjangrene for øyeblikket: PUBG, Fortnite, Apex legends, COD:Warzone og Hyperscape er noen få av de gigantiske aktørene i markedet.

	Game title	Publisher	Change
1.	Minecraft	Mojang	-
2.	League of Legends	Riot Games	-
3.	Counter-Strike: Global Offensive	Valve Corporation	-
4.	Call of Duty: Modern Warfare/Warzone	Activision	-
5.	Valorant	Riot Games	1 ▲
6.	Grand Theft Auto V	Rockstar Games	1 ▼
7.	Fortnite	Epic Games	-
8.	Apex Legends	Electronic Arts	1 ▲
9.	ROBLOX	Roblox Corporation	1 ▲
10.	Rocket League	Psyonix	2 ▼
11.	Call of Duty: Black Ops Cold War	Activision	1 ▲
12.	Tom Clancy's Rainbow Six: Siege	Ubisoft	4 ▲
13.	Overwatch	Blizzard Entertainment	-
14.	World of Warcraft	Blizzard Entertainment	3 ▼

Figur 1.1: Toppselgere for mars 2021 (NewZoo 2021). Samtlige spill har flerspillerfunksjonalitet, de fleste av Battle Royale varianten.

Spillet skal inkorporere disse egenskapene:

- Flerspillerfunksjonalitet via internett.
- Viktige elementer fra esport.
- Viktige elementer fra Battle Royale sjangeren.

Begrensninger

Enter studio består av en enkelt ildsjel, Marius Irgens, og spillet må derfor bygges under begrensede forutsetninger. Det er svært vanskelig å lage et fullstendig spill på gitt tidsrom. Produktet vil derfor fungere som en prototype, som kan brukes for å oppnå midler til å fullføres.

Hver eneste bit av spillet skal lages internt. Det skal ikke brukes noen eksterne ressurser utover spillmotoren, servertjenesten og 3D-modelleringsmykvaren. Dette vil øke omfanget på produksjonen ytterligere, men samtidig øke omfanget av den opparbeidete ekspertisen. Spill som utelukker eksterne ressurser blir også mer annerkjent av spillere.

Ressurser

- *Unreal Engine:* en av verdens ledene spillmotorer, som er spesielt godt egnet for å lage nettverksbaserte multiplayer spill. Motoren driftes av Epic Games, som blandt annet har laget Fortnite, Unreal Tournament og Gears of War.
  https://www.unrealengine.com

- *Amazon Web Services (AWS) Gamelift:* en servertjeneste spesielt utviklet for dataspill. Amazon er verdens største serverleverandør.
  https://aws.amazon.com

- *Blender:* et open source 3D-modelleringsprogram. Blender er helt gratis, i forhold til konkurrerende produkter som gjerne koster rundt 2000 kroner i måneden. Funksjonaliteten i Blender er ikke bare like god, men i noen tilfeller bedre enn konkurrerende produkter, og blir brukt i spillindustrien i økende grad.
  https://www.blender.org/

- *Visual Studio:* Unreal Engine fungerer i tandem med Visual Studio, slik at man kan skrive all koden her mens man bruker funksjonalitet fra spillmotoren. Unreal Engine har også en editor som gjør det lett å visualisere, bygge og teste funksjonalitet.
  https://visualstudio.microsoft.com/

- *Spilltestere:* Det er nærmest umulig å lage et spill uten brukere som kan teste det underveis. Tilbakemeldinger om spillfunksjonaliteten er helt vital for prosjektet, og bør komme så tidlig og ofte som mulig.

Prosjektbeskrivelse

Praktisk bakgrunn

“I fear not the man who has practiced 10,000 kicks once, but I fear the man who has practiced one kick 10,000 times.” - Bruce Lee.

God kompetanse i spillindustrien kan ikke bygges over et enkelt prosjekt. Det kreves gjentatte forsøk over lengre tid for å utvikle ekspertise og spisskompetanse, gjerne med fokus på èn spesifikk sjanger. Flerspillersjangeren er den med størst personlig affeksjon i selskapet, og den er også en av verdens mest populære. Enter studio skal derfor jobbe utelukkende rundt denne sjangeren.

Kompleksiteten rundt server-klient kommunikasjon gjør dette til en vanskelig sjanger, forbeholdt utviklere med bakgrunn i både viderekommen programmering og nettverk. Dette er en av grunnene til at få uavhengige selskap utvikler slike spill: De mangler kompetansen. Den største fordelen med en Dataingeniørutdannelse fra HVL i forhold til for eksempel en fagskoleutdannelse i spilldesign er at man får nettopp denne utvidete kompetansen, som gjør en i stand til å utvikle disse mer komplekse spillene.

Spillet skal ikke bare dra nytte av nettverkspensumet til bachelorstudiet, men også resten av emnene: Programmering, algoritmer, fysikk, matematikk, logikk, smidig utvikling og grafikk. På den måten skal spillet også være representativt for Dataingeniørutdannelsen på HVL.

Tidligere arbeid

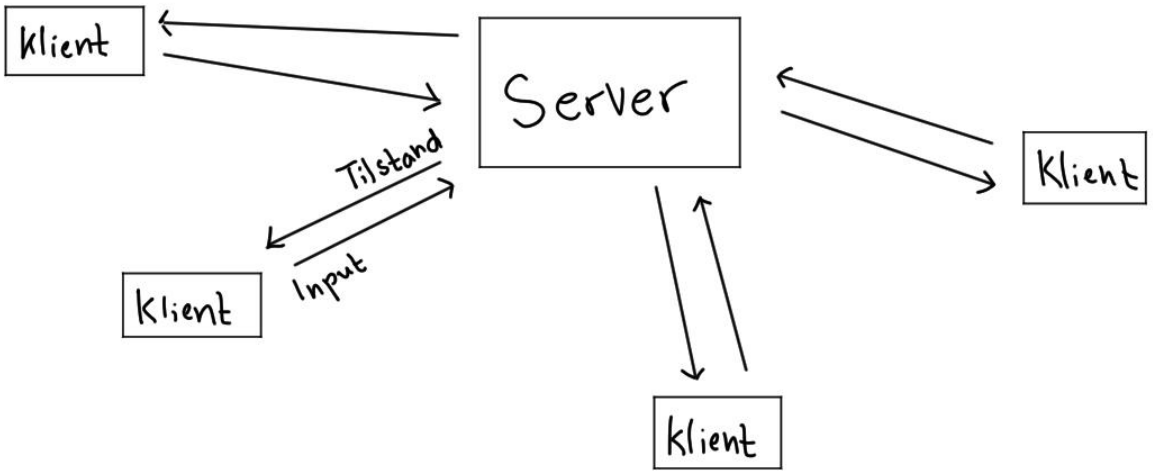
Det er særlig et eksisterende dataspill jeg vil dra vesentlig inspirasjon fra: *Rocket League*. Dette spillet er lite, kompakt og har minimal funksjonalitet, men er samtidig et av verdens mest populære spill. Konseptet er enkelt: fotball med biler (https://www.rocketleague.com/). Spilleren kan kjøre, hoppe, og har en jetmotor bakpå bilen som gir muligheten til å hoppe ekstra høyt (derav navnet *Rocket* league). Det er tre spillere på hvert lag, og alle banene har samme utforming. Et såre enkelt, men ekstremt velutformet konsept. Med tanke på tidsrommet for bacheloroppgaven vil en såpass enkel ide være en fordel.

Figur 2.1: *Rocket League - et enkelt men velutformet konsept (Rocket League 2021).*

Utover dette skal spillet i tillegg inneholde så mange konsepter fra Battle Royale sjangeren som mulig. Dette er spill der hensikten er å være den siste gjenlevende i et utvalg personer. Sjangeren er basert på filmen “Battle Royale”, som handler om en klasse japanske tenåringer som blir sendt ut til en øde øy for å kjempe mot hverandre.
Sjangeren er hovedsaklig fokusert rundt “First Person Shooter” subtypen, hvor man opplever omverden gjennom øynene til avataren man spiller. Her finner man våpen, leter etter motstandere, og legger feller for de andre innenfor et definert område. Området krymper med tiden, noe som gjør spillet mer og mer intenst. Det gjør også at det utspiller seg over en mer definert tidsramme; etter ti minutter er kanskje området så lite at det er vanskelig å unngå motstandere. Dette forhindrer at en kamp tar for lang tid.

Figur 2.2: *Fortnite - et av verdens mest poulære spill de siste fem årene (Fortnite 2021).*

Dette byr på mange interessante problemstillinger i programvaren. Det behøves en felles enighet om spillets tilstand, og hvis spillet bygger på bevegelse, taktisk plassering og rask reaksjonsevne må denne tilstanden være synkronisert med minst mulig forsinkelse - helst mindre enn den menneskelige terskelen til å føle forsinkelse, rundt 20 millisekunder. Alle spillerene opplever at de spiller på sin egen maskin, men i realiteten spiller alle på èn felles ekstern servermaskin, som siden sender en kopi av spillets tilstand tilbake til alle klientene. For å oppnå tilfredsstillende tilstandsforsinkelser må man gjøre en hel del triks. Dette er en av de største utfordringene med flerspillerfunksjonalitet.



Figur 2.3: Server/klient kommunikasjon.

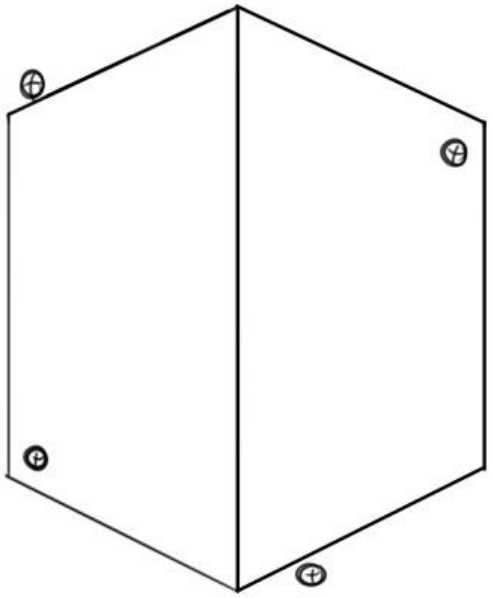
En annen stor utfordring er det kreative aspektet: Man kan lage et spill som møter alle de tekniske kvalifikasjonene, men hvis det mangler underholdningsverdi er alle disse tekniske utførelsene bortkastet. Man kan bruke årevis på å perfeksjonere oppdateringsfrekvens og grafikkbehandling hos klientene, men hvis ingen liker spillet er det ikke mange som kommer til å kjøpe det.
I boken “Fundamentals of Game Design” *(Adams 2014 s.17)* pekes det på egenskaper av ulike utfordringstyper som oppleves som underholdende. Under typen “Physical Coordination Challenges” pekes det på:

- Speed and reaction time.
- Accuracy or precision.
- Timing and rythm.

Disse egenskapene er iboende egenskaper i esport- og Battle Royale sjangrene, og vil bli tatt høyde for under utformingen av spillkonseptet.

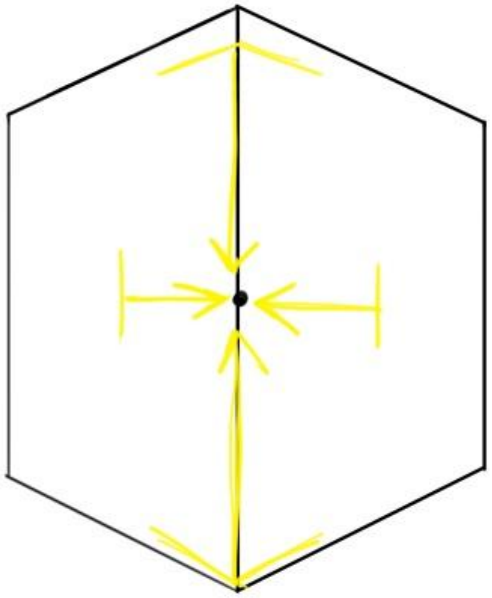
Initiell løsningside

I den opprinnelige konseptplanen for spillet ble følgende ide skissert:



Figur 2.4: Kube og spillere.

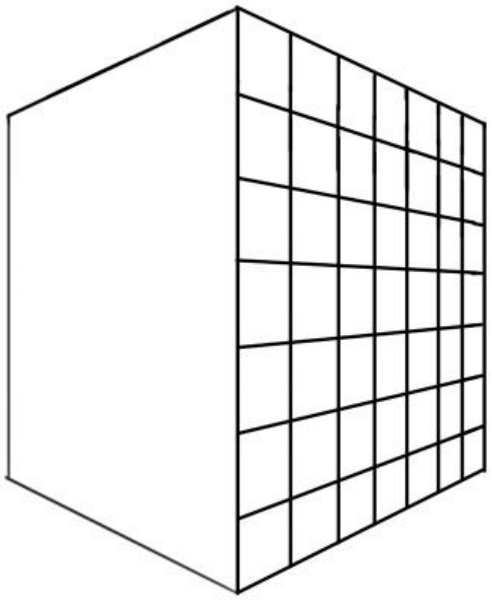
Hver spiller styrer en kule. De plasseres på en stor kube. På denne kuben er gravitasjonen alltid inn mot midten, slik at hvis noen ruller over kanten på kuben så blir den nye gravitasjonen hens relativ til den siden hen ruller på. Gravitasjonen er illustrert her med gule vektorer.



Figur 2.5: Gravitasjonsvektorer.

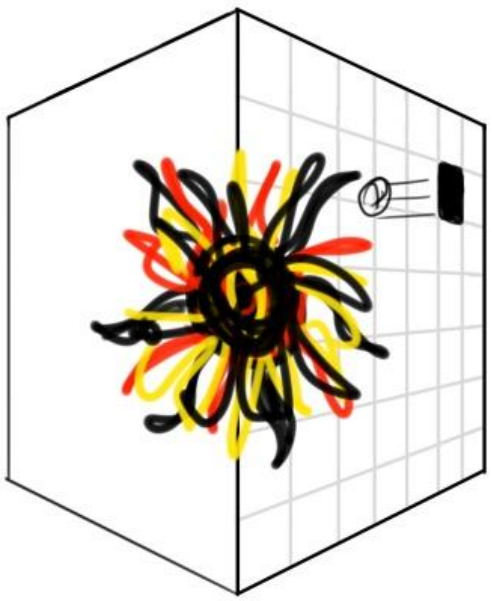
Hver side av kuben er bygget opp av et areal kvadratiske ruter. Hvor mange ruter hver side skal bygges opp av bestemmes av antall spillere. Hvis det er kun 4 spillere er en side for eksempel bygget opp av 10 * 10 ruter. Med 8 spillere kan man øke til 20 * 20 ruter. Dette er

en økning fra 600 ruter (10 ruter * 10 ruter * 6 sider) til 2400 ruter (20 ruter * 20 ruter * 6 sider). Arealet øker altså kvadratisk.



Figur 2.6: Kuben er bygget opp av ruter.

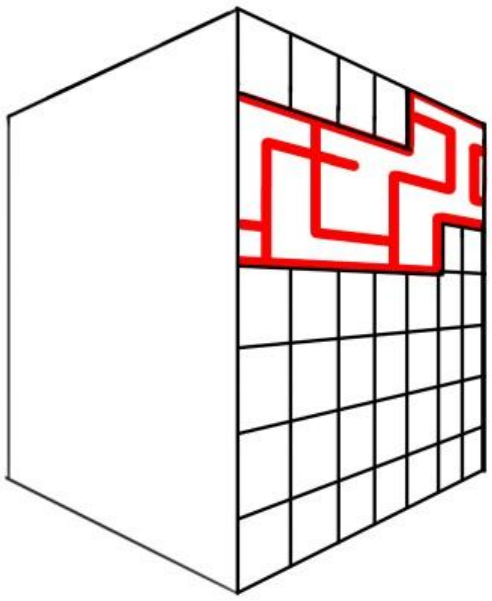
Spillerene ruller rundt på kuben, og når de kommer i kontakt med en rute, aktiverer de destruering av denne. Etter to sekunder forsvinner ruten, og hvis de ikke har rullet videre til en ny rute faller de inn i et svart hull i midten av kuben. Etter hvert som de spiller går altså kuben i oppløsning, og de får mindre og mindre areal å rulle på.



Figur 2.7: I midten av kuben er et svart hull som sluker spilleren.

I tillegg til dette har de ferdigheter de kan bruke strategisk mot hverandre. Hver gang de ruller over en rute og den forsvinner, tjener de 1 kreditt. Kredittene kan de bruke på disse ferdighetene. Akkurat hvilke ferdigheter spillerene skal kunne velge mellom er ikke bestemt enda! Det finnes sidevis med ideer til ferdigheter i prosjektets “brainstormingbok” - men siden ferdighetene må prototypes, testes og vurderes med flere spillere må kjernen bygges først. Siden kan ferdighetene prototypes, og de beste velges ut og utvikles videre.
Å hoppe blir sannsynligvis en ferdighet - et hopp til prisen av 1 kreditt. En form for angrep mot andre spillere må også være med, og kanskje noe som lar spillerene fjerne ruter strategisk. Med 2400 ruter (20 * 20 * 6) Vil 8 spillere ha gjennomsnittlig 300 poeng å rutte med. Prisen på ferdigheter kan justeres ut ifra det - men perfekte kredittpriser må nesten justeres under testing.

Denne opprinnelige ideen ble siden utviklet videre til å ha et underliggende mønster/veinett under rutene. En spiller ruller over en rute, og ruten forsvinner, men en form blir igjen. Disse formene danner et slags underliggende vilkårlig mønster som spillerene må balansere på:



Figur 2.8: Underliggende mønster.

Mønsteret skal bygges “prosedyremessig”, det vil si at hver gang man starter en ny kamp skal det underliggende mønsteret være annerledes. Det skal også gi mening, ikke bare slenges sammen av tilfeldige biter, men danne noe som faktisk ligner et sammenhengende veinett.

Løsningskrav

For å utføre denne ideen må man bruke konsepter fra hele bachelorstudiet:

- *Programmering:* Unreal Engine bruker C++, med heftig bruk av arv/klassehierarki.

- *Algoritmer:* Det underliggende mønsteret vil kreve en algoritme som bygger det fra bunnen av hver gang man starter en ny kamp.

- *Fysikk og matematikk:* Gravitasjonssystemet vil kreve kontinuerlig vektorkalkulering på hver spiller. Spillerbevegelse vil bruke funksjonalitet fra fysikkmotoren (kulen ruller ved at den får påført en kraft i den retningen spilleren bestemmer med kontrollen).

- *Logikk:* Spill har mye boolsk algebra i seg; for eksempel for å sjekke hva en spiller har lov til å gjøre, om noen har vunnet, etc.

- *Smidig utvikling:* Det er umulig å vite akkurat hva som er gøy før man har prøvd. Man kan spekulere; men spillerfunksjonalitet utover det basale (som bevegelse) må nesten testes og vurderes - ikke bare av èn, men av flere spillere. Dette krever testing med feedback, og tilpassning etter resultatene. Spillutvikling er helt klart en prosess som krever smidig tilnærming; kanskje mer enn annen mykvare, nettopp på grunn av den iboende x-faktoren.

- *Grafikk:* Det vil ikke bli brukt noen eksterne ressurser. Det betyr at alle 3D-modeller, teksturer og animasjoner skal lages internt i prosjektet.

- *Nettverk:* Den mest komplekse delen av prosjektet blir å håndtere replikering av serveren til alle klientene med lavest mulig forsinkelse. Man kan ikke replikere alt - man må velge de viktigste tingene slik at linjesignalet får lavest mulig belastning inn til og ut fra server.

Litteratur om problemstillingen

Man kan ikke akkurat påstå at det er mangel på litteratur om spillutvikling. Nettet er overmettet. Det finnes tilogmed store konferanser tilegnet emnet (også her i Bergen - Konsollkonferansen er kjempepopulær. https://konsoll.org/). All dokumentasjon om programvaren er lett tilgjengelig på nett, og nettforum gjør det mulig å søke etter veldig spesifikke problemstillinger og finne svar. Både spillmotoren Unreal Engine, utviklingsmiljøet Visual Studio og 3D-modelleringsmykvaren Blender har forumer og ekstensiv dokumentasjon på nett. Når det gjelder spilldesign vil jeg hovedsaklig lene meg til boken “Fundamentals of Game Design” av Ernest Adams.

Servertjenesten Amazon Web Services har lite dokumentasjon på hvordan man bruker det i tandem med Unreal Engine. Dette blir den største utfordringen, for utvikleren har ingen tidligere erfaring med dette. Dette påfører prosjektet en risiko. Det vil bli satt av ekstra mye tid til dette, for å kompensere for mangel på kompetanse og læringsressurser.

Design av prosjektet

Den ideelle løsningsmetoden ble avgjort allerede i den konseptuelle utviklingen, men flere alternativer ble vurdert:

Alternativ til spillmotor

Det finnes flere spillmotorer å velge mellom, men ikke alle besitter verktøy for flespillerfunksjonalitet. De mest relevante valgene var:

- *Unreal Engine:* Motoren var i utgangspunktet et internt utviklingsverktøy hos Epic Games, som ble brukt til å lage førstepersonskytespillet Unreal (utgitt 1998) og senere flerspillerversjonen Unreal Tournament (utgitt 1999). Unreal Tournament-serien har siden vært en søyle i nettverksbasert flerspillerhistorie. Siden Epic Games hovedsaklig bruker motoren til egne utgivelser og oppdaterer den ved behov for ny funksjonalitet i egne spill, er Unreal Engine spesielt egnet til den type nettverksspill de utvikler. Den nyeste produktserien til Epic Games er Battle Royale spillet Fortnite; et av verdens mest populære spill.

- *Unity:* En veldig populær spillmotor blandt mindre selskap, utgitt for første gang i 2005. I utgangspunktet var Unity ment som en billig løsning for mindre utviklingsselskap, og dette er fremdeles motorens hovedfokus.
  De fleste spillmotorene på markedet er basert rundt en lisensieringsmodell; de er "gratis", mot at man betaler en viss sum av inntektene tilbake til de hvis man tjener penger på spillet. Unreal Engine tar for eksempel 5% av bruttoinntektene etter tjente 30.000 kroner. Dette kan fort koste mange tusen kroner for et mindre selskap (eller millioner for et større selskap). Unity koster kun 2000 kroner i måneden (fast beløp per utvikler) *hvis* man tjener mer enn 1000.000 kroner i året. I de aller fleste tilfeller vil dette lønne seg økonomisk; men det kommer selvsagt også ann på hvilke funksjonaliteter man behøver i motoren. Det er oftest dyrere (men fullt mulig) å bruke tid på å utivde motoren med funksjonalitet som ikke allerede er implementert.
  Unity tilbyr i utgangspunktet den samme funksjonaliteten som Unreal Engine, men er ikke typisk brukt til flerspillerfunksjonalitet. Unreal Engine har bedre funksjonalitet for grafikk, nettverk, esport og typiske Battle Royale egenskaper. Unity er gjerne bedre på å utvikle mindre krevende spill, som spill til mobiltelefoner (som også er et stort, stadig voksende marked). Unity blir ofte sett på som mer effektiv, nettopp fordi at mye funksjonalitet ikke er automatisk implementert.

- *Amazon Lumberyard:* En relativt ny spillmotor (2015) med en kontroversiell fødsel. Motoren bygger på en annen spillmotor, den svenske "Cry Engine", som er høyt annerkjent for sin grafikkkapabilitet. I utgangspunktet "stjal" Lumberyard kodebasen til Cry Engine, og hadde ingen intensjon om å betale for den, men endte til slutt med å måtte betale $70 millioner til Crytek i lisensavgifter. Lumberyard er (utrolig nok): hundre prosent gratis, og muligens enda bedre på serverfunksjonalitet enn Unreal Engine - Amazon er verdens største serverforsørger, og har innlemmet funksjonalitet mot sine egne servertjenester direkte i motoren. Desverre er Lumberyard en ung spillmotor, som betyr at den er særdeles vanskelig å utvikle med. Dokumentasjon, opplæringsvideoer og brukerforum er også mangelvare.

Det er altså en del fordeler og ulemper her å veie opp mot hverandre. Amazon Lumberyard er helt klart fristende, med innlemmet serverfunksjonalitet og gratis lisensiering. Unity har mindre tydelige fordeler for dette spesifikke prosjektet. Hovedsaklig faller valget av spillmotor på Unreal Engine basert på følgende grunnlag:

- Utvikleren har utstrakt erfaring med Unreal Engine. Han har lite erfaring med de to andre motorene. Gitt tidsrommet for bacheloroppgaven er det for tidkrevende og risikabelt å skulle lære seg en helt ny spillmotor, selv om Lumberyard er et veldig fristende alternativ.

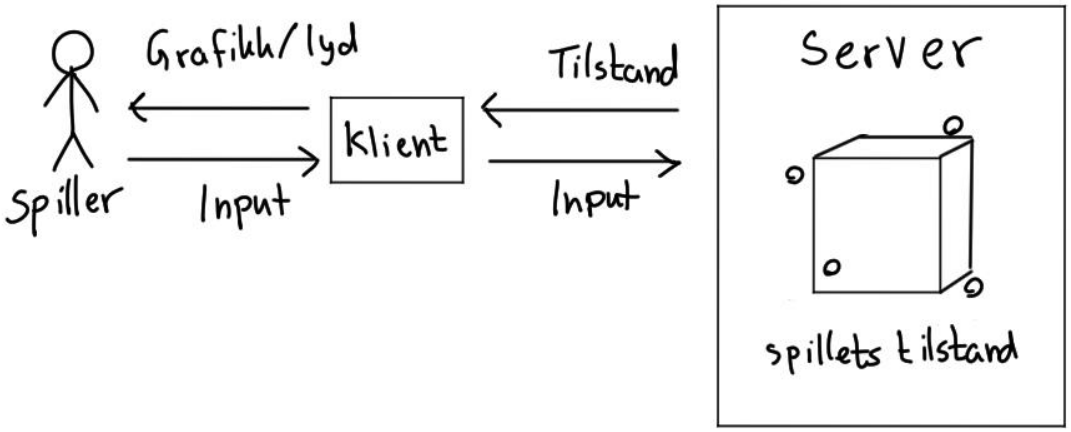
- Unreal Engine er gammel og vis, spesielt når det gjelder flerspillerfunksjonalitet. Den er robust og (relativt) enkel å bruke.

- Det er et godt, behjelpelig utviklersamfunn rundt denne motoren, noe som fort kan bli uvurderlig når problemene blir ugjenkjennelige. Dokumentasjonen er prisløs.

Valg av spillmotor: Unreal Engine!

Alternativ til serverlogikk

Når man skal lage et spill med flerspillerfunksjonalitet, må man implementere server/klient logikk. Serveren kjører en egen versjon av spillet (hovedversjonen), som holder rede på alle variabler som er felles for alle spillerene. Alle spiller egentlig på denne eksterne maskinen, ikke på sin lokale maskin (klienten). Alle lokale kontrollbevegelser og tastetrykk sendes fra klienten til serveren, som utfører funksjonaliteten hos seg, og siden replikeres denne informasjonen tilbake igjen til alle klientene. Spillets tilstand på spillerens lokale klient er kun en kopi av spillets tilstand på servermaskinen. I tillegg må alt synkroniseres hvis man skal unngå uenigheter i spillets tilstand. Det er umulig å unngå forsinkelser i signalet, så her må man planlegge spillfunksjonalitet i forhold til forventet sgnalforsinkelse, som igjen er avhengig av valg av kommunikasjon- og servertype.



Figur 3.1: Spiller/klient/server kommunikasjon for en enkelt spiller.

Når det gjelder servertyper finnes det hovedsaklig to typer å velge mellom:

- *Ekstern server (Dedicated Server):* Spillet kjører på en tilegnet maskin ekstern for alle spillerene. Maskinen eksisterer for eksempel på en serverfarm i Frankfurt, og spillere fra Norge, Sverige, Tyskland, Spania, England og Portugal kobles direkte til

denne maskinen. Alle spiller på denne maskinen, og tilstanden replikeres tilbake til klientene. Denne eksterne maskinen behøver ikke å gjengi grafikk eller lyd; den behandler kun alle spillobjektenes tilstand (som lokasjon, rotasjon, verdier, osv). Klientene gjengir spillets tilstand i form av grafikk og lyd til hver enkelt spiller. Å gjengi grafikk er ofte det mest ressurskrevende, så her sparer man serveren for mye arbeid. Serveren har derimot intensiv prosessering av spillerforespørsler og objekttilstander, og høyt trykk på kommunikasjonslinjene (både inn fra og ut igjen til klientene).

- *Klientbasert Server (Listen Server):* En av klientene fungerer samtidig som server. Det betyr at spillets tilstand kontrolleres av denne maskinen samtidig som den gjengir grafikk og lyd til en av spillerene. Dette er hovedsaklig brukt til sammarbeidspill med to til fire spillere, ikke til kompetativt spill for flere enn fire spillere.

Når det gjelder kommunikasjonstyper finnes det to relevante:

- *LAN (Local Area Network):* God, gammeldags LAN forbindelse var veldig populært for flerspillerfunksjonalitet før nettverkskommunikasjon ble såpass rask som den er nå. Dette er fremdeles den raskeste og mest robuste kommunikasjonsformen, men har den fysiske begrensningen at klientene må kobles sammen med nettverkskabler. WLAN (Wireless LAN) gjør at man ikke behøver de fysiske kablene, men man er likevel begrenset til det trådløse signalets rekkevidde. Hvis klientene alltid befinner seg i samme rom (eller hus) er dette ofte det optimale valget. Tilnærmet null forsinkelse av signal (under 1 ms) gjør det svært enkelt å utvikle spill for LAN og WLAN.

- *Internett:* For å koble til en server utenfor signalrekkevidden må man bruke internett. Da behøver man IP-adressen og portnummeret til serveren man vil kobles til, samt passord og annen informasjon som kreves av serveren. Disse kravene er nærmere spesifisert av servertjenesten og spillmotoren, og vil variere med applikasjonen.

Siden spillet skal ta inspirasjon fra Battle Royale og esportspill som Rocket League, gir det mest mening å tilby spilleren kamper mot spillere fra hele verden, og ikke bare venner i samme husholdning. Dette betyr at spillere må kunne kobles sammen via internett. Når det gjelder servertype er det flere grunnlag for å velge en dedikert server:

- *Stabil kommunikasjon:* Hvis èn klient skal ta hånd om spillets tilstand, er det større risiko for ustabil kommunikasjon siden det er vanskelig å forutse klientens maskinvarespesifikasjoner. Selv om man kan implementere logikk for å velge den *best egnede* klienten som server, er det ikke sikkert at *noen* av klientene er egnet til å fungere som både klient *og* server; Dessuten er det ganske høyt trykk på kommunikasjonslinjen allerede med mer enn fire spillere. Da er det mye bedre å bruke en dedikert server som er designet for nettopp dette.

- *Forhindre juks:* Juks blir stadig mer utbredt i esport, og mange useriøse spillere jobber iherdig med å hacke nye spill, spesielt kompetative flerspillerspill, for å ødelegge konkurransen og jukse seg opp til toppen. Dedikerte servere tilbyr et ekstra lag med sikkerhet mot juks, siden spillets tilstand bestemmes på serversiden og klientene kun har en kopi av denne tilstanden. For å jukse må hackeren isåfall bryte

seg inn på serveren, og hvis man bruker en seriøs servertjeneste er dette veldig vanskelig.

- *Ekstra funksjonalitet*: Servertjenester kan tilby ekstra funksjonalitet som databaser for lagring av spillerdata, påloggingssystem, automatisk opp- og nedskalering av maskinvare, billettsystem for køorganisering, osv. Et godt eksempel er Amazon Web Services sitt system for kampgenerering: Spillerinformasjon kan lagres i databasen med variabler som bestemmer hvem de skal matches opp mot - nybegynnere møter nybegynnere, eksperter møter eksperter. Klienter i nær geografisk rekkevidde kobles sammen for å unngå forsinkelser. Hvis man melder noen for dårlig oppførsel vil man ikke møte denne personen igjen. Rocket League har implementert et ekstremt godt slikt system.

Valg av servertype: Ekstern server!
Valg av kommunikasjonstype: Internett!

Alternativ til 3D-modelleringsmykvare

- *3D Studio Max:* Dette er mykvaren utvikleren er opplært i fra før, og den er veldig godt egnet til å lage modeller og animasjoner til spill. Desverre koster den 2000 kroner i måneden hvis man vil bruke 3D-modellene i en utgivelse, selv om utgivelsen ikke engang tjener penger.

- *Blender:* Mykvare med åpen kildekode som er garantert gratis å bruke, uansett inntekter. Denne mykvaren har hatt en kraftig vekst av funksjonalitet og brukere de siste fem årene, og er i manges øyne bedre enn (veldig) kostbare alternativ. Mykvaren kan gjøre alt fra modellering og animering til teksturering og redigering. Stadig flere profesjonelle spillutviklingsselskaper bruker Blender i sin interne arbeidsprossess.

Målet med dette prosjektet er å få økonomisk støtte til fullstendig ferdigstillelse. Derfor har Blender blitt valgt, slik at all grafikken som lages kan brukes problemfritt i en eventuell utgivelse.

Valg av 3D-modelleringsmykvare: Blender!

Prosjektmetodikk

Smidig utvikling vil være i hjertet av prosessen. Først bygges kjernen, siden utvides denne kjernen med funksjonalitet så langt budsjettet tillater, eventuelt til kunden er fornøyd. I dette prosjektet er “kunden” selve *spilleren*, representert av veiledere og eksterne spilltestere (venner og frivillige fra klassen i første omgang, siden mest mulig utenforstående personer). Underveis utføres stresstester så ofte som mulig.

- *Kjernen:* Spillet trenger først og fremst kuben bygget av ruter, og spillere som kan rulle på denne. Rutene skal forsvinne når spilleren ruller over de, og avdekke et underliggende mønster. Spillerens gravitasjon skal være relativ til siden de befinner seg på. Siden nettverksbasert flerspillerfunksjonalitet har så stor innvirkning på

programstrukturen må kjernen bygges med hensyn til server/klient kommunikasjon allerede fra begynnelsen av.

- *Servertesting*: Spillet må implementeres på en dedikert serverløsning så tidlig som mulig, slik at man kan teste og forsikre seg at server/klient funksjonaliteten faktisk fungerer. Unreal Engine har funksjonalitet for å simulere nettverksforbindelse mot server, og denne vil bli brukt. Det er derimot en simulering, og det er forventet at feilsøking på en reell implementasjon vil avdekke en hel del problemer i koden.

- *Tidlig testing:* Så fort spillet har blitt implementert på en server med kjernefunksjonalitet vil spillet testes med eksterne spillere for å avdekke eventuelle problemer i kommunikasjonen. Kritikk av kontroll og spillerfunksjonalitet er også velkommen her.

- *Utvide funksjonalitet / testing med tilbakemelding:* Når kjernen er stabilisert vil prosjektet gå over i en testbasert, iterativ prosess, der ny funksjonalitet vil bli lagt til basert på tilbakemeldinger fra testere. Dette vil pågå til prosjektet er ferdig.

Bygge kjerne > Implementere på server > Stabilisere > Utvide funksjonalitet.

Prosjektplan

Utvikleren har et valgemne fra UiB, som ikke er organisert i forhold til HVL sitt vanlige 6. semesteropplegg. Istedenfor å ferdigstille valgemnet og eksamen i mars, og deretter jobbe utelukkende med bacheloroppgaven, må utvikleren gjøre bacheloroppgaven i tandem med valgemnet hele semesteret, da valgemnet har eksamen 7. juni.

Dette gjør det egentlig lettere å planlegge, siden hele semesteret får et statisk opplegg uten store overraskelser. Bacheloroppgaven tildeles 2/3 av den totale studietiden for hele semesteret, cirka 30 timer i uken. Valgemnet (INF250 ved UiB) tildeles 15 timer. INF250 har faste ukedager for forelesninger hele semesteret - tirsdager og fredager. Disse to dagene dedikeres til valgemnet med 8 timer per dag. Bacheloroppgaven tildeles mandager, onsdager, torsdager og lørdager med 8 timer per hverdag og 6 timer på lørdagene.

En overordnet plan for semesteret:

- *Uke 1 - 9:* Bygge kjernen.
- *Uke 9 - 13:* Implementere kjernen på en dedikert server.
- *Uke 13 - 17:* Tidlig testing, stabilisering av kjerne, planlegge utvidet funksjonalitet.
- *Uke 17 - 21:* Iterativ utvikling, med intensiv testing og utviding av funksjonalitet i forhold til tilbakemeldinger fra testere.
- *Uke 21 - 23:* Ferdigstilling av prototype for visning til eventuelle interessenter. Dette inkluderer en reklamefilm av prototypen, og en plan videre i fall økonomisk støtte skulle mottas.

Hvert punkt planlegges nærmere i punktets innledningsfase. Disse mer detaljerte planene, og utførelsen, vil være hovedfokuset for neste del av rapporten, “Detaljert Design”. Her vil selve mykvarestrukturen presenteres del for del, i takt med punktlisten.

Risikovurdering

Det er reelle risikoer forbundet med dedikerte serverløsninger. Denne risikoen øker spesielt i omfang siden utvikleren mangler kompetanse på området. Det vil bli lagt ekstra mye tid på dette for å kompensere for og minimere risikofaktoren. Risikofaktorer forbundet med dedikerte servere:

- *Forsinkelse i signalet:* Dette er helt klart den viktigste og vanskeligste problemstillingen ved nettverksbaserte spill. Dette er umulig å unngå. Det vil alltid være forsinkelser i signalet når server/klientkommunikasjonen sendes over lange distanser. Valg av servertjeneste vil være viktig, men dette kan kun redusere problemet, ikke fjerne det. Spillfunksjonalitetene må designes rundt disse forsinkelsene. Unreal Engine har funksjonalitet for å simulere forsinkelser mellom server og klient, slik at det er lettere å utvikle rundt de.

- *Kommunikasjonsavbrudd:* Klientene er koblet til en ekstern server via internett. Hvis denne kommunikasjonslinjen brytes under spilling vil spilleren miste forbindelsen og kastes ut av kampen. Årsaken til dette ligger ofte hos klienten, men både klienten og serveren må vite hvordan de skal forholde seg til dette problemet og løse det hvis det oppstår. Det skal for eksempel ikke ødelegge kampen for resten av spillerene. Serveren skal ikke kræsje.

- *Prisøkning:* Servertjenester er ikke gratis. Alt som koster penger står i faresonen for å øke i pris. Spillet må selges til en pris som kompenserer for disse kostnadene - forandring i serverkostnader vil påvirke prisen på spillet. Det betyr også at spillet må ha en kontinuerlig inntektsflyt. Dette er en av grunnene til at mange serverbaserte spill har “in-game purchases”.

Detaljert design

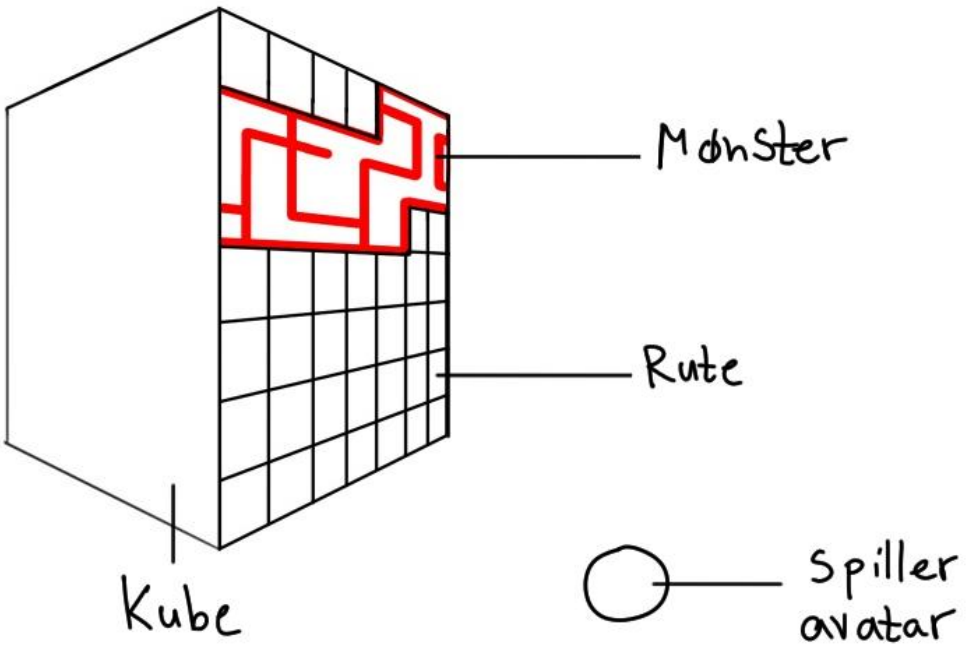
Denne delen av rapporten skal ta for seg en mer detaljert beskrivelse av prosjektets hovedfaser: byggingen av kjernen og den iterative utvidelsen etterpå.

Uke 1 - 9: Bygge kjernen

Kjernen av spillet er kuben, spilleren, og spillets fundamentale regler. Reglene blir mer detaljerte etter hvert som ny funksjonalitet blir implementert.

Fundamentale regler

I midten av kuben er et svart hull som sluker spillerene. Kuben går i oppløsning under spillets gang, og spillerene får mindre og mindre plass å bevege seg på. Hvis spilleren faller inn i det svarte hullet taper hen. Vinneren er siste stående.

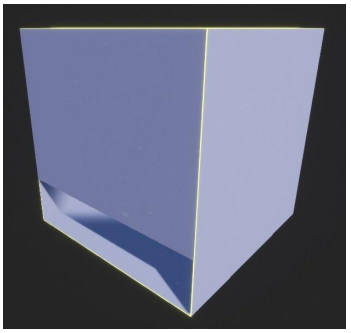


Figur 4.1: Kjernens hovedelementer. Kuben er laget av ruter, som igjen har et underliggende mønster. Spilleren er representert av en avatar.

Kuben

Kubeobjekt:

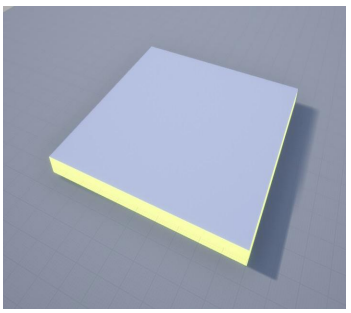
- Seks sider: Back, top, front, bottom, left, right.
- Matrise med 6s^2 ruter.



Figur 4.2: Kubeobjekt.

Ruteobjekt:

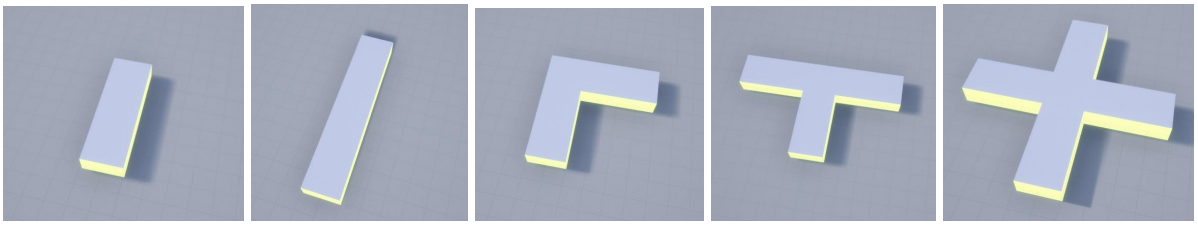
- Posisjon.
- Rotasjon.
- Siden den tilhører.
- Mønsterobjekt.
- 3D modell.



Figur 4.3: Ruteobjekt.

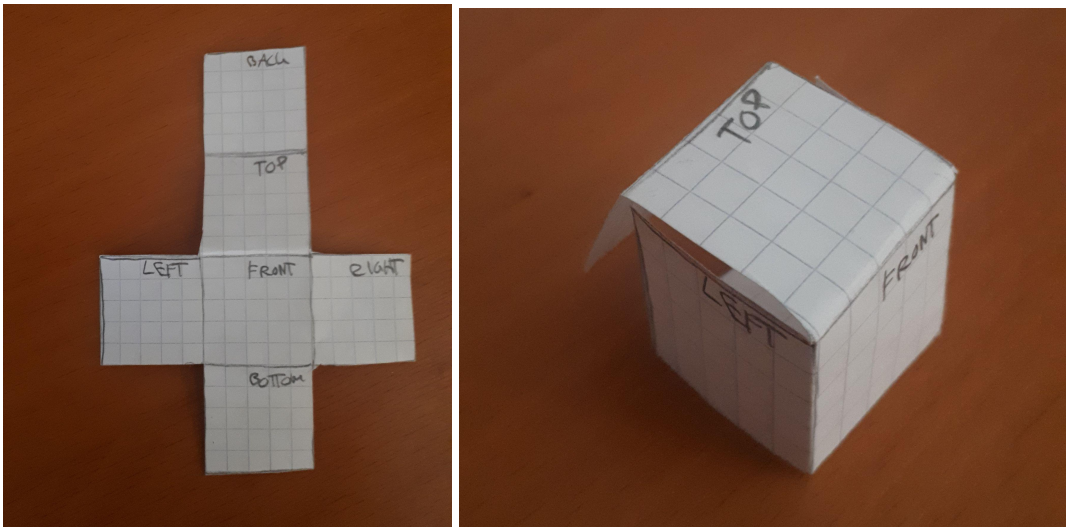
Mønsterobjekt:

- Rotasjon (relativ til ruten den ligger under).
- 3D modell (en av fem ulike valg).



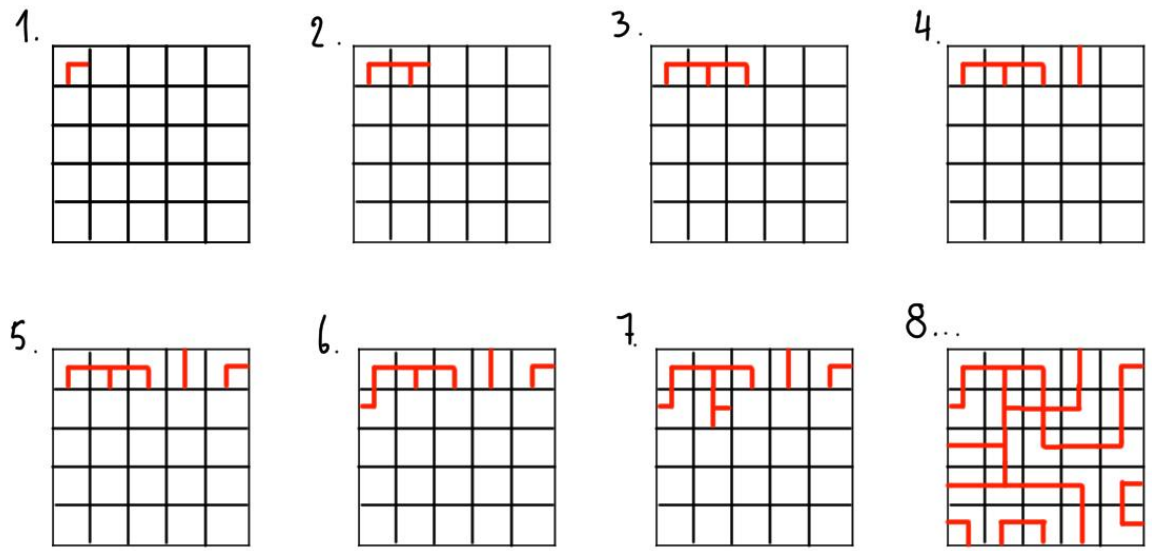
Figur 4.4: Mønsterobjekt. Fr.v: Stopp, rett, sving, T og kryss.

Konstruksjon:
En kube består av en todimensjonell matrise med ruteobjekter. Hvert ruteobjekt inneholder rutens posisjon og rotasjon i et tredimensjonalt rom. Utvikleren har skrevet en funksjon som genererer slike posisjoner rundt et sentrum (3-dimensjonalt nullpunkt). Rotasjonen er basert på hvilken side ruten tilhører.



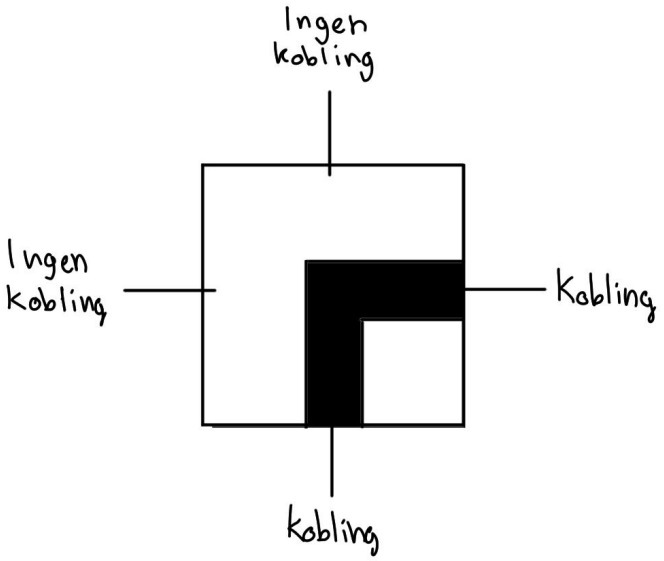
*Figur 4.5: Den originale skissen for kuben. I dette eksempelet kan man tenke at en side består av 5*5 ruter. Denne todimensjonelle matrisen kan man så brette rundt et tredimensjonalt nullpunkt for å få en kube. Posisjonen til en rute i matrisen kan alltid defineres ut ifra en variabel (s). I eksempelet over er antall ruter i en sidelengde s = 5. Man kan finne matriseplasseringen til midterste rute på toppsiden ved å regne ut 1s^2 + 2s + 3 = 25 + 10 + 3 = 38. s^2 gir siden (faktor 0 til 5), s gir raden (faktor 0 til s-1), konstanten gir kolonnen (0 til s-1).*
0s^2 + 0s + 0 gir aller første rute i matrisen.
5s^2+4(s)+4 gir aller siste rute i matrisen.

Algoritmen som genererer det underliggende mønsteret bruker blandt annet ruteposisjonen i matrisen. Den er lettest å forklare med bilder:



Figur 4.6: Mønstergenerering.

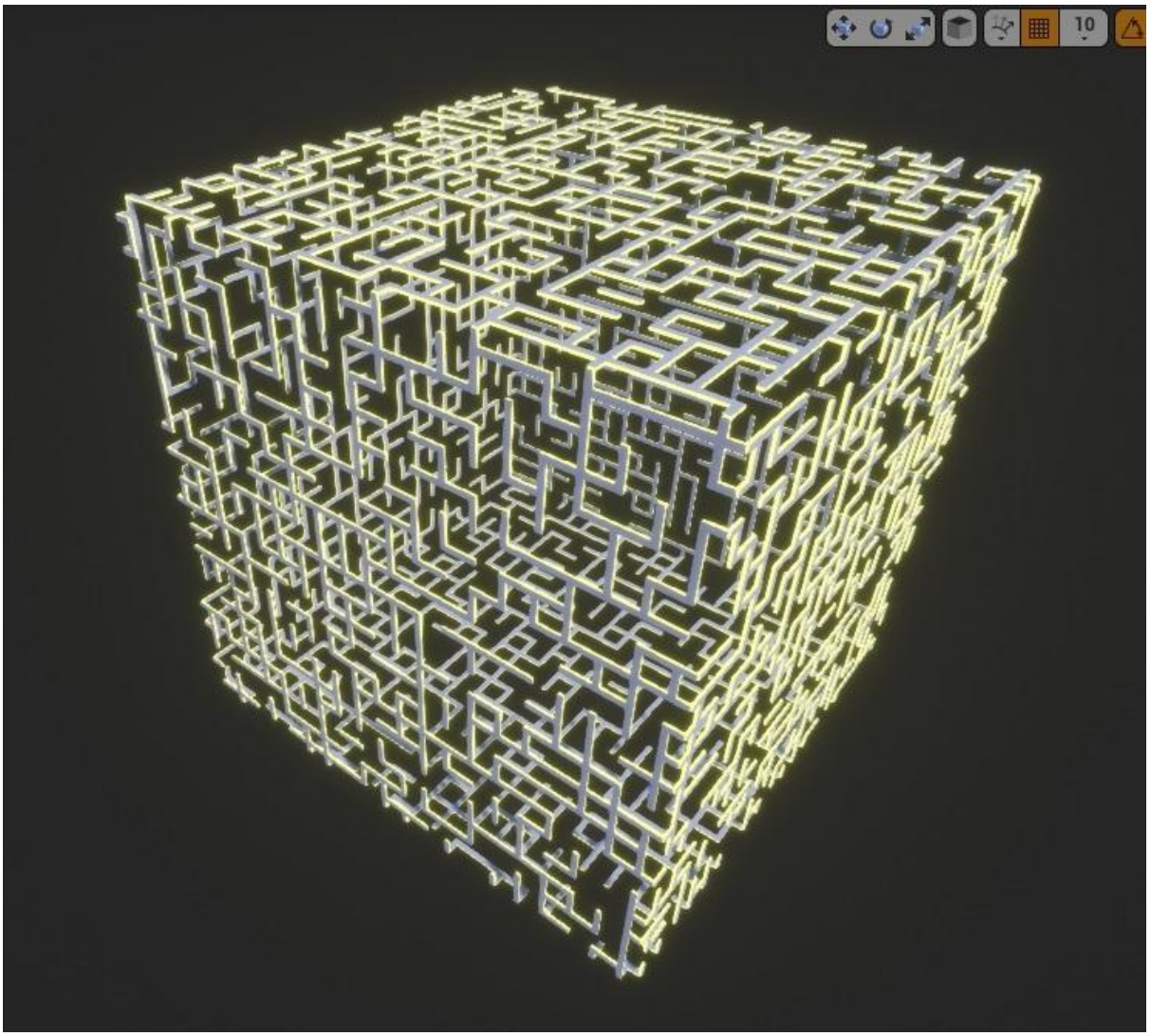
Den begynner med å opprette en helt tilfeldig form på første plass i matrisen. I dette eksempelet opprettes først en sving, med kobling mot sør og øst (1).



Figur 4.7: Sving med kobling mot øst og sør.

Den er koblet direkte mot øst, og direkte mot sør. Da må formene direkte øst og direkte sør ha tilsvarende imøtekommende koblinger.
Form nummer to i matrisen må derfor ha kobling mot vest, og form nummer seks må ha kobling mot nord. Så, i algoritmen får form nummer to automatisk kobling mot vest (2), og form nummer seks får kobling mot nord (6).
Hver genererte form sjekker rundt seg etter eksisterende koblinger. Hvis en sjekket retning ikke allerede har en definert kobling/ingen kobling, lages det en vilkårlig kobling i denne retningen. Dette fortsetter til en side er fylt opp med former (8). Algoritmen kjøres for alle sidene i kuben.

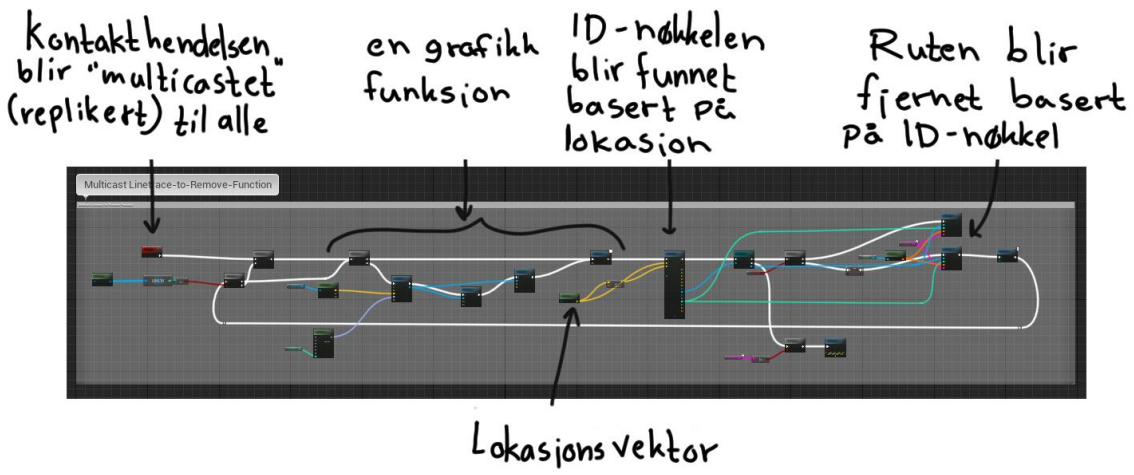
Algoritmen har en hel del mer detaljer, men den er ganske omfattende og utelates fra rapporten for å holde rapportlengden innenfor grensen. Resultatet blir slik:



Figur 4.8: Algoritmen mates med et tilfeldig heltall, og returnerer et tilfeldig kubemønster. På den måten kan man generere milliarder av mønstervariasjoner (2^32). En mer detaljert gjennomgang av denne algoritmen er å finne i bachelorprosjektbloggen på hjemmesiden til utvikleren: http://mariusirgens.no/Bachelor/21.Desember/21.Desember.html

Når alle disse 3D modellene skal tegnes på skjermen, sendes hver enkelt modell som en "draw call" til grafikkkortet. Med 6*20*20 ruter eller mønsterbiter blir dette 2400 drawcalls per bilde. Med en bildefrekvens på 120 bilder i sekundet svarer dette til rundt 300.000 drawcalls i sekundet. Dette er uakseptabelt høyt. Selv maskinen som spillet er utviklet på, med 8-kjerner 3.60 GHz prosessor og RTX 2060 grafikkkort, klarer ikke mer enn 30 bilder i sekundet * 2400 drawcalls, dvs rundt 70.000 drawcalls i sekundet.
Løsningen ble å bruke en funksjonalitet i Unreal Engine som heter "mesh instancing". Hvis man oppretter flere kopier av den samme modellen kan alle disse kopiene samles sammen i ett eneste drawcall. Disse modellene kan fremdeles ha ulik lokasjon, rotasjon og skalering. Dette er perfekt hvis man vil bygge større objekter av mindre byggesteiner, slik som i dette

prosjektet. Hele kuben konstrueres av kun 6 ulike modeller, noe som gjør at antall drawcalls går fra 2400 per bilde til 6. Dette er 400 ganger mer effektivt!
Det er selvsagt en rekke begrensninger ved å bruke instansierte modeller. Den vanskeligste begrensningen i dette tilfellet var at alle modellene blir lagt i en ikke-replikerbar liste, men for å få tak i en spesifikk modell må man aksessere denne listen. I dette spillet blir modeller fjernet og opprettet titalls ganger i sekundet, så denne listen aksesseres ofte.
Unreal Engine anvender seg av en reorganiseringsalgoritme på denne listen, som gjør at ID nøkkelen for et element forandrer seg hvis man fjerner et annet element fra listen. En rute kan ha ID nøkkel 500, og sekunder senere ID nøkkel 495. Akkurat hvilken nøkkel en modell får følger en optimaliseringsalgoritme som er vanskelig å følge. Dette gjør at en og samme modell får ulik ID nøkkel hos de forskjellige klientene på grunn av forsinkelser i signalet.
Når en spiller kommer i kontakt med en rute, og denne skal fjernes hos alle klientene, må altså serveren og alle klientene finne modellen individuelt. Istedenfor å finne ruten på serveren og distribuere ID nøkkelen, blir selve kontakthendelsen med ruten replikert, med all nødvendig informasjon som behøves for å utføre detektering av denne eksakte ruten på hver enkelt klient. På denne måten kan klientene individuelt finne hver sin unike ID nøkkel til ruten som skal fjernes, basert på lokasjonen til kontakthendelsen.



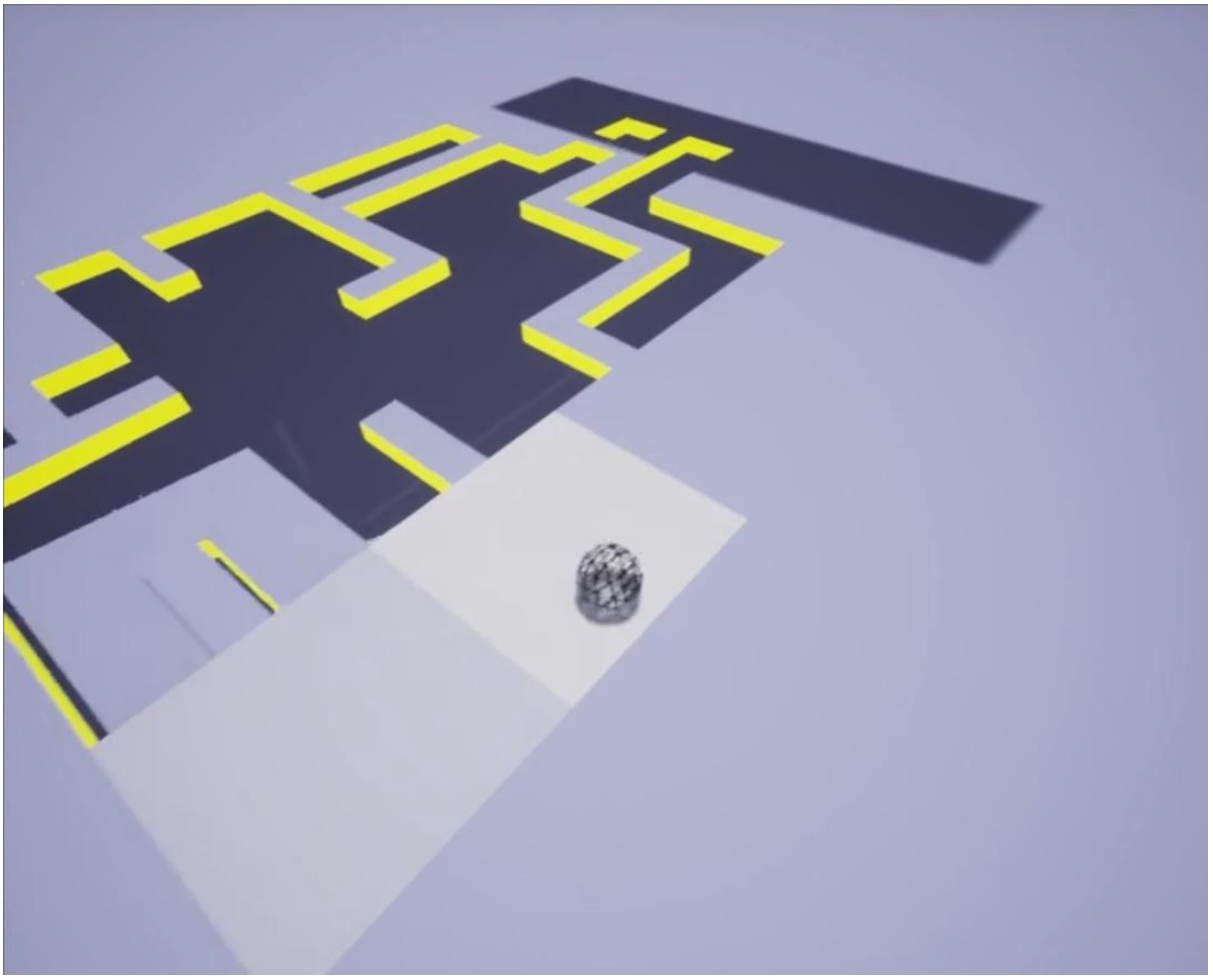
Figur 4.9: Bilde av rutefjerningsalgoritmen. Den første boksen aktiverer resten av koden. Denne aktiveringen replikeres fra serveren til alle klientene, slik at alle aktiverer resten av koden lokalt. Lokasjonen til hendelsen er også replikert.
Først utføres en visuell grafikkfunksjon (hvit markering av ruten). Siden finner funksjonen ruten i det tredimensjonelle rommet basert på lokasjonen til hendelsen. ID nøkkelen blir sendt videre til en funksjon som fjerner modellen fra listen av instansierte modeller, og dermed fra minnet.

Dette bildet viser også Unreal Engine sitt visuelle programmeringsinterface. Hovedsaklig er de detaljerte funksjonene programmert i Visual Studio (C++); siden setter man sammen disse funksjonene til større systemer i det visuelle interfacet.

Spilleren

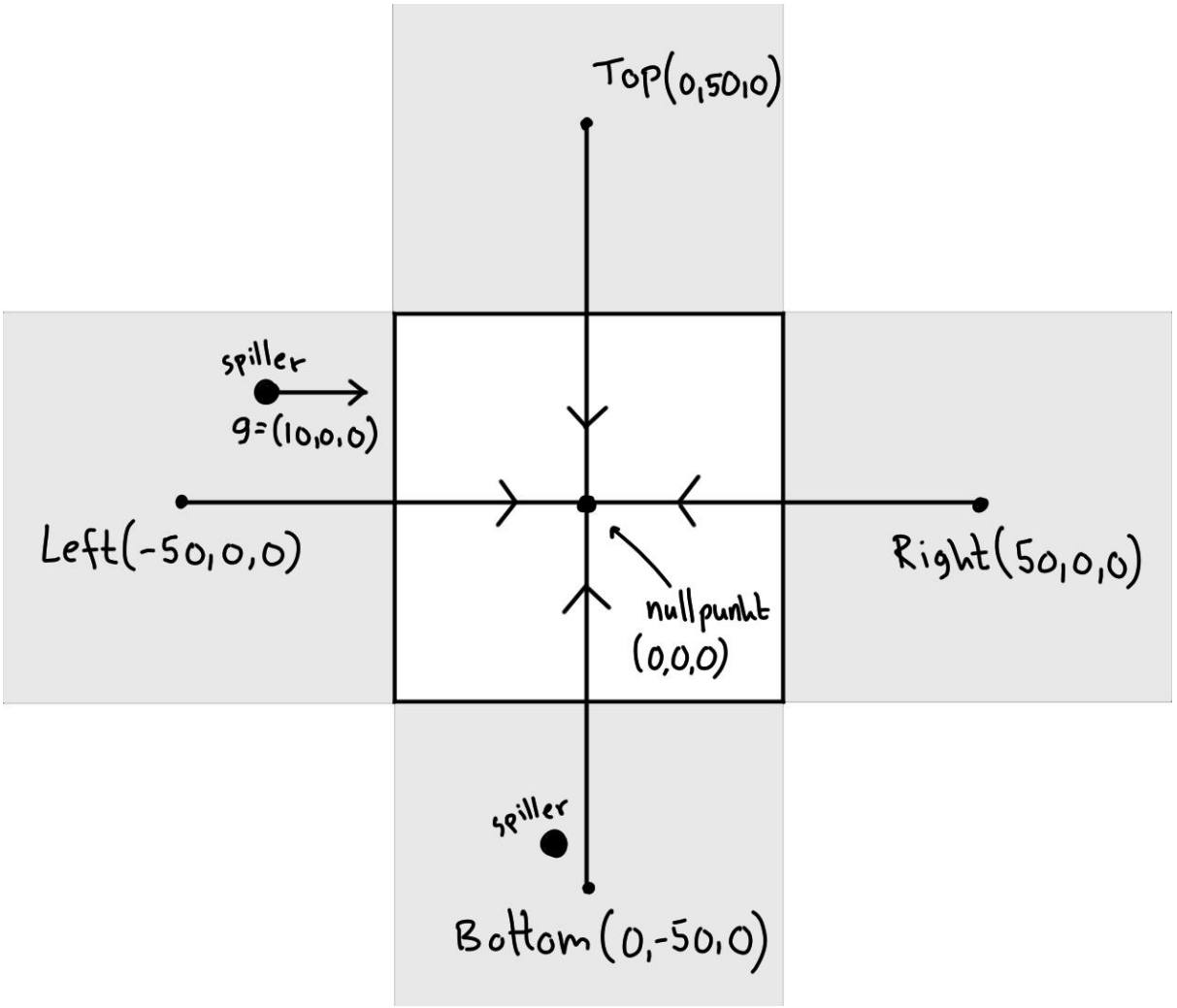
(Avataren har blitt forandret fra rullende kule til svevende droide etter nettverk- og brukertesting, men bortsett fra utseendet på avataren er koden stort sett lik; kun rullingen er fjernet fra det opprinnelige spillerdesignet, og ny bevegelseslogikk har blitt implementert. Dette forklares nærmere under Uke 13 - 17: Tidlig testing, stabilisering av kjerne, planlegge utvidet funksjonalitet):

Hver spiller styrer en kule, som ruller på kuben. Kulens gravitasjon er alltid relativ til siden de ruller på. Kulen ruller ved hjelp av fysikk - Når spilleren i holder en retning på kontrolleren påføres kulen en kraft i den retningen.
Hvis en spiller kommer i kontakt med en rute, vil ruteobjektet kjøre en funksjon som fjerner ruten og avsløre en underliggende mønsterform. Hvis spilleren kjører over denne mønsterformen, vil den samme funksjonen kjøres, og fjerner da mønsterformen.



Figur 4.10: Spilleren ruller over en rute, og ruten forsvinner. Dette avslører en underliggende mønsterform, som igjen forsvinner når spilleren ruller over dèn.

Spillerens gravitasjon skifter når spilleren triller over kanten til en ny side. Det er lettest å forklare med en illustrasjon:



Figur 4.11: Tverrsnitt av kuben og gravitasjonsboksene rundt.

Dette er et todimensjonelt tverrsnitt som utelater to av sidene. Kuben er omringet av usynlige gravitasjonsbokser (illustrert i grått). Spilleren på venstre side befinner seg i en slik gravitasjonsboks (Left). Alle spillere inne i denne boksen har gravitasjon regnet ut på følgende måte:
Gravitasjonsboksens tredimensjonelle posisjon (-50, 0, 0), normalisert, flippet, og ganget med en gravitasjonsfaktor. Hvis gravitasjonsfaktoren er 10:

- (-50, 0, 0) *normalisert* = (-1, 0, 0).
- (-1, 0, 0) *flippet* = (1, 0, 0).
- (1, 0, 0) *ganger gravitasjonsfaktoren* = (10, 0, 0).

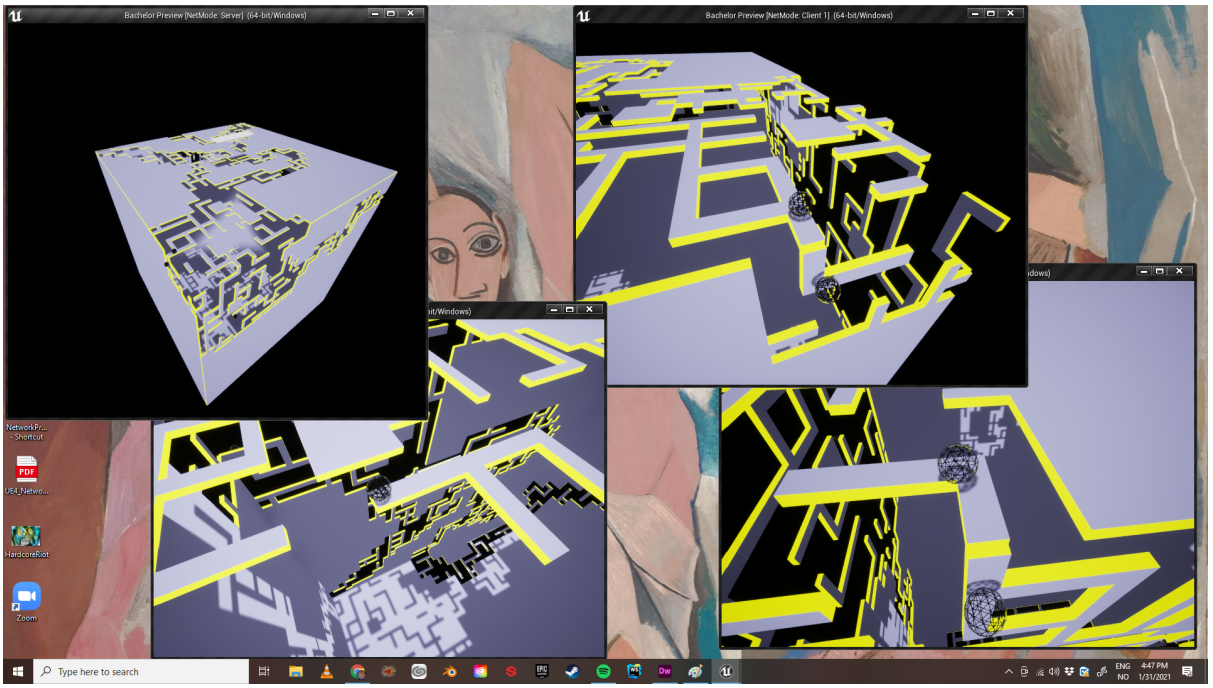
Gravitasjonen for denne spilleren blir dermed 10 i x-retning.

Den nederste spilleren får altså gravitasjon (0, 10, 0), som gir gravitasjon 10 i y-retning. Gravitasjon 10 betyr at spilleren få påført en akselleresjon i den retningen lik 10m/s^2.

Uke 9 - 13: Installere kjernen på en dedikert server

Spillet skal ha nettverksbasert flerspillerfunksjonalitet. Hver spiller kobles opp mot en dedikert server, der de spiller mot andre spillere. Det har ikke noe enkeltspillermodus.

På grunn av kompleksiteten rundt server/klient kommunikasjon må spillet utvikles med hensyn til dette helt fra begynnelsen. I Unreal Engine er det mulig å teste koden underveis på en simulert (lokal) server, med et valgt antall klienter.

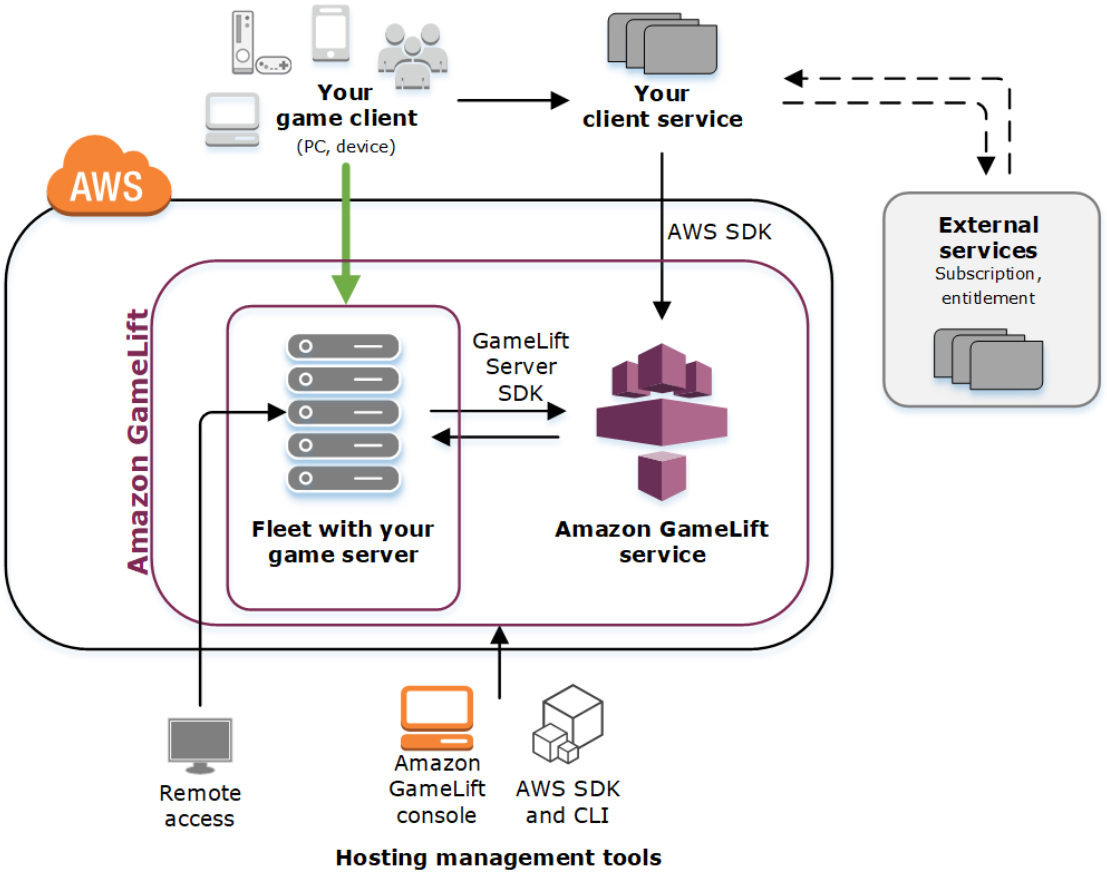


Figur 5.1: Dette er et bilde fra simulert server/klient kommunikasjon i spillmotoren. En server opprettes lokalt på maskinen, sammen med tre tilkoblede klienter. Bildet oppe til venstre er fra serverens synspunkt, de tre andre er klienter.

Unreal Engine kan simulere server/klient kommunikasjon, men det er likevel viktig å få testet programvaren på en ekte server tidlig. Dette for å avdekke problemer i koden som simuleringen ikke kan oppdage.

Dedikert servertjeneste: Amazon Web Services Gamelift

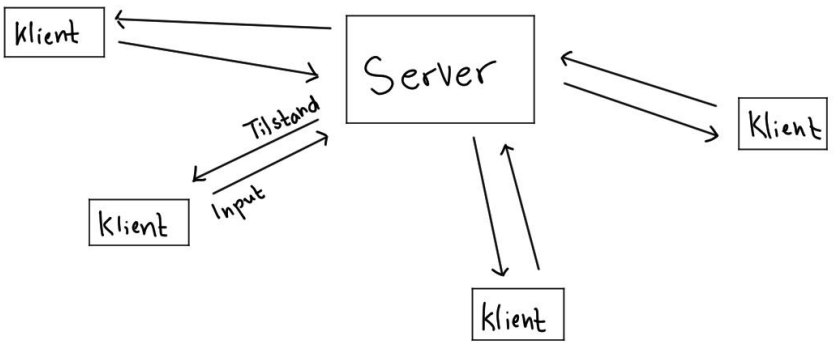
Det finnes en del dedikerte servertjenester å velge mellom på markedet, men få laget spesielt for spill, og enda færre kompatibelt med Unreal Engine. Det var vanskelig å finne et bedre alternativ enn Amazons Gamelift tjeneste, som blir brukt av flere ledende flerspillertitler, blandt annet Fortnite.



Figur 5.2: Amazon gamelift oversikt.

Dette bildet fra Amazon Gamelifts online dokumentasjon (Amazon Web Services 2021) viser hvordan det dedikerte serversystemet fungerer. Det kan virke komplisert, men kan i stor grad forenkles til:

- Serverversjonen av spillet er installert på en maskin hos Amazon (det de kaller en "fleet").
- En spiller (hvor som helst i verden) starter klientversjonen av spillet, som inneholder kode for å koble seg direkte opp til serveren ("fleet'en").
- Serveren ("fleet'en") kobles mot et antall spillere, og fungerer på den måten som et knutepunkt for alle deltagerene.



Figur 5.3: Forenklet illustrasjon av server/klient kommunikasjonen.

Spillet må altså deles opp i to ulike versjoner: En som installeres på serveren, og en som installeres hos klientene.

Serverversjonen:
Amazon Gamelift har et eget SDK (Software Development Kit) for serverversjonen til spill laget med Unreal Engine, med kode for å opprette en ny spillsesjon, ta imot forespørsler fra klienter om å få koble til denne spillsesjonen, stoppe en spillsesjon, og et antall andre funksjoner og variabler. SDK'et er tilgjengelig i C++ format, og kan relativt enkelt legges til spillkoden i form av en plugg. Man laster ned en mappe med filer, som man kopierer og limer inn i prosjektet, og så fyller man ut denne med egen kode etter behov.

Klientversjonen:
Amazon Gamelift har ingen enkel SDK løsning for klientversjonen til spill laget med Unreal Engine. Her finnes det kun et C++ bibliotek som dekker over alle mulighetene man har med Gamelift, uansett spillmotor. Det finnes ingen informasjon om hvilke filer man skal bruke, og det finnes heller ingen bruksanvisning for hvordan man innlemmer koden i Unreal Engine. Dette var en tidstyv. Det tok lang tid å lese dokumentasjon, teste, feile, teste og feile igjen. Til slutt falt valget på å kjøpe en tredjepartsplugg til Unreal Engine som kan installeres like enkelt som server SDK'et fra Amazon. Hvorfor Amazon ikke selv tilbyr en slik plugg til klientkoden er vanskelig å si. Pluggen kostet kun et par hundre kroner, men sparte prosjektet for et par hundre arbeidstimer (*Eeldev 2021*).

Første test

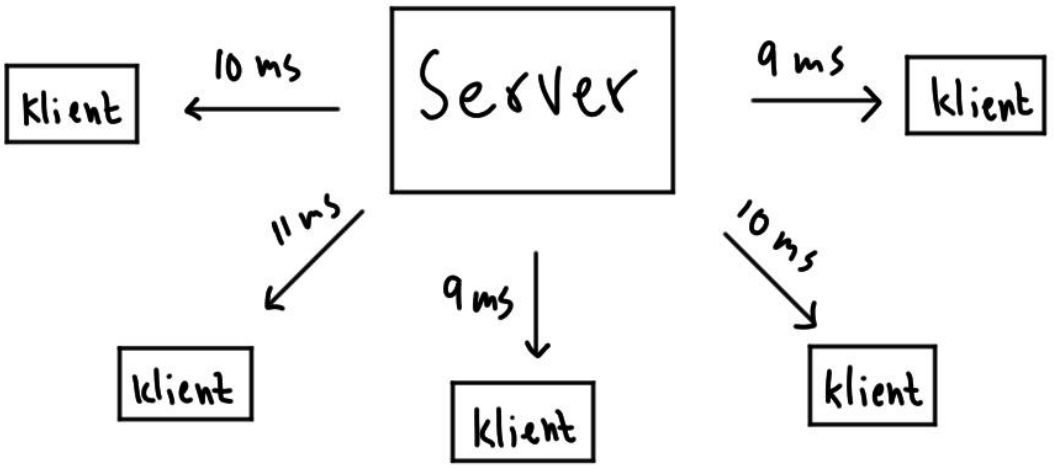
All den nødvendige koden ble implementert og serverversjonen installert på serveren. En klient ble kjørt, og koblet seg riktig opp mot serveren. Den første reelle testen av kjernefunksjonaliteten på en dedikert server viste at kuben, rutene og mønsteret fungerte akkurat som det skulle. Testen avslørte også et stort problem: Spilleravataren (kulen) gikk veldig hakkete!

Problemet:
Kulens bevegelser lener seg på fysikkmotoren. Hvis man holder spaken på spillkontrollen en retning, vil kulen få påført en kraft og rotasjon i den retningen man holder. Serveren og klienten kalkulerer denne fysikken hver for seg. Når disse kalkulasjonene blir ulike, vil serveren, som har siste ordet i hva som skjer til enhver tid i spillet, sende sitt svar av utregningen til klienten og "rette feilen" hos klienten. Dette skjer cirka hvert tidels sekund, og fører til hakkete og kløne te bevegelser.
Den første ideen til løsning: ikke utfør fysikkutregningen hos klienten i det hele tatt, men send kontrollretningen til serveren og utfør utregningen kun der. Klienten får returnert den resulterende posisjonen og rotasjonen som svar fra serveren. Dette ble prøvd, men på grunn av forsinkelser i signalet førte dette til svært luggete kontroll. Spillet er blandt annet basert på balanse og presisjon, og krever presis kontroll; lugging i kontrollen er ikke bare frustrerende, men avgjørende.

Dette belyser et klassisk problem rundt nettbasert flerspillerfunksjonalitet: Hvordan er det mulig å lage et spill uten forsinkelser, når det er fysisk umulig å fjerne selve forsinkelsen i signalet? Tenk deg:

Spiller A holder en retning på kontrolleren for å bevege seg. Hvis spillet er basert på presisjon bør avataren bevege seg momentant, uten noen som helst forsinkelse. Men alle de andre spillerene på samme server skal også se posisjonen til spiller A. Hvis posisjonering er en viktig del av spillet (som det er i alle skytespill, ballspill, kontaktsportspill etc) må posisjonen til enhver spiller være synkronisert over alle klientene. Dette leder til et dilemma: Forsinket bevegelse versus usykronisert bevegelse.

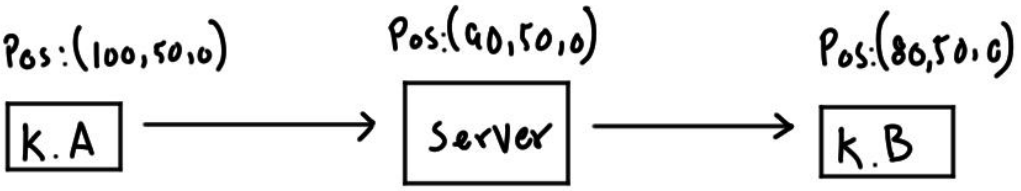
Forsinket bevegelse:
For å ha best mulig synkronisering av posisjon, er det best at serveren håndterer all bevegelse. Hvis spilleren ikke kan bevege seg på sin egen klient vil det ikke oppstå noen asynkronisering av posisjon mellom klienter og server. Synkroniseringen av spillerens posisjon asynkroniseres kun av *forskjellen* i tiden det tar for signalet å nå hver enkelt klient.



Figur 5.4: Klienter mottar samme signalet på ulike tidspunkt.

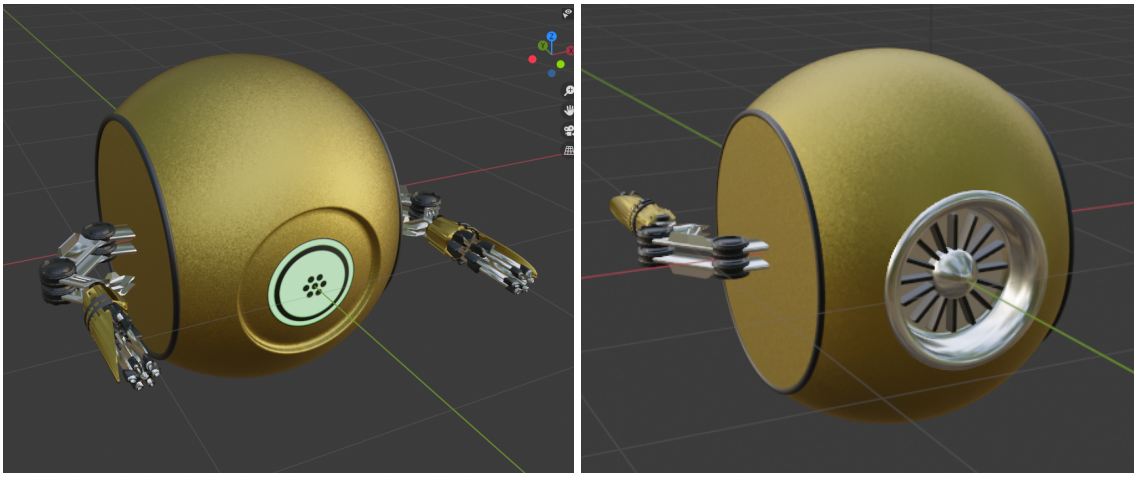
Her vil for eksempel den maksimale asynkroniseringen være 2 ms, som er neglisjerbart. Problemet med dette er at spilleren må først sende et ønske om å bevege seg til serveren. Hvis det tar 10 ms å sende dette ønsket til serveren, og siden 10 ms å få returnert resultatet, vil spilleren oppleve en 20 ms forsinkelse i bevegelsen. Signalforsinkelse mellom en klient og dedikert server ligger ofte mellom 10 og 50 ms, og dette kan føre til hakking, lugging og generelt dårlig respons.

Usynkronisert posisjonering:
Hvis man derimot utfører bevegelsen lokalt hos klienten, og siden sender posisjonen til serveren, som siden sender denne posisjonen til resten av klientene, vil man oppleve en asynkronisering av posisjonen til spillerene.



Figur 5.5: Her er spilleren på posisjon (100, 50, 0) lokalt (klient A), men serveren har ikke blitt oppdatert enda. På serveren er spilleren fremdeles på posisjon (90, 50, 0). Denne felles posisjonen på serveren har, igjen, ikke blitt replikert til klient B enda, og personen som spiller på klient B tror spilleren på klient A er på posisjon (80, 50, 0). Hvis spilleren på klient B sikter på spilleren fra klient A og skyter, treffer han?

Løsningen:
Epic Games, som står bak Unreal Engine, har jobbet med denne problemstillingen siden Unreal Tournament i 1999. Kort sagt finnes det ingen magisk formel som fjerner problemet, kun en sekk med triks. De har faktisk gjort et av triksene sine tilgjengelig i form av en "movement component" i spillmotoren, som man enkelt kan legge til i koden. Denne komponenten bruker blandt annet prediksjon- og kompensasjonsalgoritmer for å gi umiddelbar bevegelse lokalt hos klienten, og samtidig synkronisert posisjonering på serveren! Desverre støtter ikke komponenten fysikkbasert bevegelse. Bevegelsen må være lineær.
Dette førte til at spilleravataren ikke lenger kunne være en rullende kule. Løsningen ble en flygende droide!



Figur 5.6: Spilleravataren.

Det var ikke så lett å finne på spillerfunksjonalitet når avataren var en rullende kule. En flygende droide gav opphav til to nye ideer:

- *Armer som kan skyte.*
- *Jetmotor som gir ekstra fart.*

Det ble også lettere å lage en interessant 3D modell siden avataren ikke lenger måtte være en helt rund kule.

Uke 13 - 17: Tidlig testing, stabilisering av kjerne, planlegge utvidet funksjonalitet.

Spillet har fått en fungerende kjerne, og koden for server/klient kommunikasjon fra en dedikert Amazon server er implementert. Spillerene har følgende funksjonalitet:

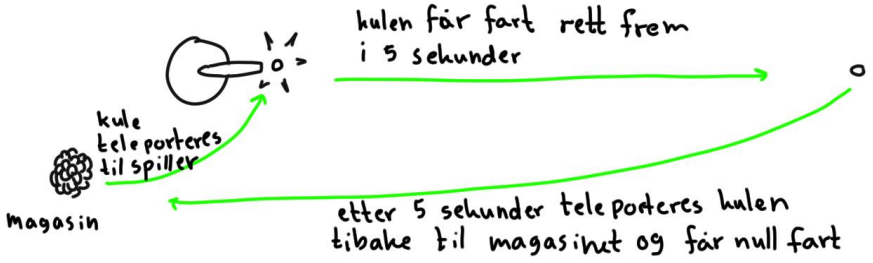
- *Bevegelse i alle retninger med L-spaken.*
- *Rotasjon av kamera med R-spaken.*
- *Zoom kamera inn/ut med D-pad opp/ned.*
- *Skyte laser med R-trigger.*
- *Boost (ekstra fart) med L-trigger.*
- *Skjold mot laser med R-shoulder.*

Utvidet funksjonalitet

Det siste punktet, skjold mot laser, har blitt lagt til jamfør et brukerønske etter brukertesting. Brukerens ønske kom tidlig i brukerens første test, og det føltes som en åpenbar utvidelse av spillerfunksjonaliteten. Det var også en enkel utvidelse å legge til. Det tok ikke lang tid å utvikle den (noen timer), og den la lite ekstra arbeid til serveren.

Når ekstra funksjonalitet skal legges til, må det taes hensyn til hvor mye ekstra arbeid funksjonen legger på serveren. Hvis det er åtte spillere på serveren, skal informasjon om denne funksjonen sendes til og mottas fra åtte klienter. Serveren skal også behandle funksjonen med prosessering og lagring i nærminne for opp til åtte instanser samtidig. Hvis funksjonen oppretter nye objekter i minnet (spawn object) kan minnet fort fylles opp. Det er også krevende for serverprosessoren å oprette og fjerne objekter fra minnet fortløpende.

Laserkuler:
Laserkuler er potensielt krevende for en server. Spilleren trykker på skyteknappen, og en laserkule blir opprettet. For ikke å fylle opp minnet må laserkulen fjernes etter bruk. I dette spillet kan èn spiller skyte en kule hvert 0.15 sekund. Dette gir rundt 7 kuler i sekundet per spiller. Kulen forsvinner etter 5 sekunder. Da eksisterer det potensielt 7 * 5 = 35 kuler per spiller til enhver tid. Med 8 spillere gir det 35 * 8 = 280 kuler.
280 kuler i minnet om gangen går helt fint, det er ikke spesielt krevende. Det som *er* krevende for serveren er å hele tiden fjerne kuler fra minnet, og opprette og finne ledig plass til *nye* kuler. For at serveren skal slippe å fortløpende opprette og fjerne kuler for alle spillerene, opprettes det istedenfor et sett med 50 kuler til hver spiller ved begynnelsen av hver omgang (35 pluss en buffer på 15). Disse kulene er usynlige, står helt i ro, og er plassert på posisjon (0, 0, 0). Når en spiller trykker på skyteknappen, teleporteres en av disse kulene til spilleren, får fart og rotasjon i riktig retning, og blir synlig. Når kulen er ferdig med jobben sin teleporteres den tilbake til posisjon (0, 0, 0), bevegelsen blir fryst, og kulen blir usynlig igjen. Kulene i settet (magasinet) tar tur om å aktiveres.



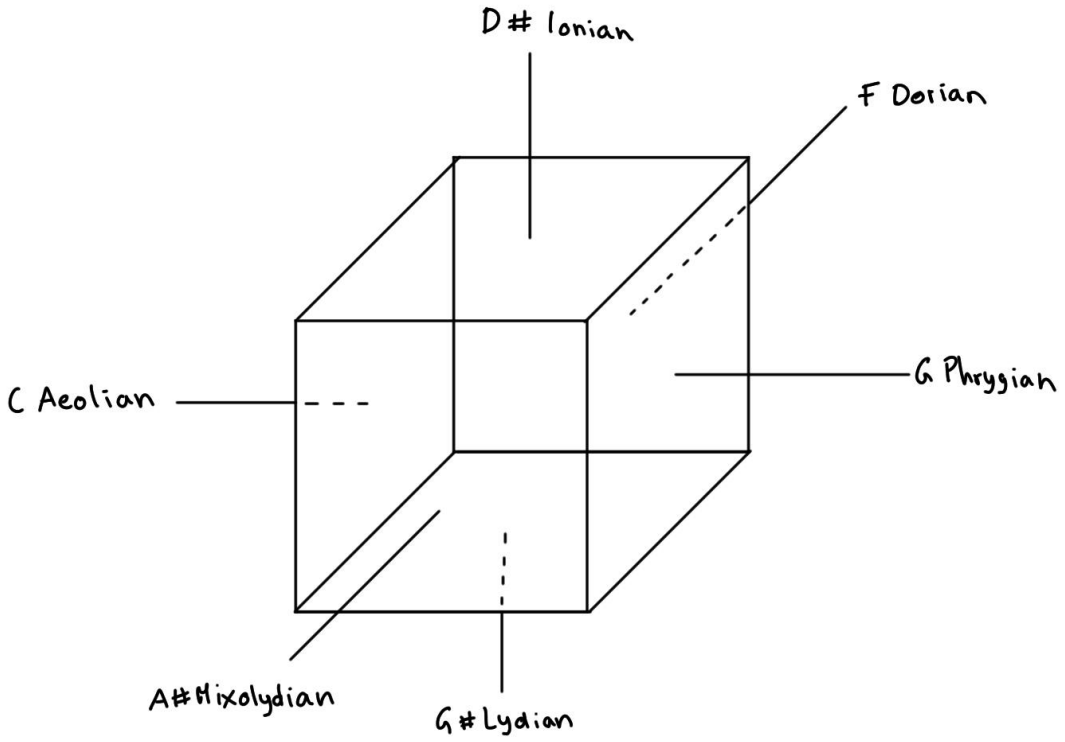
Det kan, per spiller, eksistere 35 kuler på en gang, men magasinet til spilleren holder 50 kuler, så en kule kan aldri reaktiveres før den er deaktivert og har returnert tilbake til magasinet. Alle kulene eksisterer hele tiden på samme plass i minnet, og oppdaterer bare variablene sine. På denne måten slipper serveren å opprette og fjerne kuler i minnet.

Grafikk

All grafisk behandling på serveren er unødvendig, siden ingen skal se det som serveren ser. For å gi serveren minst mulig arbeid har alle krevende kosmetiske funksjoner blitt fjernet fra serverversjonen. Klientene får den grafiske arbeidsbyrden. Det samme gjelder musikk og lydeffekter.

Musikk og lydeffekter

Musikken er orientert rundt D# major skalaen, dvs D# Ionian. Hver side av kuben har en egen modal skala av D# major forbundet med seg, slik at når spilleren skifter side på kuben, transponerer musikken til den skalaen. Skalaene er: D# Ionian, F Dorian, G Phrygian, G# Lydian, A# Mixolydian og C Aeolian. Musikken er hovedsaklig tepper av den grunnleggende 7'er akkorden for hver modale skala, med litt av de modale kjennetegnene drysset inn (som b2 i Phrygian og #4 i lydian).



Figur 6.1: Alle sidene av kuben har hver sin modale skala forbundet med seg.

Lyden når du kjører over en full rute (som gir deg energi) er en tilfeldig note i D# major skalaen, slik at uansett hvilken siden av kuben du er på vil noten være relativ til skalaen som spilles i musikken - men følelsen forandrer seg.
Musikken og alle lydene er laget med samme analoge synth. Dette gir lydbildet en samhørighet fremfor å bruke mange forskjellige lydkilder.

Uke 17 - 21: Iterativ utvikling, med intensiv testing og utviding av funksjonalitet i forhold til tilbakemeldinger fra testere.

Andre test - 8 spillere samtidig

I uke 18 ble spillet testet med 8 spillere samtidig (studenter fra 1. klasse data på HVL). Hovedformålet med testen var å sjekke om serveren taklet så mange spillere samtidig. Det er den billigste serverløsningen som blir brukt (som gir 125 gratis timer i måneden). Denne leverer 2 prosessorkjerner og 4 gigabyte RAM, og opp til 10 gigabit netverksytelse.
Testen gikk helt perfekt. Det var ingen ytelsesproblemer fra serveren, og heller ingen nettverksrelaterte problemer. Det er all grunn til å tro at det kan være enda flere spillere på serveren, kanskje 16 eller 24 spillere samtidig. Dette er ikke lenger en teknisk problemstilling, men lener gjerne mer mot en designmessig vurdering - hvor mange spillere samtidig gir best underholdningskvalitet? 24 spillere kan bli for mye; men det er vanskelig å vite før man har testet det. Resultatet av testen var suksess.

Uventet designproblematikk: Boostfunksjonen

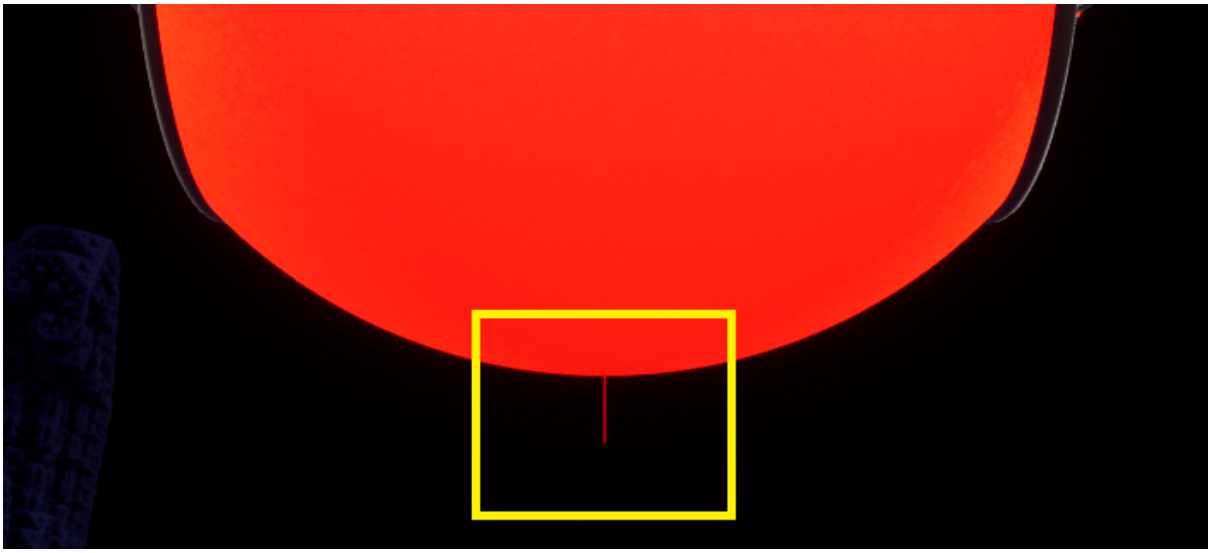
Det er en designmessig feil i spillet som må fikses før ny funksjonalitet legges til. Hovedoppgaven til boostfunksjonen er å gi spilleren ekstra fart fremmover når den aktiveres, men den ble forandret slik at spilleren flyr i tillegg. Dette gir spilleren mulighet til å fly over hullene (som man taper når man faller ned i). Den koster for lite energi å aktivere. Dette resulterer i at alle spillerene bare flyr endeløst rund på kuben, og det ødelegger omtrent alle navigeringsutfordringene forbundet med at kuben går i oppløsning.
I utgangspunktet skulle spillerene bruke en hoppefunksjon for å hoppe over hull, men det var en del problematikk rundt den på grunn av uoverenstemmelse i fysikkutregninger mellom server og klienter. For å løse dette problemet ble flyging når man booster lagt til som en erstatning for hopping. Istedenfor å kunne hoppe over hull, flyr man over de ved å booste. Desverre ødela dette for resten av spilldesignet. Hoppfunksjonaliteten må tilbake igjen i spillet!

Da må man lage en hoppefunksjon som ikke trenger å bruke fysikkmotoren. For øyeblikket er gravitasjonen en fysikkutregning som utføres på serveren.
For å unngå forvirring: tidligere i rapporten ble det nevnt at fysikkutregninger ble fjernet fra spilldesignet på grunn av uenigheter mellom serveren og klienter. Spillerene var opprinnelig en kule som rullet, men dette designet ble fjernet til fordel for en avatar som ikke var avhengig av fysikkutregninger - en svevende droide. Men, for at spilleren skulle kunne “falle inn i midten” måtte fremdeles en eller annen form for gravitasjonskraft eksistere. Løsningen var en fysikkmotorbasert kraft som dyttet spilleren. Denne kraften ble utført utelukkende på serveren. Dette førte til en liten forsinkelse mellom serveren og klienten, men siden denne kraften ikke trengte å være like umiddelbar (som for eksempel bevegelse) var dette en tilfredsstillende løsning. Gravitasjonskraft behøver ikke umiddelbarhet siden fartsendringen starter på null, og bygges opp over tid i forhold til akselerasjonen.
Hvis man vil legge til en hoppfunksjon derimot, så er denne forsinkelsen uakseptabel. Det er ikke tilfredsstillende å utføre hoppfunksjonen på serveren. Den må ha null forsinkelse. Når man trykker på knappen må hoppet skje umiddelbart.

Løsning: lage falsk gravitasjon med movement component

Den nye avataren implementerte en “movement component”, en valgfri funksjonalitet fra spillmotoren Unreal Engine. Denne komponenten bruker algoritmer som Epic Games har utviklet i over 20 år, for å bevege spilleren med null forsinkelse hos klienten, men samtidig holde posisjonen oppdatert på serveren. Komponenten inkluderer desverre ikke fysikkbaserte bevegelser uten forsinkelser, det støtter kun “lineære” bevegelser (men tilbyr interpolering mellom verdier for å etterligne fysiske interaksjoner). For å lage en hoppefunksjon må man altså noe som ser ut som hopping med slike lineære, interpolerte bevegelser.

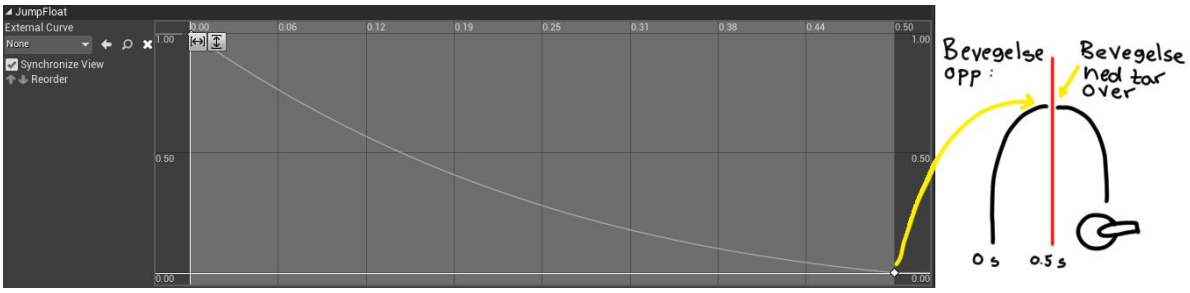
Først ble den fysikkbaserte gravitasjonen byttet ut. For å lage en tilnærmet fysikkbasert gravitasjon ble det lagt til en konstant bevegelse i samme retning som gravitasjonsvektoren peker. Denne bevegelsen blir aktivert når avataren ikke er i kontakt med bakken.
En såkalt “line trace” blir sendt fra midten av avataren og i samme retning som gravitasjonsvektoren. Hvis denne linjen kolliderer med bakken, setter den en boolsk verdi “onGround” til “true”. Hvis denne verdien er “false”, vil den falske gravitasjonsbevegelsen aktiveres.



Figur 7.1: En linje blir sendt ut fra avataren og detekterer andre objekter. Hvis det andre objektet er kuben, settes “onGround” til “true”.

Problemet med denne falske gravitasjonen er at den ikke bygger opp moment. Den akselererer ikke, den påfører kun en konstant fart i en retning. Bevegelseskomponenten har heldigvis innstillinger for “smoothing” i fartsendringen slik at endringen ikke er umiddelbar. Farten har en myk overgang fra en retning til en annen, og hvor myk denne er kan settes med en interpoleringsvariabel. Dette kan gi en illusjon av at bevegelsen er fysikkbasert. Når spilleren ikke har bakke under seg blir det et sekunds myk overgang til en maks fallhastighet. I motsetning til ekte gravitasjon (og den funksjonen som allerede var implementert) slutter den å akselerere veldig tidlig. Ekte gravitasjon akslererer lenger. Det er ikke den mest overbevisende gravitasjonseffekten, men det er egentlig ikke så farlig. Det er ikke så lenge spilleren faller inn mot midten uansett.

Med denne falske gravitasjonen var det enkelt å lage hoppfunksjonalitet med umiddelbar reaksjon. Hoppknappen påfører en bevegelse i motsatt retning av gravitasjonsvektoren over en viss tid. For å simulere en ulineær minkende hastighet måtte denne bevegelsen få en kurvet faktor i tillegg. Kurvebrattheten er ikke basert på ekte fysikk, men det som føltes best. I spillmotoren finnes det en tidsgrafkomponent som lar utvikleren tegne en kurve, hvor x-retningen er tid (fra aktivering) og y-retningen er returverdi.



Figur 7.2: Tidsgrafkomponenten til Unreal Engine. Ved aktivering (x = 0 sekunder) er returverdien maksimal (y = 1), og etter et halvt sekund (x = 0.5 sekunder) er returverdien null (y = 0).

Mens hoppfunksjonen sender spilleren opp, er den falske gravitasjonen deaktivert. Når spilleren har nådd toppen (etter 0.5 sekunder), aktiveres den falske gravitasjonen igjen til hen når bakken. Hvis spilleren ikke treffer bakken, fortsetter hen å falle inn i midten av kuben og taper.

Uke 21 - 23: Ferdigstilling av prototype for visning til eventuelle interessenter, og plan videre.

Den nye hoppfunksjonaliteten har blitt testet, og nå føles prototypen bra. Prototypen er installert på serveren, og kan aktiveres på under fem minutter. Hvis noen interessenter vil teste spillet kan de få tilsendt en nedlastningslink og serveren aktiveres. Prototypen er dermed definert som ferdigstilt, og produktdelen av bacheloroppgaven er avsluttet.

Plan videre

Først og fremst vil spillet utvides til 16 (muligens 24) spillere samtidig. Utover det må følgende gjøres å lage et fullverdig kommersielt produkt av prototypen:

- *Lage mer spillerfunksjonalitet*. Hopping, boosting, skyting og skjold gir ikke spillet nok dybde for langvarig spillerdedikering. Her må ytterligere funksjonalitet utforskes, testes, balanseres og finpusses for å oppnå et profesjonelt produkt.

- *Lage mer innhold*. Spilleren bør kunne velge mellom ulike droider, gjerne med ulike ferdigheter og personligheter som passer spillerens spillestil. Det trengs også flere bakgrunner (fraktale mønster). Det hadde vært kult med bakgrunner i ulike farger, med matchende farge på kuben. Rød bakgrunn og rød kube, Grønn bakgrunn og grønn kube, etc.

- *Bedre startmeny*. Spilleren må få kontroll over maksimal bildefrekvens og andre prestasjonsrelaterte innstillinger. Spilleren må også kunne velge sitt eget kontrolloppsett.

- *Meny under spillets gang*. Spilleren må ha tilgang til en meny hvor hen kan avslutte spillet, forlate en pågående kamp, etc.

- *Chatfunksjon*. Det hadde vært fint om spillerene kunne kommunisere via en chatfunksjon, slik som i Rocket League.

- *Matchmaking*. Amazon Web Services har funksjonalitet for et køsystem som setter spillere opp mot hverandre basert på en database av lagrede spillervariabler. Her kan man sette sammen kamper av spillere med likt ferdighetsnivå, nær geografisk lokasjon, etc.

- *Flere baner*. Det kunne vært kult med alternativer til kuben. En sfære, en pyramide og en tolvsidet dodekaeder er eksempler på mulige alternativer.

- *Flere typer spillmodus*. For eksempel kunne det vært gøy med lagspill. Kanskje noen spillmodus kunne hatt egen spillerfunksjonalitet som ingen av de andre har? Dette kunne man utforsket mer mot slutten, og kanskje noen modus kunne inkludert spillerfunksjonalitet som ble for “vill” for normalt spillmodus.

Evaluering

God underveisevaluering av et spill kan være vanskelig å oppnå. Hva utgjør egentlig et bra spill? Hvilke egenskaper betegner en suksess?

Først og fremst har man den helt selvsagte egenskapen: Spillet kræsjer ikke og kjører problemfritt. Utover det har man noen typiske egenskaper som alle anser å være viktig:

- *God kontroll/lav forsinkelse*: Umiddelbar respons er universelt prissatt. Det betyr ikke at det ikke kan interpoleres langsomt mellom bevegelser, men at denne interpoleringen isåfall starter med en gang spilleren gir kommando. Spilleren er “oppslukt i spillet”; forsinkelser ødelegger følelsen av å være inne i spillet, og gjør det generelt vanskelig å kontrollere avataren.

- *Pen grafikk og lyd*: Alle spillere setter pris på visuell og hørbar stimuli. Hva som kjennetegner “bra” visuell og hørbar stimuli er ikke så lett å definere; her må man nesten spørre testerene.

- *Tydelig interface*: Det skal være enkelt å forstå menyene, og all spillerstatistikk (som energinivå og tabellplassering) skal være lett å lese under spillets gang.

Den vanskeligste spillegenskapen å teste er underholdningsverdien. I kapittel 4: “Understanding your player” i boken “Fundamentals of game design” beskriver Ernest Adams de esoteriske spillegenskapene som tiltrekker spillere.

Jason VanderBerghe, kreativ leder hos Ubisoft, la i 2012 frem en tilnærmingsmåte for å forstå ulike typer spillere i forhold til psykologiens "femfaktormodell" (OCEAN-modellen): "Based on his understanding of the Five Factor Model, VanderBerghe proposed that we play games to satisfy the same motivations that we feel in real life, and this is particularly true if we are unable to satisfy them in real life. Play gives us an outlet" (Adams 2014 s.82). VanderBerghe korrelerte de fem trekkene fra femfaktormodellen med fem domener i spill som kan oppfylle de - og dermed sees som ettertraktede egenskaper i et spill:

- *Novelty*. Korrelerer med "Openness to experience" (Ocean): på en side av spekteret finner man spillere som søker originalitet i spill, med mye variasjon og uventede elementer. På motsatt side av spekteret finner man spillere som søker familiaritet.

- *Challenge*. VanderBerghe korrelerte søken etter utfordringer i spill med "Conscientiousness" (oCean): på en side av spekteret spillere som liker å vinne intense konkurranser. På andre siden de som liker rolige spill med lav "straff", hvor man kan leke i en virtuell verden uten å testes kompetativt.

- *Stimulation*. Sosialt stimuli, korrelerer med "Extraversion" (ocEan): på en side spillere som liker interaksjon med andre menneskelige spillere (flerspiller spill), på andre siden spillere som foretrekker å utforske en virtuell verden alene.

- *Harmony*. Liker man sammarbeid eller konflikt? Korrelerer med "Agreeableness" (oceAn): på en side av spekteret har man spill som krever sammarbeid mellom spillere, på andre siden spill som oppfordrer til sabotasje for andre spillere.

- *Threat*. Følelsen av frykt, fare og andre "negative" følelser, korrelerer med "Neuroticism" (oceaN): på en side finner man spillere som aktivt søker slike følelser, på andre siden spillere som helst unngår de.

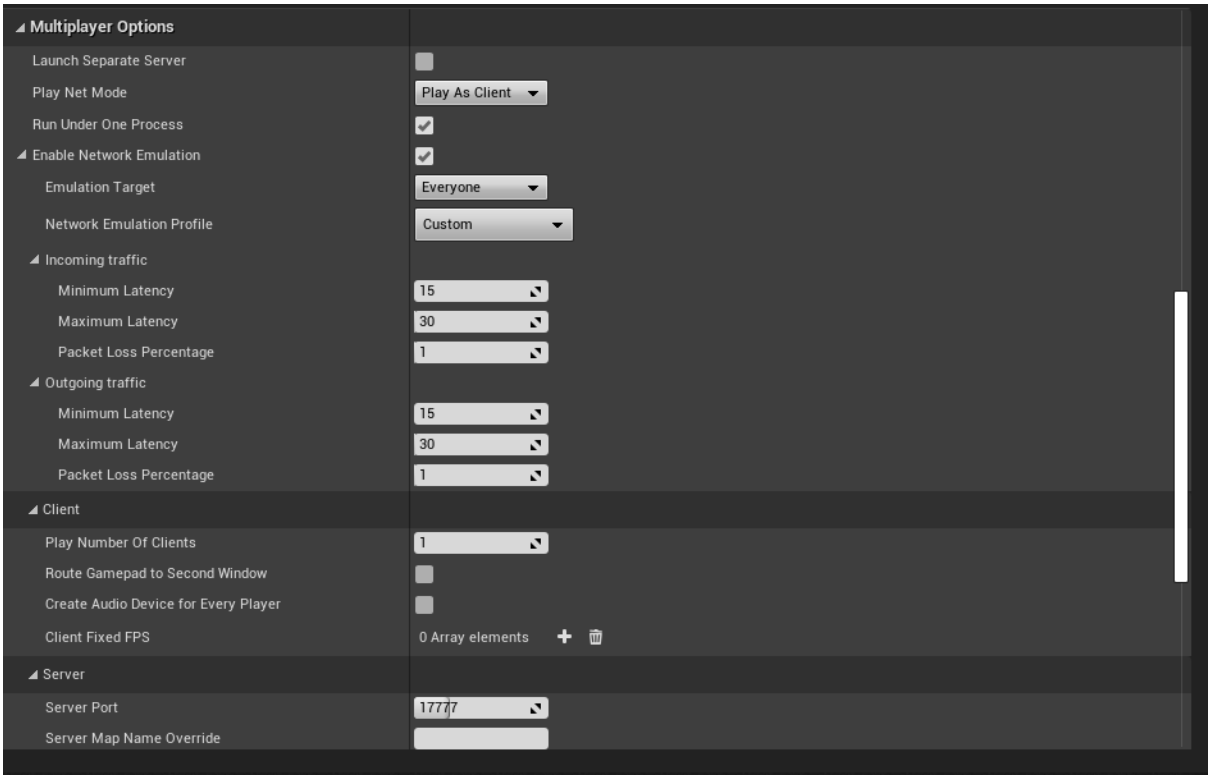
"These are not binary qualities. They are continuums, sliding scales. What's more, they don't describe what players *always* like; our moods change. VanderBerghe's point, and mine, is that by keeping these qualities of games in mind, we can decide as designers *how* we want to entertain them." (Adams 2014 s.83).

Alle disse egenskapene har blitt betraktet underveis. En av de vanskeligste delene av prosjektet har vært å evaluere spillet på egen hånd, før en fungerende serverversjon har blitt ferdigstilt og testing/evaluering med eksterne brukere har blitt tilgjengelig.

Evaluering på egen hånd

Istedenfor å basere designet på egne meninger om hva som er gøy, har inspirasjon fra andre vellykkede spill styrt denne første delen av prosessen. Selvsagt har det blitt brukt intuisjon, men denne har vært basert på (tilsynelatende) universelle kvaliteter.
I spillmotoren Unreal Engine er det mulig å trykke "play" for å teste koden helt fra prosjektet er opprettet. Kuben måtte programmeres først, og siden spilleren. Både kuben og spilleren har blitt programmert i tandem med testing internt i spillmotoren. Hva som skal ligge på serveren og hva som skal ligge hos klienten er ikke alltid like åpenbart; den interne testfunksjonaliteten er ekstremt viktig for å fortløpende sjekke om koden og

kommunikasjonen fungerer. Unreal Engine har dessuten innstillinger for å simulere ulike nettverksforbindelser. Dette er gull verdt; noen problemer er usynlige til man skrur opp forsinkelsen.



Figur 8.1: Unreal Engine har mange instillinger for den lokale nettverkssimuleringen som hjelper deg å finne problemer i koden tidlig. Spesielt nyttig er instillingene for minimum/maximum latency.

Kontrollen av spilleren og spillkonseptet i seg selv ble testet av andre før en serverversjon var ferdig, ved at venner kom på besøk og testet å spille gjennom spillmotorens interne testfunksjonalitet. Deres reaksjon rundt kontrollen og konseptet ble tatt i betraktning, men med kun èn spiller i spillet var det begrenset hvor mye av konseptet de egentlig forstod.

Initiell ekstern respons

Når en fungerende serverversjon endelig var bygget og installert på en ekstern Amazonserver var spillet klart for testing med flere spillere samtidig. I første omgang var formålet å luke ut eventuelle problemer i koden. Fungerte spillet i det hele tatt?
Før noen spillere ble invitert til testing ble spillet testet på den eksterne serveren med kun èn spiller: utvikleren selv. Som fortalt tidligere i rapporten floppet den opprinnelige rullende-kule ideen på grunn av uenigheter mellom server og klient. Det var ingen poeng i å teste denne versjonen med eksterne spillere. Spillet måtte repareres først.

Den nye flygende-droide ideen ble utviklet og testet på samme måte, og den fungerte som den skulle. Spillet var dermed klart for sin første flerspillertest! Tre eksterne testere ble invitert til test, og fikk tilsendt en nedlastningslenke. Siden denne første flerspillertesten har

ingen av problemene lenger vært av teknisk natur; men designmessige problemer har blitt blottlagt.

De første testene var egentlig for å sjekke det rent tekniske, men siden alt fungerte som det skulle ble alle tilbakemeldingene rundt spilldesignet. Alle forslag ble notert, og nesten alle ble implementert: Skjoldet er det beste eksempelet på tilbakemeldingsbasert funksjonalitet. Et annet eksempel er støtte for mus og tastatur.

Test med 8 spillere samtidig

Testen med 8 spillere avdekket det største problemet med spillet: flyging med boostfunksjonen. Denne ble fjernet og erstattet med en hoppefunksjon basert på falsk gravitasjon istedenfor. Dette ga endelig en menigsfull, problemfri prototype, som er lett å bygge videre på, og kan føres i mange ulike retninger designmessig.

Spørreskjema basert på VanderBerghes modell

Ved videre utvikling av en kommersiell utgivelse kan et spørreskjema basert på VanderBerghes modell være nyttig. Man kan først be testere om å evaluere spillets nåværende rangering i de fem domemene: Novelty, Challenge, Stimulation, Harmony og Threat. Siden kan man bruke denne informasjonen til å skreddersy for og markedsføre mot spillere som liker denne egenskapskombinasjonen. Det er ikke mulig å lage et spill for alle; et spill med høy “Novelty” faktor vil for eksempel ekskludere spillere som liker gjenkjennelighet.

Det kan også være en fordel å føre spillet mot en ønsket kombinasjon. Ved å analysere populære spill kan man finne en meningsfull kombinasjon som imøtekommer mange spillere. Selv om spillet allerede peker mot en spesifikk kombinasjon, er det fullt mulig å “dra” i disse skalarene ved videre utvikling. Noen egenskaper er derimot vanskeligere å forandre enn andre, for eksempel Stimulation. Det *er* mulig å lage et ensligspillerspill av dette (for eksempel ved å programmere datastyrte motstandere), men det er veldig krevende. Det er mulig å tilby “begge ender” av hvert domene for å tilrettelegge for absolutt alle, men dette er en stor jobb.
Rocket League tilbyr for eksempel spilleren å spille både mot andre mennesker og mot datastyrte motstandere. Men, Rocket League er en stor virksomhet med mange hundre ansatte.

Eksempel på spørreskjema

Svar med et tall mellom 1 og 10 (1 = helt uenig, 10 = helt enig):

- *Originalitet* (Novelty): Spillet er originalt.
- *Utfordring* (Challenge): Spillet er utfordrende.
- *Sosialt* (Stimulation): Spillet er sosialt.
- *Sammarbeid vs konflikt* (Harmony): Spillet krever høy grad av sammarbeid.
- *Fryktinngytende* (Threat): Spillet skremmer meg.

Dette spørreskjemaet ble levert til seks spilltestere etter en test søndag 16. mai, og fikk følgende resultater (avrundede gjennomsnittsverdier):

- *Originalitet: 8*
- *Utfordring: 8*
- *Sosialt: 7*
- *Sammarbeid: 2*
- *Fryktinngytende: 2*

De fleste anser konseptet som originalt, og spillet som sosialt og utfordrende. Det er lite sammarbeid (noen nevner spontane 2-mot-1 sammarbeid), og lite frykt (det fraktale bakgrunnsmønsteret, det svarte hullet i midten og bruddet med fysikklovene har en viss spenning forbundet med seg, men den er ikke høy nok til å ansees som skremmende).

Diskusjon

For å diskutere konsekvensen av valgene som ble tatt er det en fordel å oppsummere valgene som ble gjort på forhånd:

- Spillet skal inkludere mest mulig av pensum fra bachelorforløpet.
- Prosjektet skal gi studenten innsikt i replikering av interaktiv grafikk og lyd.
- Det skal ikke brukes noe eksternt innhold; alt skal lages internt.
- Ideen skal være simpel nok til at èn person kan ferdigstille en prototype innen gitt tidsrom.
- Det skal brukes smidig utvikling.

Det ble også gjort valg av ressurser:

- Spillmotor: Unreal Engine.
- Servertjeneste: Amazon Web Services Gamelift.
- 3D modelleringsmykvare: Blender.
- Tilbakemeldinger: Spilltestere

Noen valg ble også gjort underveis:

- Modellinstansiering.
- Avatar: fra kule til droide.

Spillet skal inkludere mest mulig av pensum fra bachelorforløpet.

Hva var konsekvensen av å bruke pensumet som en kreativ brønn? Det gjorde det lettere å finne løsninger på designkonsepter. Spesielt viktig var vektorkunnskapene fra matematikk og fysikk. Spillet har også fått en slags ingeniørmessig estetikk over seg, med konstruksjon og gravitasjon som hovedtema.
På den negative siden er gjerne dette noe som ikke faller i smak for mange. Spillet har blitt testet av spillere uten ingeniørbakgrunn, og disse stiller seg ofte likegyldig til disse temaene.

Vil neste spillprosjekt basere seg på dataingeniørpensumet? Denne kunnskapen vil jo ikke forsvinne, så den vil nok alltid influere fremtidige prosjekter; men det blir nok ikke lagt vekt på å inkludere mest mulig fra dette spesifikke kunnskapsområdet.

Prosjektet skal gi studenten innsikt i replikering av interaktiv grafikk og lyd.

Det er mange ganger vanskeligere å lage et flerspillerspill enn et ensligspillerspill, men replikering har blitt en forventet funksjonalitet i dagens spillmarked; kanskje ikke kun i spillmarkedet, men i mykvaremarkedet generelt. Konsekvensen av valget var at store deler av tiden måtte tilegnes planlegging og utvikling av replikeringslogikken. Det var dette som ga de største utfordringene, men samtidig de største kunnskapsgevinstene. Spillet hadde selvsagt tatt en helt annen retning om det ikke ble bygget for flerspillerfunksjonalitet, men mye av innholdet kunne fint vært det samme; for eksempel kunne kjernen også dannet grunnlaget for et enspillerspill.

Det skal ikke brukes noe eksternt innhold; alt skal lages internt.

For å maksimere kunnskapsgevinsten skulle det ikke brukes noe eksternt innhold. Dette ga en ekstra utfordring rundt planleggingen. Konsekvensen er et spill med mindre innhold, men med full kontroll og eierskap over det innholdet som er der. Spillsamfunnet har dessuten en forkjærlighet for utviklere som velger å utvikle alt på egen hånd.

Ideen skal være enkel nok til at èn person kan ferdigstille den innen gitt tidsrom.

Siden alt innholdet skulle lages internt ville det ikke være tid til å lage et fullstendig kommersielt produkt. Istedenfor ble det fokusert på å bygge en prototype som siden kan utvikles videre i flere forskjellige retninger. Konsekvensen av dette er at prototypen alene er uferdig. Interessenter må være interessert i å finansiere videre utvikling av produktet, da det ikke er et ferdig produkt.

Det skal brukes smidig utvikling.

Den grunnleggende kubeideen skulle fungere som en kjerne som kunne utvikles relativt raskt, og etterpå utvides med ytterligere spillerfunksjonalitet i samspill med tilbakemeldinger fra testere. Konsekvensen av å tilrettelegge for slik smidig utvikling var mindre forutsigbarhet, og konstant behov for spilltestere. Dette behovet for spilltestere ble en av de største utfordringene i utviklingsprosessen; spillet er laget for 8 spillere samtidig. Det måtte derfor testes av 8 spillere med jevne mellomrom for å få tilbakemeldinger om ny funksjonalitet. Ventetiden mellom testrundene ledet til mindre effektiv utvikling. Hvis det skal utvikles videre for 16 (eller 24!) spillere vil dette bli en enda større utfordring.

Konsekvenser av ressursvalg.

Den største konsekvensen av å kombinere Unreal Engine med Amazon Web Service Gamelift var at det var tilsynelatende vanskelig å implementere API'et for klientene i spillmotoren. Heldigvis hadde noen utviklet en plugg for Unreal Engine av API'et, som var veldig enkelt å implementere. Før denne ble tatt i bruk ble det brukt en del tid på å prøve å

implementere det på egen hånd. Denne tiden var ikke helt bortkastet; intens dokumentasjonslesing ga god innsikt i hvordan tjenesten fungerer.

Modellinstansiering.

For å muliggjøre konstruksjon av en kube bestående av tusenvis av ruter måtte det brukes en funksjonalitet fra spillmotoren for å instansiere modeller; dette for å unngå hundretusenvis av drawcalls i sekundet. Likevel legges det relativt stort trykk på maskinvaren, og maskiner uten dedikerte grafikkkort har hatt litt problemer med å kjøre spillet. Selv om de aller fleste brukerene har hatt rapportert veldig god bildefrekvens (over 100 bilder i sekundet), har bildefrekvenser helt ned i 15 bilder per sekund blitt rapportert på laptoper. Akkurat hva som skaper trykket er vanskelig å si; det finnes veldig lite informasjon rundt denne instansieringsfunksjonaliteten i dokumentasjonen. Det er mulig den anvender funksjonalitet fra Direct-X biblioteket som krever spesielle grafikkkortspesifikasjoner.
Dette spillet ser ikke ut som det burde ha noen særlige maskinvarekrav, men det er ikke lett å si om man kunne gjort noe annerledes; det er vanskelig å opprette tusenvis av individuelt justerbare modeller i en scene og samtidig oppnå høy ytelse. Det kan være man burde utvikle en spesialisert spillmotor for dette (hvis det ikke allerede finnes en spillmotor som håndterer dette bedre).

Avatar: fra kule til droide.

Den største forandringen skjedde etter første test: den rullende kulen som var avhengig av fysikkmotoren skapte kommunikasjonsproblemer mellom server og klient. Basert på gjenstående tilgjengelig tidsrom og manglende kunnskaper rundt emnet falt valget på å bytte ut avataren med noe som kunne implementere spillmotorens dedikerte bevegelsesløsning for nettverk istedenfor. Dette ga spilldesignet en endring i innfallsvinkelen som definitivt ga store konsekvenser for den iterative utviklingen av spilleregenskaper.
Det hadde vært utrolig interessant å studere og prøve å utvikle spillmotorens bevegelsesløsning selv, fra bunnen. Den inneholder over tjue års erfaring med nettverksbasert interaktivitetsproblematikk. Hvis man skal lage et nettverksbasert spill må man helst enten forstå innholdet i og kunne utvikle en sånn løsning selv, ellers bør man implementere og basere seg rundt en slik tilbudt løsning fra begynnelsen av.

Konklusjon

Hovedmålet med utviklingen var å ferdigstille en prototype klar for videre arbeid. Den kan taes i flere ulike retninger ut fra interessentens ønske. Videre utvikling må ta spesielt hensyn til problematikken rundt instansierte modeller - svake maskiner kan ikke målrettes for.
Når det gjelder læringsutbytte ble det oppnådd generell kompetanse innen replikert interaktiv grafikk og lyd - en kompetanse som tas med videre til masteroppgaven. Det ble også oppnådd dypere ekspertise rundt flerspillerfunksjonalitet i spillmotoren Unreal Engine, implementering av spill på en Amazon Web Service server, 3D modellering med modelleringsmykvaren Blender og smidig utvikling av spill jamfør tilbakemeldinger fra spilltestere.
Spillet står disponibelt for videre arbeid av fremtidige bachelorstudenter med erfaring i Unreal Engine og Amazon Web Services. Som forklart nærmere i siste seksjon under

kapittelet Detaljert Design (Uke 21 - 23), for å lage et fullverdig kommersielt produkt kan følgende punkter utføres:

- *Utvide til 16 spillere (eventuelt 24)*
- *Lage mer spillerfunksjonalitet*.
- *Lage mer innhold*.
- *Bedre startmeny*.
- *Meny under spillets gang*.
- *Chatfunksjon*.
- *Matchmaking*.
- *Flere baner*.
- *Flere typer spillmodus*.

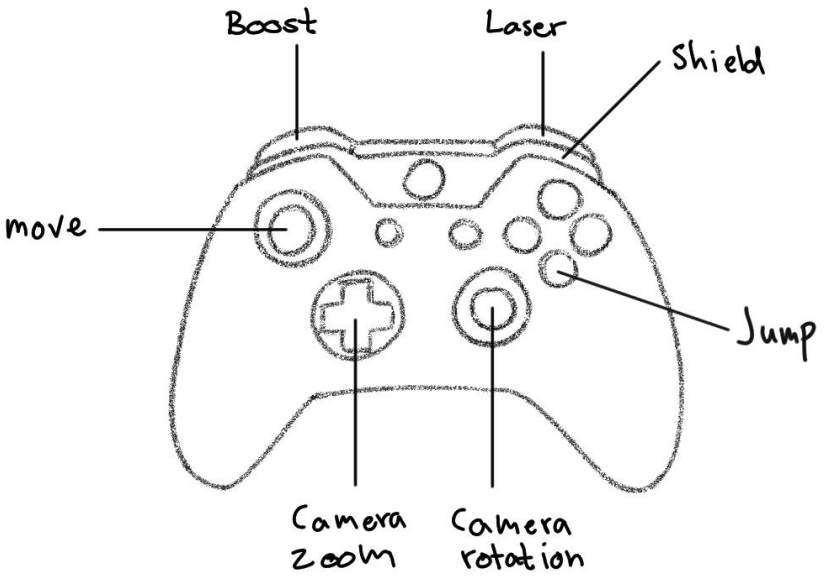
Utvikleren vil isåfall være tilgjengelig for spørsmål og rådføring. Koden er allerede kommentert, og prosjektet er ryddig og pent.

Referanseliste:

1. *Marius Irgens. 2021.*
   Bachelor blogg.
   http://mariusirgens.no/Bachelor/Bachelor.html
2. *Oslo Economics. 2018.*
   “Den norske spillbransjen - Utredning for kulturdepartementet.”
   https://osloeconomics.no/wp-content/uploads/Den-norske-spillbransjen.pdf
3. *Broadbandsearch. 2020.*
   “Online gaming statistics - 45 facts that will blow you away.”
   https://www.broadbandsearch.net/blog/online-gaming-statistics
4. *NewZoo. 2021.*
   “Most poular PC games | Global.”
   https://newzoo.com/insights/rankings/top-20-pc-games/
5. *Rocket League. 2021.*
   Hjemmeside.
   https://www.rocketleague.com/
6. *Fortnite. 2021.*
   Hjemmeside.
   https://www.epicgames.com/fortnite/
7. *Ernest Adams. 2014.*
   “Fundamentals of Game Design.”
8. *Amazon Web Services Gamelift. 2021.*
   “Amazon Gamelift Documentation.”
   https://docs.aws.amazon.com/gamelift/index.html
9. *EelDev. 2021.*
   “AWS Client Plugin.”
   https://eeldev.com/index.php/awscore-gamelift/

Appendix

Manual

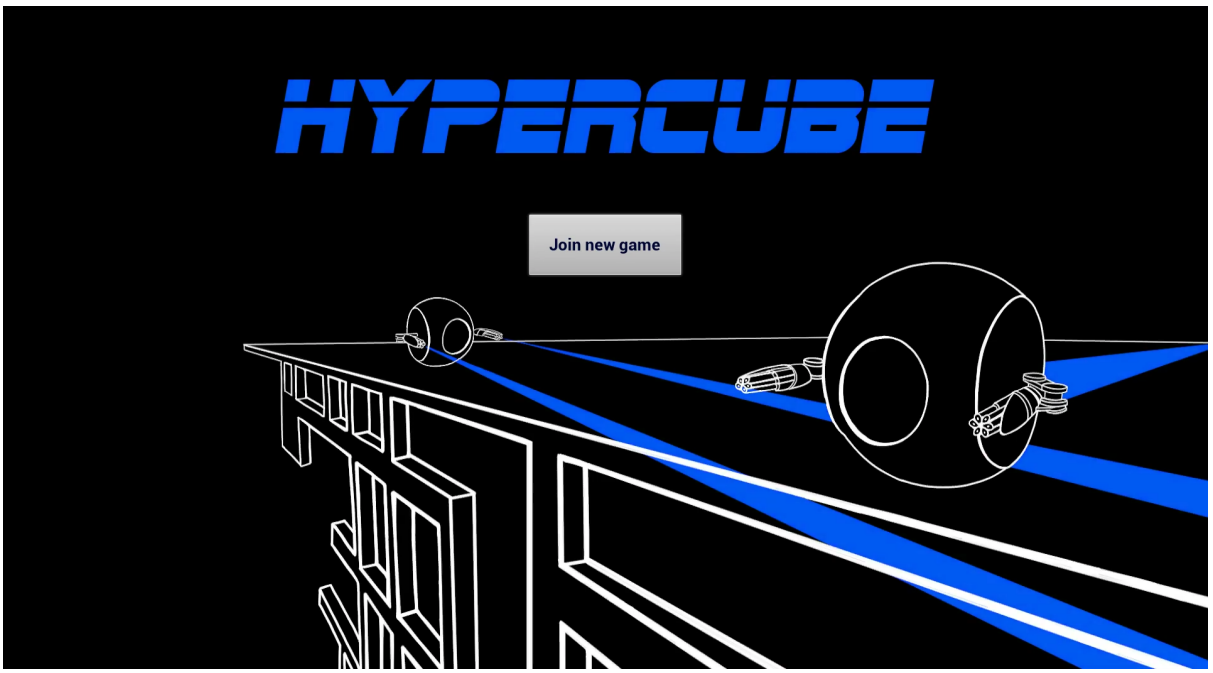


Move: L-Stick
Camera rotation: R-Stick
Camera zoom: D-Pad up/down
Boost: Left Trigger
Laser: Right Trigger
Shield: Right Shoulder
Jump: A button

Det er også mulig å bruke ***mus og tastatur***, men det er ikke ferdig utviklet og har feil:

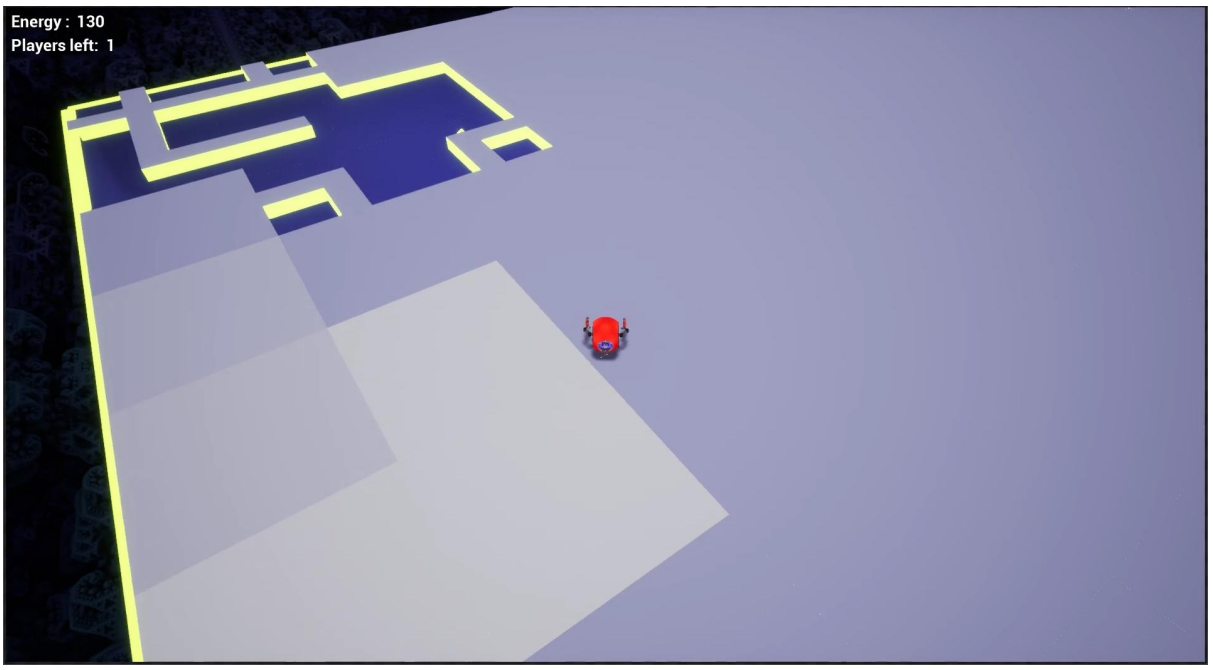
Move: AWSD keys
Camera rotation: Mousemovement
Camera zoom: D-Pad up/down
Boost: E key
Laser: Left mousebutton
Shield: Right mousebutton
Jump: Spacebar

Mål: Beveg droiden din rundt kuben og hopp for å unngå å falle inn i det svarte hullet i midten.Bruk energi på taktiske ferdigheter, men vær forsiktig: hvis energinivået ditt blir negativt kan du ikke bevege deg mer.
Hvis du skyter en motstander mister hen energi, men husk: de skyter tilbake! Bruk skjoldet ditt for å beskytte deg. Boost lar deg fly raskt.

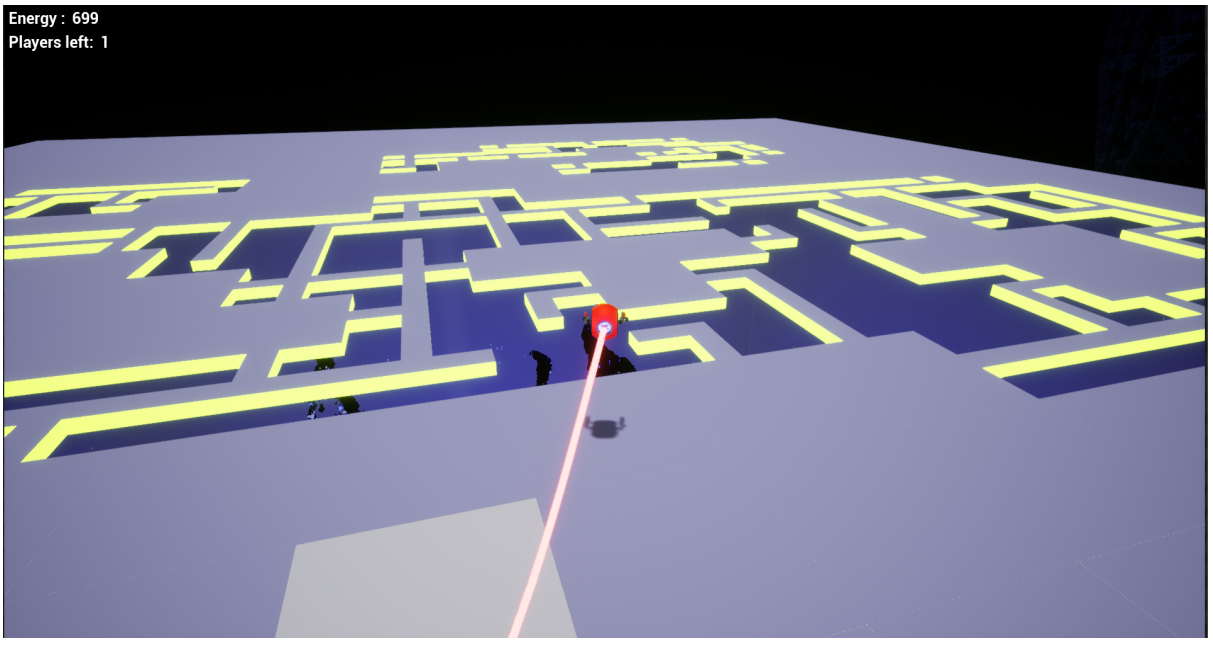


Join new game: Hvis det allerede eksisterer en kamp på serveren med mindre enn åtte spillere, blir man med i den. Hvis ikke oppretter serveren en ny kamp.

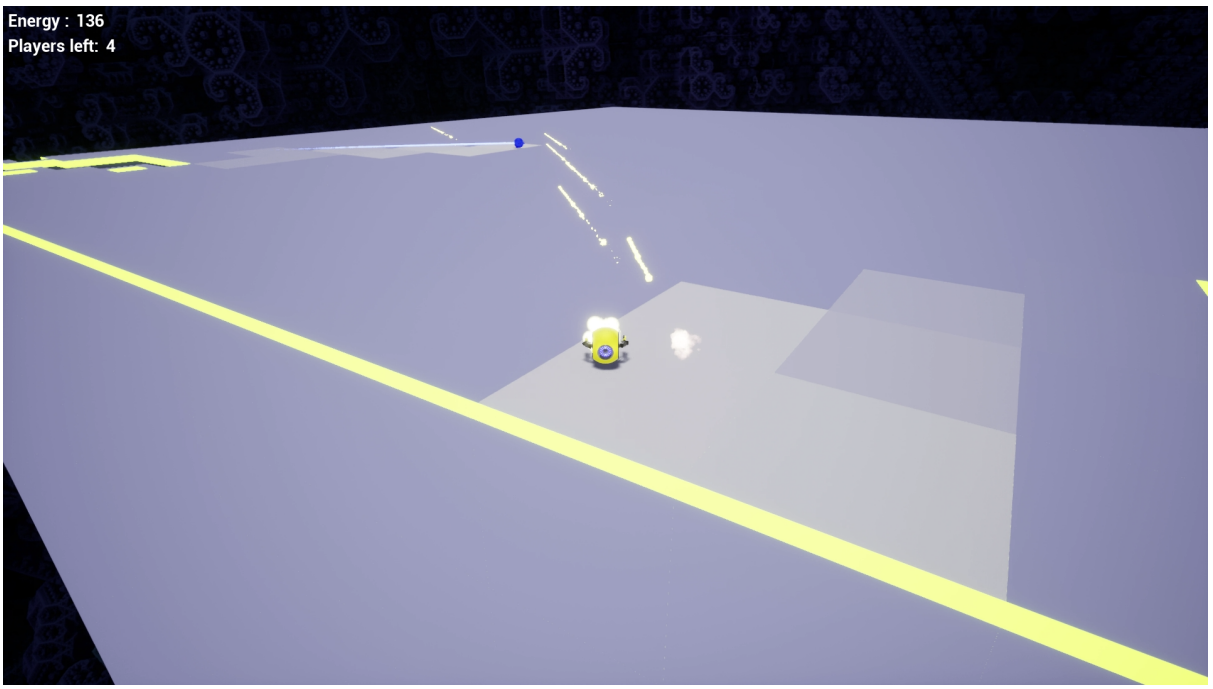
Kampen starter når den er fylt opp med åtte spillere. I testversjonen av spillet kan man starte en kamp før den er fylt opp ved å trykke **shift + S**.



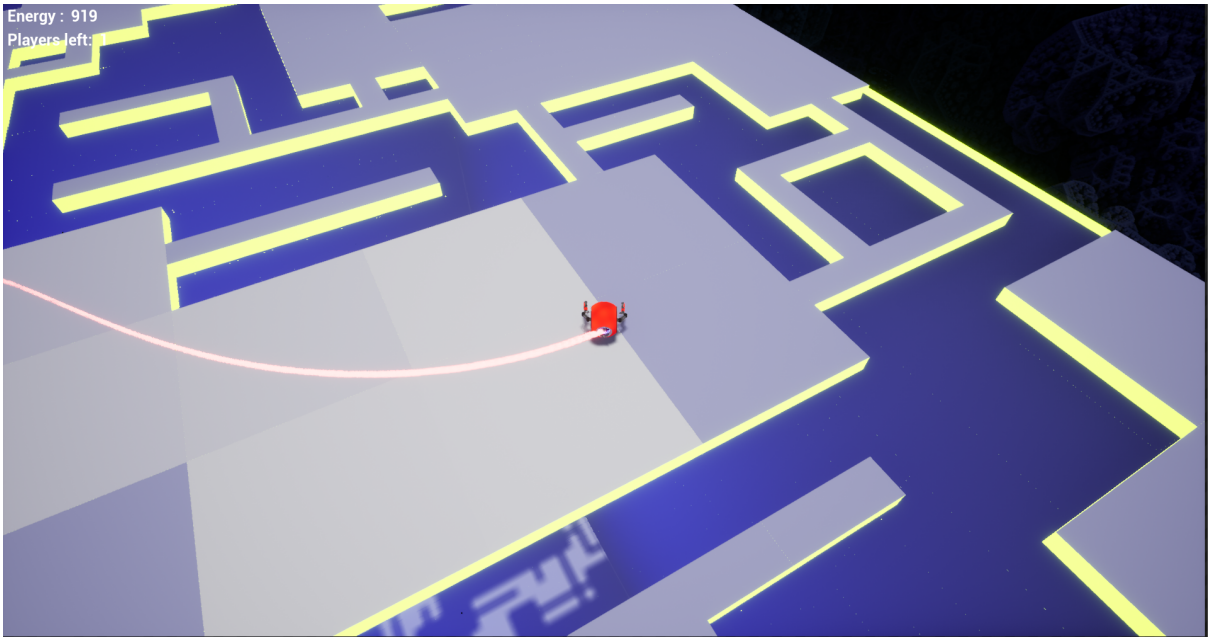
Ruter: Når droiden beveger seg over en rute/form, aktiveres en timer for fjerning av ruten/formen. Dette visualiseres med en hvit farge som falmer i løpet av to sekunder. Når fargen har falmet helt, forsvinner ruten/formen.



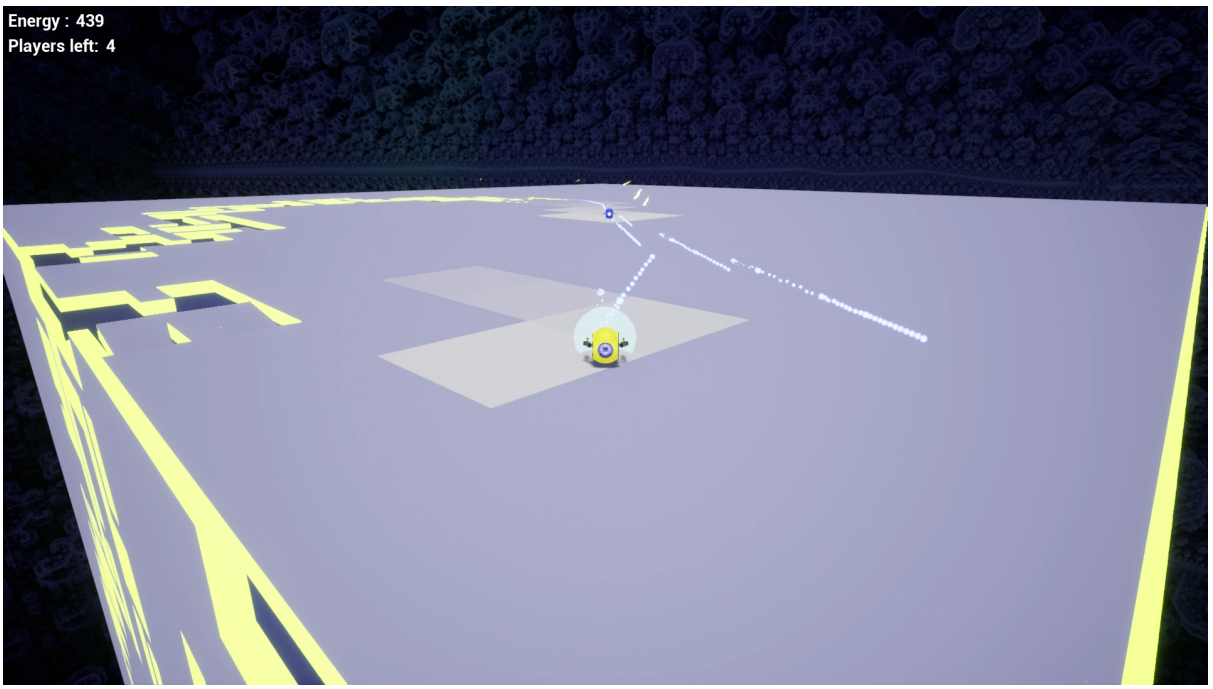
Jump: Hopp over hullene i kuben for å unngå å falle.



Laser: Hvis du treffer en motstander med laser mister hen energi.



Boost: Boost gir spilleren fart. Det er ikke mulig å bevege seg til siden under boost, men det er mulig å snu retning (rotere kamera).



Shield: Beskytter mot laserkuler.

Risikoliste

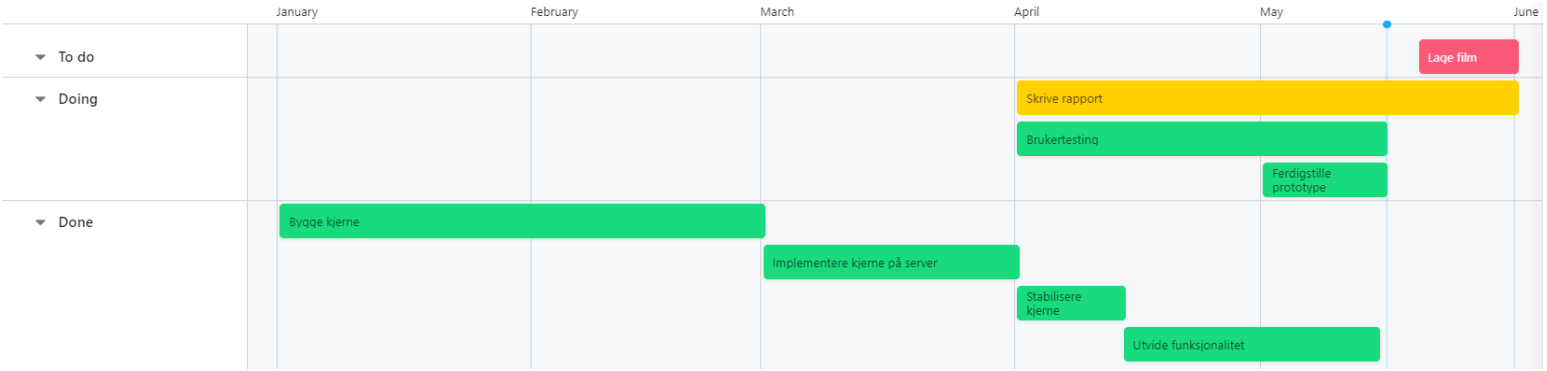
Målet med dette prosjektet er å ferdigstille en prototype. Hvor detaljert denne prototypen blir er avhengig av tilgjengelige arbeidstimer. Det føles unaturlig å beskrive det begrensede tidsrommet som en risiko; prototypen blir så detaljert som tiden tillater. Likevel er det en tidsrelatert risiko som må taes særlig hensyn til: dedikert serverimplementasjon. Serverimplementasjonen er i kjernen av oppgaven, og uten den faller hele prosjektet fra hverandre. Ingen tidligere erfaring med dedikerte servere stiller en særdeles høy risiko for prosjektet. Dette har blitt tatt spesielt hensyn til under utviklingen.
Utenom dette er prototypens suksess avhengig av at den fatter interesse. Hvis den skal få finansiering fra interessenter må den være gøy. Denne konsekvensen er ikke like høy for bacheloroppgavens suksess, men helt klart like høy for prototypens generelle suksess.

S: Sannsynlighet for at hendelsen inntreffer.
K: Konsekvenser ved at hendelsen inntreffer.
RF: Risikofaktor. Sannsynlighet multiplisert med konsekvens.

1-5: 1 er liten sannsynlighet og konsekvens av uønsket hendelse inntreffer. 5 er stor sannsynlighet og konsekvens av uønsket hendelse inntreffer.

Suksessfaktorer	S	K	RF	Tiltak
Dedikert serverimplementasjon blir for komplisert	4	10	40	Sett av ekstra god til til serverimplementasjon
Prototypen blir kjedelig	5	4	20	Innfør og utnytt brukertesting så tidlig som mulig

GANTT skjema



Figur 9.1: Oppdatert og endelig Ganttskjema.

Ganttskjemaet har fått en liten endring på slutten i forhold til den opprinnelige planen. Den opprinnelige planen var som følger:

- *Uke 1 - 9 (1. jan - 1. mar):* Bygge kjernen.
- *Uke 9 - 13 (1. mar - 1. apr):* Implementere kjernen på en dedikert server.
- *Uke 13 - 17 (1. apr - 1. mai):* Tidlig testing, stabilisering av kjerne, planlegge utvidet funksjonalitet.
- *Uke 17 - 21 (1.mai - 1. jun):* Iterativ utvikling, med intensiv testing og utviding av funksjonalitet i forhold til tilbakemeldinger fra testere.
- *Uke 21 - 23 (1. jun - 14. jun):* Ferdigstilling av prototype for visning til eventuelle interessenter. Dette inkluderer en reklamefilm av prototypen, og en plan videre i fall økonomisk støtte skulle mottas.

For å få tid til å øve til eksamen i INF250 7. juni, skrive ferdig rapporten og lage film av prosjektet, ble prototypen ferdigstilt 16. mai; Uke 21 - 23 vil ikke lenger inkludere ferdigstilling av prototype da dette allerede er gjort. Filmen lages i Uke 20 istedenfor. Hele det siste punktet fjernes, og dets innhold fremskyndes (Jamfør Ganttskjemaet).

Det har blitt satt av tid til å gjøre klart den muntlige presentasjonen 14. juni etter eksamen i INF250 7. juni.

Til tross for mangel på erfaring med dedikerte servere har alle punktene av prosjektet holdt seg innenfor de planlagte tidsrammene.