Bacheloroppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

Hans-Jeiger Gram, Ruben Solvang Valen,
Sebastian Martin Andresen

Utvikling av virtuelle kameradroner som plugin i unreal engine

Bacheloroppgave i Dataingeniør
Veileder: Ole Christian Eidheim
Mai 2020

NTNU
Kunnskap for en bedre verden

Hans-Jeiger Gram, Ruben Solvang Valen, Sebastian Martin Andresen

Utvikling av virtuelle kameradroner som plugin i unreal engine

Bacheloroppgave i Dataingeniør
Veileder: Ole Christian Eidheim
Mai 2020

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

NTNU
Kunnskap for en bedre verden

Forord

Denne bacheloroppgaven, for 3 årig bachelor dataingeniør vår 2020, er gjennomført ved Institutt for Datateknologi og Informatikk ved Norges Teknisk - Naturvitenskapelige Universitet. Oppgaven ble utlyst av Riddlebit Software.

Vår gruppe valgte denne oppgaven, fordi det virket som en spennende utforsking av et domene vi ikke hadde særlig erfaring ved, nemlig spillutvikling. Oppgaven tillot oss også stor kunstnerisk frihet, og ville derfor tillate en større utforskning av hva slags produkt vi ville utvikle. Prosessen fram til resultatet innebar mye lærdom og subjektiv vurdering av produktet.

Vi ønsker å takke vår veilder ved NTNU, Ole Christian Eidheim. Han har vært til hjelp gjennom hele prosjektet, og har vært lett tilgjengelig og har gitt raske tilbakemeldinger. Vi takker også bedriftskontakten vår, Jonathan Jørgensen. Han har hjulpet gruppen med flere deler av prosjektet, både idéer, tilbakemeldinger, og en workshop for Unreal Engine han holdt. Vi takker også resten av Riddlebit for å ha gitt oss hjelp og råd under utviklingsprosessen.

Opprinnelig Oppgavetekst

Arbeidstittel: Virtuelle Kameradroner

Hensikten med oppgaven
Hensikten med oppgaven er å videreutvikle DeepSpec-systemet for å styre virtuelle kameradroner i spillet Setback.

Kort beskrivelse av oppgaveforslag
Dataspill har allerede etablert seg som en tilskuersport, men dagens løsninger på hvilke deler av spillet tilskuerne skal se er ikke tilstrekkelige for alle typer spill. Derfor utvikler vi et system hvor kameradroner styres rundt på kartet for å fange mest mulig av det som skjer i spillet. Oppgaven kan vinkles til å fokusere på ett eller flere spesifikke aspekter av prosjektet, som f.eks. maskinlæring, dronekontroll eller kinematografi.

Oppgaven passer for:	- Bacheloroppgave
Kan oppgavestiller stille arbeidsplass med nødvendig utstyr og programvare:	Deltid
Hvis ikke, hva kreves av maskin og programvare:	Minst én windows-maskin
Oppgaven passer best for antall studenter:	2-3

Opplysninger om oppgavestiller

Navn på bedrift eller ogranisasjon:	Riddlebit Software
Adresse	Munkegata 58
Postnummer	7011
Poststed	Trondheim
Navn på kontaktperson/veileder:	Jonathan Jørgensen
Telefon:	48206901
Epost:	jonathan@riddlebit.net

Sammendrag av Bacheloroppgaven

Med stadig flere flerspiller spill der konkurranse er hovedfokus er det nødvendig å ha et system for automatisk observasjon. Det finnes få open-source løsninger for dette i dag. Derfor ønsket Riddlebit Software å utvikle en løsning for dette, slik at de kan implementere det i spillet Setback. Gruppen ønsket også å utvikle en open-source løsning som kan implementeres av en rekke andre kompetitive spill. For å utforske muligheten for en slik løsning har dette prosjektet gått ut på å programmere et system for kameradroner i Unreal Engine med fokus på å gi et godt helhetlig bildet av konkurransen. Vi forsøkte også å se på mulighetene for en flokk-algoritme for spredning av droner i rommet. Dette resulterte i et dronesystem som kan benyttes i Unreal Engine prosjekter, er tilpasselig, benytter komposisjonsregler, og tilpasser seg tilstanden i 3d miljøet. Denne rapporten legger frem en av mange mulige måter å utvikle et slikt system, og hvilke faktorer som kan utvides for å skape et bedre produkt.

Tittel:	Utvikling av virtuelle kameradroner som plugin i unreal engine
Oppgave nummer:	47
Dato:	May 24, 2020
Deltakere:	Sebastian Martin Andresen Hans-Jeiger Gram Ruben Solvang Valen
Veileder:	Ole Christian Eidheim
Oppdragsgiver:	Riddlebit Software
Bedriftskontakt:	Jonathan Jørgensen, jonathan@riddlebit.net
Antall sider:	61

Contents

1 Introduksjon og relevans

1.1 Bakgrunn

I kompetitivt spill er det viktig å ikke bare kunne spille, men også se på andre som spiller. Kanskje du har lyst til å følge med på en venn, eller se på kampene til de beste spillerne? Elektronisk sport, eller e-sport, har i de siste årene blitt en viktig del av opplevelsen knyttet til et kompetitivt spill. Om det er i en uformell eller profesjonell sammenheng er observasjon viktig. Man ønsker å ha god oversikt over hva som foregår på banen, enn hva som er mulig å se igjennom en spillerskamera. I førstepersonsspill har spillerkameraet et begrenset synsfelt, som betyr at mye av informasjonen på kartet faller utenfor skjermen. I et slikt tilfelle kan det være nyttig å la observatøren være separat fra spillerkameraet. Et kamera som kan bevege seg rundt på banen kan anvendes i dette tilfellet, representert av et kvadkopter - en drone. Slike kameradroner benyttes i sport i virkeligheten.

En slik drone burde også søke etter å få et bilde med mye informasjon, presentert på en tilfredsstillende måte. Denne oppgaven er et forsøk på å finne ut av hvordan en slik drone kan lages og benyttes for Setback, et spill utviklet av Riddlebit Software, samt andre prosjekter.

1.1.1 Setback

Setback er et "fast-paced online shooter" spill. Riddlebit Software begynte utviklingen av spillet i 2017. Spillet går ut på å komme seg gjennom et sett med checkpoints spredt ut over en bane. Første person gjennom alle checkpointene vinner runden. Det er et førstepersons skytespill, og når noen blir skutt nok blir de sendt bakover i tid.

1.2 Forskningsspørsmål

Vi stilte følgende spørsmål for å definere domenet for oppgaven vår, da denne oppgaven sto åpent for å angripes på mange måter.

1. Hvordan kan et system med kameradroner forholde seg til flere dynamiske interessepunkter?
2. Hvordan kan systemet skape en helhetlig oversikt over interessepunktene?
3. Hvordan kan en kameradrone tilpasse seg et sett med interessepunkter for å oppnå en visuell komposisjon?
4. Hvordan kan dronesystemet designes så det kan integreres i ulike prosjekter?

1.2.1 Hensikt med spørsmålene

Spørsmål 1

Prosjektet omhandler 3D-miljøer hvor det er mange foregående hendelser til enhver tid. Oppgaven har vært å finne en måte å bruke kameradroner på flere hendelser rundt i rommet, og filme disse.

Spørsmål 2

Hendelsene i dette miljøet skjer flere steder spredt rundt i rommet. Her er hensikten å finne ut hvordan dronesystemet kan posisjonere kameradronene rundt i rommet slik at systemet kan få en god oversikt over alle hendelsene.

Spørsmål 3

Det er flere interessepunkter i dronens synsfelt. Ønsket er å kunne putte disse punktene i gitte komposisjoner i dronens kamerabilde.

Spørsmål 4

Et automatisert dronesystem kan være nyttig i flere prosjekter, fra spill til andre samfunnsnyttige prosjekter. Så det å desgine et system som kan nyttes av flere er av interesse her.

1.3 Rapportens oppbygging

Rapporten dekker prosjektets vitenskapelige side og den ingeniørfaglige siden av oppgaven.

- **Kapittel 2** - Det teoretiske grunnlaget for rapporten. Det er forventet at leseren av dette dokumentet har grunnleggende kunnskaper om informatikk. Derfor blir det teoretiske grunnlaget som blir beskrevet fokusert på den kinematografiske delen av oppgaven, og teori brukt for et optimaliseringsproblem.
- **Kapittel 3** - Her presenteres valg av teknologier og metoder. Teknologidelen dekker de programmer og verktøyer gruppen har benyttet i prosjektperioden. Metodedelen gjør rede for delene av droneutviklingen som kan deles inn i diskrete problemer, hvordan de problemene ble løst, og hvordan de løsningene samarbeider. I dette kapittelet blir også administrative metoder forklart, og hvordan arbeidsprosessen ble strukturert.
- **Kapittel 4** - Resultatene fra dronesystemet som relaterer direkte til forskningsspørsmålene presentert. Det er også en liste over kravene til oppgaven og det er gjort rede for statusen til disse, og en dokumentasjon av hvordan arbeidsprosessen i oppgavetiden har vært utført.

- **Kapittel 5** - Her har gruppen skrevet om hvordan resultatene kan tolkes i forhold til forskningsspørsmålene, og hvorfor de ingeniørfaglige målene endte opp slik de er og hva det betyr. Det blir også her skrevet refleksjon rundt arbeidsprosessen.
- **Kapittel 6** - Konklusjoner av oppgaven trukket basert på resultatene i kapittel 4 og kapittel 5 i forhold til forskningsspørsmålene som ble presentert i seksjon 1.2.

2 Teori

Denne rapporten tar utgangspunkt i at leseren har grunnleggende kunnskaper om informatikk. Det er flere deler av prosjektet som faller utenfor typiske informatikkfag slik som kamera- og kinematografiteori. I tillegg er det brukt spesifikke metoder som krever et teoretisk grunnlag utenom de forutsatte kunnskapene før de kan benyttes videre i prosjektet.

2.1 Kinematografi

Kinematografi er kunsten om filmfotografi og filming med elektroniske eller kjemiske metoder. Det handler om de faglige og kunstneriske valgene når det kommer til innspilling av film. Kinematografi omhandler alt av lys, farge, komposisjon, kamerateknikk og visuelle effekter i forhold til kameraarbeid. I dette prosjektet er det vært jobbet med selve komposisjonsdelen av fagfeltet.

2.1.1 Tredelingsregelen

Tredelingsregelen, eller "Rule of Thirds", er den best kjente komposisjonsregelen vi har i moderne fotografi. I følge denne regelen kan et bilde deles inn i tre like store deler horisontalt og vertikalt, slik at det skapes kryss hvor fokuset i bildet skal plasseres (Krages, 2005). Tradisjonelt blir dette forklart ved at menneskets øye søker langs disse punktene når det først ser på et bilde, men dette er ikke en forklaring grunnet i forskning (Bellard, 2019). Den viktigste effekten av å bruke denne regelen er at det bekjemper det naturlige innsiktet man har til å sette hovedfokus i senter i et bilde (Krages, 2005).

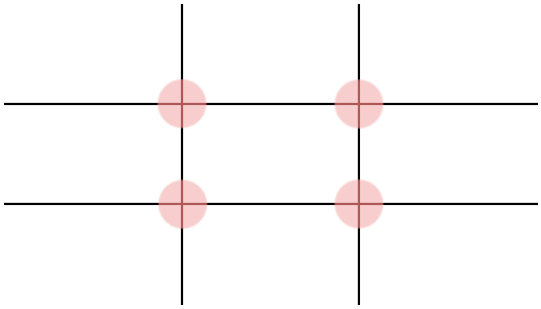


Figure 2.1: Tredelingsregelen, fokus burde plasseres nær kryssene markert med rosa.

2.1.2 Gestaltteori

Det har vært mye forskning innen nevrologi og psykologi på hvordan mennesker oppfatter visuell informasjon. Når det kommer til oppfatting av bilder er det mest prominente bidraget gestaltteori (Johan Wagemans and von der Heydt, 2012). Gestaltteori forteller

oss at når menneskehjernen oppfatter et bilde ser den først på hele bildet før den begynner å analysere delene. I følge gestaltteori følger menneskets visuelle oppfatning følgende prinsipper:

- Mennesket har en tendens til å skille mellom en figur og bakgrunnen.
- Objekter som er nær hverandre blir sannsynligvis oppfattet som en gruppe.
- Objekter som er like har større sjanse for å bli ansett som en gruppe.
- Det er en tendens til å oppfatte en komplett figur selv når deler av informasjonen mangler.
- Det er en tendens til å oppfatte subjekter som kontinuerlige figurer.

Gestaltteori viser da hvordan forskjellige komposisjonsvalg kan påvirke oppfatningen av et bilde. Mangelen på dybdesyn gjør at det gjelder andre regler for bilder enn det gjør for det man ser med det menneskelige øyet.

Gestaltteori og annen forskning på billedlig oppfatning viser også at mennesker oppfatter subjekter ved å oppfatte bildet først som en helhet. Mennesker oppfatter altså ikke bilder ved å gjøre en diskret analyse av de individuelle elementene i rammen, eller ved å følge ledelinjer slik som f.eks tredelingsregelen vil tilsi. Hva som anses som god bildekomposisjon kan være basert på balanse mellom separate elementer, men bildet blir likevel oppfattet som en helhet (Krages, 2005).

2.1.3 Åpen og lukket innramming

Innen film er åpen og lukket to strategier for innramming. Hvilken strategi som benyttes til en scene baseres gjerne på hvorvidt regissøren ønsker å inkludere seeren i bilderommet og hva naturen av det som filmes tillater.

Åpne innramminger er scener hvor regissøren ikke har mulighet til å plassere objekter i scenen eller bestemme bilderommet før filming. Åpne innramminger kan ofte sees i dokumentarer hvor det er sjeldent å ha full kontroll over en scene.

Lukkede innramminger er da scener hvor objektet og bilderommet blir nøye plassert for å gi best klarhet og balanse. Lukkede innramminger oppfattes ofte som falskere og kontrollerte.

Skillet mellom de to typene er ikke alltid enten eller. All komposisjon er på et vis bestemt av fotografen enten det er tydelig for seeren eller ikke. Reklamer vil veldig ofte ha en veldig forsiktig konstruert komposisjon med hensikt til å se naturlig ut (Katz, 1991).

2.1.4 Tracking shot

Et tracking shot er et type bilde hvor kameraet og subjektet har den samme farten. I et slikt type bevegende bilde er reglene for komposisjon ekvivalent med et statisk bilde. Om kameraet er plassert foran, bak, eller til siden for subjektet er irrelevant. Kameraet kan også filme fra så nært eller langt unna som nødvendig (Katz, 1991).

2.1.5 Blenderåpning

Kamera har en innstilling kalt blenderåpning. Blenderåpningen er arealet til linsen. Når blenderåpningen er stor slippes mer lys inn i kameraet, og når den er liten slippes mindre lys inn i kameraet. Størrelsen på blenderåpningen styres av en knivskive. Innstillingen til knivskiven måles i f-stopper. En høy f-stop betyr at knivskiven er mer lukket, og blenderåpningen er mindre. En lav f-stopp betyr at blenderåpningen er større. Når blenderåpningen er stor har bildet et grunt fokusområde, det vil si at når et objekt er i fokus vil andre objekter som har høyere eller lavere avstand til linsen være uklare, dette gjør at man kan fremheve de objektene man ønsker uten at de blander seg inn i omgivelsene. En smal blenderåpning vil gi et dypere fokusområde. Dette vil si at kameraet kan se objekter som har forskjellige avstander fra linsen, uten at de som ikke står i ideell avstand fra linsen blir uklare (Epic Games, 2018).

2.2 Dronekontroll

Kontrollering av flere droner i en samkjørt sverm har vært fokus i flere forskningsartikler. Disse har for det meste fokusert på fysiske droner som opererer i den virkelige verden (Ross et al., 2012), men er fortsatt nyttig da droner i et virtuelt spillmiljø bør oppføre seg på lignende måte. Forskjellen her er at siden dronene er virtuelle er det enklere å finne nøyaktig informasjon om hver drone og hva den gjør. Dette simplifiserer problemer som noen av forskningsartikler har måtte løse (Saska et al., 2014). Her har det vært fokus på visjonsbasert kontroll for å oppfatte droner som er rundt seg selv, og brukes maskinlæring for optimalisering av kontroll (Saska et al., 2016) (Sadeghi and Levine, 2017).

2.2.1 Flokk oppførsel

"Reynolds flocking" er en algoritme for å simulere flokkoppførsel lik den som kan observeres i fugler og fisker (Reynolds, 1987). Den har en enkel skapning som blir kalt for en boid, og bruker tre enkle regler for hvordan hver boid skal oppføre seg.

- **Separasjon** - Styr for å unngå kollisjon med andre boids
- **Samkjøring** - Styr mot en felles retting basert på lokale flokkmedlemmer
- **Sentrering** - Styr mot en gjennomsnittlig posisjon basert på lokale flokkmedlemmer

Statisk separasjon og dynamisk samkjøring er komplementære. Sammen får de en oppførsel som tilsvarer naturlig flokkoppførsel. Her prøver hver boid å holde avstand for å unngå kollisjon, men på samme tid holde seg samlet i en flokk. Hver boid har også en gitt oppfatningsradius som passer på at hver boid bare bryr seg om nærliggende boids. Likt fulger som ikke har oversikt over hele flokken, men følger hva naboene gjør.

Separasjon er bare basert på relativ posisjon fra andre boids. Mens samkjøring er basert på fartsvektor og ignorerer posisjon. Sentrering gjør at hver boid flyr mot et felles senter basert på boids som er innenfor oppfatningsradiusen sin. Hvis boiden er

nærme sentreringspunktet vil sentreringsfaktoren være liten, men hvis flokken er på kanten av oppfatningsradiusen vil faktoren være sterkere og den vil styre inn mot sentreringspunktet.

Disse tre reglene er separate og produserer isolerte forslag om hvilken retning boiden skal bevege seg i. Hver regel gir endring i fartsvektoren, altså en akselerasjonsvektor. Hver regel har også en multiplikativ faktor som er med å endre styrken på den gitte akselerasjonen. Det er opp til hver boid å samle informasjon om boids rundt seg og regne ut relevant akselerasjon. Den må kombinere og prioritere for å unngå konflikter. For å enkelt kombinere akselerasjonsvektorene tar boiden et gjennomsnitt fra de tre reglene, eller et vektet gjennomsnitt da hver regel har en multiplikativ faktor.

Kompleksiteten til denne algoritmen vokser kvadratisk med størrelsen på flokken ($o(n^2)$) (Reynolds, 1987). Da hver boid må regne ut sin relative fartsvektor basert på alle andre boids. Men siden hver boid bare tar hensyn til boids som er innenfor oppfatningsradiusen sin, vil kompleksiteten synke ved stor separasjon av flokken.

3 Teknologi og metode

Dette kapitlet gjør rede for de valgene som er tatt for å løse oppgaven og svare på forskningsspørsmålene og prinsippene som er anvendt. De valgene som er gjort er basert både på kunnskap gruppen hadde før prosjektperioden og på kunnskap som ble tilegnet under. Noen av valgene bygger på teori forklart i kapittel 2, og noen bygger på egne ferdigheter.

3.1 Teknologi

Her blir det gjort rede for hva slags teknologier som er anvendt. Videre blir det gjort rede for hvilke aspekter av teknologien som er relevant for dette prosjektet.

3.1.1 Droner

I konteksten til denne oppgaven er en ikke-virtuell drone et kvadrokopter. Det er et luftfartøy som oppnår oppdrift ved bruk av fire selvstendige rotorer. Ved å variere farten på rotorene kan dronen generere oppdrift, og justere for ubalanse fra vind og bevegelse for å holde seg stabil. Droner blir mye brukt for filming da det er en tilgjengelig måte å fange bilder fra luften.

3.1.2 Virtuelle droner

Dronen vi skal programmere skal eksistere i et virtuelt 3d miljø. I UE4 vil dronen være definert som et punkt som kan flyttes dit den skal være. Det er ikke noe simulert vind som det må tas hensyn til, og rotorer som er en del av 3d modellen som festes til dronen er kun estetiske. Dette vil si at en virtuell drone ikke støter på mange av de problemene som en fysisk drone møter på. Dronen trenger ikke å ta hensyn til noe vær, den trenger ikke oppfatte omgivelsene selv og kan heller hente informasjon om det fra minnet, og selvstyringen trenger ikke å unngå å fly over mennesker i tilfelle en av motorene slutter å fungere. På den andre siden må dronen ta hensyn for å ikke bryte illusjonen om at den er en drone. Det vil si at den må unngå å teleportere, gå gjennom terreng, eller bryte andre av fysikkens lover.

3.1.3 Kunstig intelligens i spill

Kunstig intelligens i spill er ikke det samme som akademisk kunstig intelligens. I spill er det mer basert på smarte/spesialiserte algoritmer og beslutningstrær. Mens i det akademiske er mer basert på optimaliserting av et gitt problem, og er ofte ikke skalerbare

eller praktiske nok for spill. Det er forsøkt, og har blitt gjort store fremskritt på å knytte det akademiske mer mot spill (Yannakakis, 2012), men fortsatt lite brukt i dag.

I dette prosjekt fant vi det mer gunstig å bruke noen definerte algoritmer til å styre dronen rundt i miljøet. Siden bare ett gruppemedlem hadde erfaring med det maskinlære valge vi å ikke gå den rettningen. Vi ønsket også å vinkle oppgaven mot kinematografi. Dette gjør at systemet er enklere å forklare og justere, dersom man har ønsker om å endre på noe oppførsel eller legge til nye regler for dronene.

I den opprinnelige oppgaveteksten er det gitt at systemet skal være en videreutvikling av DeepSpec-systemet. DeepSpec var en bacheloroppgave som fra 2019 for Riddlebit Software som brukte et nevralt nettverk for å predikere interessepunkter i Setback (Bjørseth et al., 2019). Vi fikk senere beskjed fra oppdragsgiver om å ikke bruke dette i oppgaven da modellen ikke predikerte langt nok inn i fremtiden.

3.1.4 Unreal Engine 4

Dronesystemet er i sin helhet programmert i Blueprints i Unreal Engine. UE4 er en open-source spillmotor utviklet av Epic Games. Motoren er bygget på C++ og bruker det i motoren. Blueprint er et visuelt skript system i UE4 som en abstraksjon av C++. Blueprints lagres som kompilerte .uasset filer, ikke .cpp.

Det var naturlig å lage systemet i UE4 da spillet oppgaven er rettet mot også er laget i UE4.

Rotator-variabeltype

I UE4 er det en egen type variabel som heter Rotator, og definerer en rotasjons-transformasjon i 3d-rommet. Denne består av tre flyttall, for rotasjoner rundt de tre aksene x, y og z. I figur 3.1, 3.2 og 3.3 vises det hvordan de tre variablene påvirker et objekt i form av et kamera.

- Rotasjon om x-aksen kalles Roll (fig. 3.1).
- Rotasjon om y-aksen kalles Pitch (fig. 3.2).
- Rotasjon om z-aksen kalles Yaw (fig. 3.3).

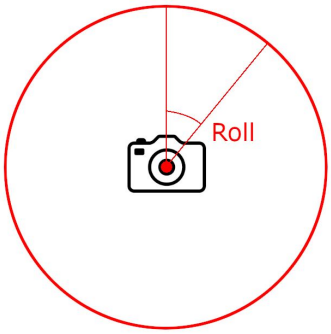


Figure 3.1: Hvordan Roll roterer et objekt. Kameraet er sett forfra.

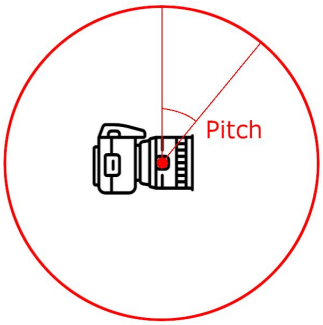


Figure 3.2: Hvordan Pitch roterer et objekt. Kameraet er sett fra siden.

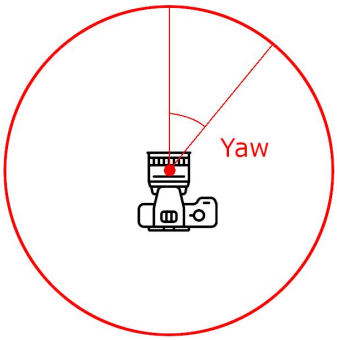


Figure 3.3: Hvordan Yaw roterer et objekt. Kameraet er sett ovenifra.

Brukte UE4-objekter

- **Actor**
  I UE4 er alle objekter/klasser som kan plasseres i 3d-miljøet barn av actor-klassen. Disse får en transformasjon i 3d-rommet bestående av lokasjon, rotasjon og skala.

- **Pawn**
  Pawn er barn av Actor. Dette brukes gjerne for figurene til spillkarakterer. Disse kan bli kontrollert av spillere eller spill-AI.

- **AIController**
  Controller er barn av Actor, og blir brukt for å kontrollere Pawns handlinger. Det er to typer som brukes mest. PlayerController er en type Controller som styres av spillere. AIController er en Controller som styrer pawns med en definert logikk. I denne pluginen brukes AIController for å styre droner.

- **Actor Component**
  En Actor-Component er en klasse som ikke har en transformasjon, men som kan festes ved Actors.

Line Trace for Objects

Line Trace For Objects er en funksjon i UE4 som sjekker om det finnes objekter mellom to punkter. Det trekkes en linje fra et punkt A til punkt B. Funksjonen returnerer et objekt som forteller om det var et objekt av en type på linjen. Objekttyper som skal sjekkes for er definert av utvikleren. I vårt produkt anvendes dette for å sjekke etter objekter for å unngå kollisjon med dem, og for å sikre frie siktlinjer. UE4 har flere typer "Traces", som for eksempel "Sphere Trace" som sender ut en kule, og returnerer info om hvordan kulen eventuelt krysser med et objekt.

Line Trace for Objects er den minst ressurskrevende typen trace. Siden produktet skal være skalerbart, og vi ikke har bruk for annet er det denne typen trace vi benytter.

InterpTo

InterpTo er en funksjon i UE4 som brukes for å flytte en verdi mot en annen med en jevn bevegelse ved hjelp av interpolering. Funksjonen finnes for de vanligste kontinuerlige datatypene i UE4, som er Floats, Vectors og Rotators. Funksjonen ser slik ut:

```
//Interpolate To
//current: variabelen ved starten
//target: variabelen som current skal bevege seg mot
//deltaTime: Tid siden forrige utregning
//speed: Variabel som bestemmer hvor fort bevegelsen gaar
function interpTo(*datatype* current, *datatype* target, float deltaTime,
    float speed){
    return current + (target - current) * deltaTime * speed;
}
```

Om vi for eksempel setter verdiene slik:
current: 0
target: 10
DeltaTime: 1/60 (for 60 fps)
Speed: 1

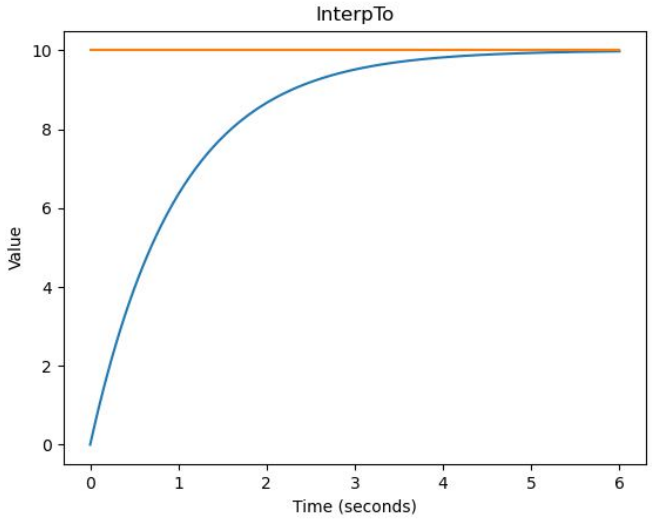


Figure 3.4: Hvordan InterpTo endrer en verdi over tid.

Her er den blå kurven current-verdien, mens den oransje linjen er target-verdien current går mot. Over tid har current nærmet seg target med en jevn bevegelse.

3.1.5 Blenderåpning

I et spill hvor handlingen kan dekke et stort vertikalt areal, og dronen filmer fra siden, trenger vi et dypt fokusområde for at handlingen som blir fanget av kameraet ikke faller ut av fokus. Som forklart i kapittel 2.1.5 vil en stor f-stopp gi et dypere fokusområde. Kameraet til dronen har derfor en liten blenderåpning, med f/8.

3.1.6 Andre programmer

- **GitHub**
  Brukt for versjonskontroll av utviklet programvare.
- **Python**
  Brukt for et optimaliseringsproblem.

- **Overleaf**
  Brukt for å skrive rapport.
- **Google Drive**
  Brukt for å dele filer med hverandre.
- **Microsoft Teams**
  Brukt for kommunikasjon med veileder.
- **Discord**
  Brukt for kommunikasjon med hverandre og oppdragsgiver.

3.2 Metode

Denne seksjonen forklarer hvordan oppgaven har blitt løst praktisk. Arbeidsmetode beskrives.

3.2.1 Roller

Rollefordelingen på prosjektet ble beskrevet slik i gruppekontrakten:
"Vi fordeler ikke overordnede roller. Vi anser heller oppgaver som de kommer, og fordeler ansvar deretter. Møteleder og referent går på rundgang."

3.2.2 Scrum

Gruppen ble enige om å bruke Scrum som arbeidsform grunnet tidligere erfaring hvor dette ble oppfattet som en god måte å arbeide på. Det ble også enighet om å vente til etter eksamen (18.03.2020) før vi begynte å følge Scrum-prosessen skikkelig, da undervisning, øvinger, og eksamener ville gjøre det vanskelig å sette gode sprint-mål. Før dette ble mer tid brukt på å lære å bruke UE4, samt å definere oppgaveomfanget.

3.3 Dronesystemet

Her beskrives systemets funksjonalitet og struktur. Først og fremst vil den generiske pluginen som er anvendbar i alle UE4-prosjekter gjøres rede for, og der den Setback-spesifikke pluginen skiller seg ut vil dette beskrives separat. Det gjøres også rede for et optimaliseringsproblem som er utforsket i sammenheng med å finne en alternativ metode som kan anvendes for å løse forskningsspørsmålene (ref. 1.2).

Det som beskrives her er å finne i kildekoden som er lagt ved som vedlegg for rapporten. Den generiske pluginen, den Setback-spesifikke pluginen og koden for optimaliseringsproblemet har hver sin prosjektkatalog.

Hovedproduktet for dette prosjektet er en UE4-plugin. Den kan anvendes i alle UE4-prosjekter i 3d-miljø, og er ment for å videreutvikles og tilpasses det enkelte prosjektet. Selve pluginen er designet og dokumentert i et forsøk på å gjøre videreutviklingen intuitiv.

Det er også laget en videreutvikling av denne pluginen for oppdragsgivers spill Setback, hvor pluginen er blitt spesielt tilpasset aspekter ved spillet som vi kommer nærmere inn på iløpet av dette kapittelet.

3.3.1 Pluginstruktur - Klassediagram

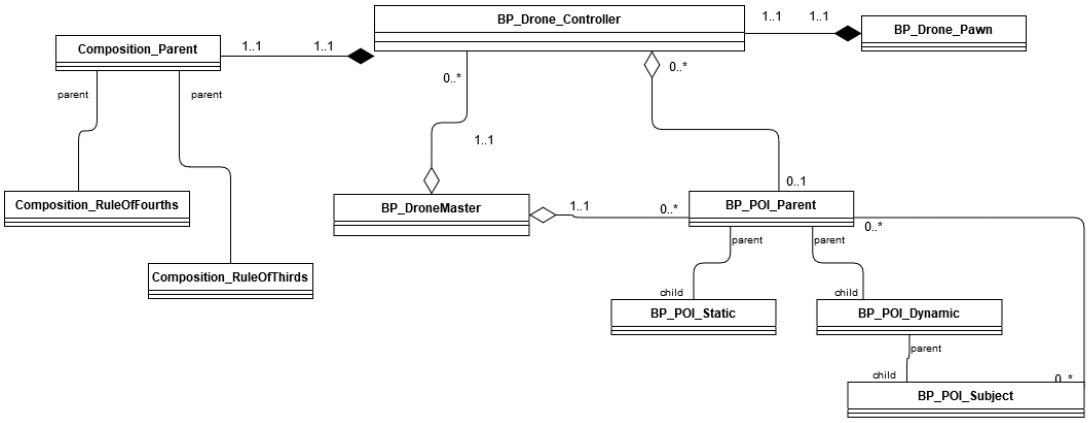


Figure 3.5: Redusert versjon av klassediagram for den generiske pluginen.

- **BP_Drone_Pawn -** Actor som representerer selve dronen.
- **BP_DroneMaster -** Actor som styrer systemet (se kapittel 3.3.3)
- **BP_Drone_Controller -** AIController som styrer *BP_Drone_Pawn*.
- **Composition_Parent -** en Actor Component som bestemmer komposisjonen dronen skal forsøke å filme i (se kapittel 3.3.5).
- **BP_POI_Parent -** Actor Component som brukes for å definere en Actor som interessepunkt (se kapittel 3.3.2)

3.3.2 System for interessepunkter

Det kan være mange forskjellige ting som ønskes å filmes i et prosjekt. I dette systemet blir alle disse abstrahert til enkelte punkter som systemet forholder seg til, som her kalles interessepunkter.

I 3d-omgivelsene kan det være mange forskjellige typer Actors en ønsker at kameradronene skal oppfatte, og man må da kunne definere dem som interessepunkter. Likevel er det ikke hensiktsmessig å behandle alle interessepunkter likt. For eksempel burde dronen kunne skille mellom en spillerkarakter og et prosjektil, siden disse oppfører seg ulikt

i et spillmiljø. Derfor er det laget et system av POI Actor Components som kan anvendes etter behov. POI-komponentene inneholder den informasjonen dronene trenger for å forholde seg til dem, og de kan anvendes i alle 3d-miljøer i UE4 siden de er Actor Components som kan festes til alle typer Actors som er plassert i 3d-miljøet.
Forholdet mellom ulike typer interessepunkter er beskrevet i figur 3.6.

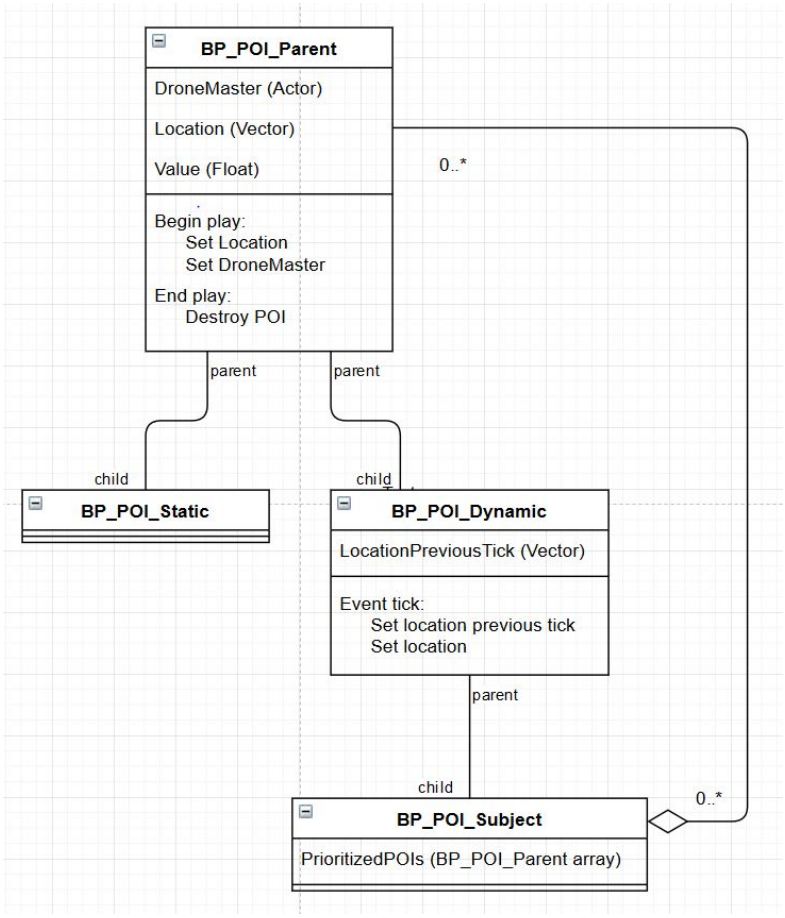


Figure 3.6: Klassediagram for interessepunkter.

Alle POI-komponenter som skal anvendes er barn av en hovedkomponenten *BP_POI_Parent*. I denne komponenten kjøres metoder for opprettelse og ødeleggelse av komponentene ettersom de opprettes og fjernes fra 3d-miljøet. Denne har også variabler for interessepunktets lokasjon, og en verdi som brukes i beregning av droners siktsverdi (ref. seksjon 3.3.9).

BP_POI_Static er POI-komponent for statiske interessepunkter. Disse henter kun informasjon om sin Actors posisjon når de opprettes, og er barn av *BP_POI_Parent*.

Denne komponenten brukes på Actors som ikke flytter seg i koordinatsystemet, som kan være for eksempel Checkpoints.

BP_POI_Dynamic er POI-komponent for dynamiske interessepunkter, og er barn av *BP_POI_Parent*. Her hentes informasjonen om Actors posisjon hvert tick, siden de vil bevege på seg. Denne komponenten kan brukes på for eksempel ulike typer prosjektiler og karakterer som beveger seg.

BP_POI_Subject er POI-komponent for Actors som man vil at dronen skal følge og filme som subjekt, og er barn av *BP_POI_Dynamic*. Denne komponenten har et array med andre interessepunkt-komponenter som dronen tar med i beregninger av POIMidpoint og ViewValue, og arrayet blir oppdatert jevnlig av DroneMaster. Denne komponenten er ment først og fremst for spillerkarakterer, men den kan festes til hvilke som helst Actors.

Dette systemet fungerer som sagt for hvile som helst miljøer, men det er laget med tanke på å tilpasses miljøet det brukes til. Et spill kan ha mange forskjellige dynamiske interessepunkter det vil følge, som for eksempel mange ulike typer fiender og prosjektiler. Det kan hende de også har ulike Actors som de vil kamera skal følge, men likevel behandles ulikt.

Dersom den som anvender Pluginen har behov for dette, er det meningen å lage barnekomponenter av de relevante POI-komponentene. I disse kan det opprettes egne variabler og funksjoner etter behov. Det kan også endres hvordan dronen forholder seg til den enkelte typen interessepunkt, som er spesielt viktig for utregning av midtpunkt (ref. seksjon 3.3.4) og siktsverdi (ref. seksjon 3.3.9).

Et eksempel for tilpassede POI-komponenter har vi laget i en Setback-spesifikk plugin. Figur 3.7 viser hvordan POI-systemet i figur 3.6 er utvidet for Setback:

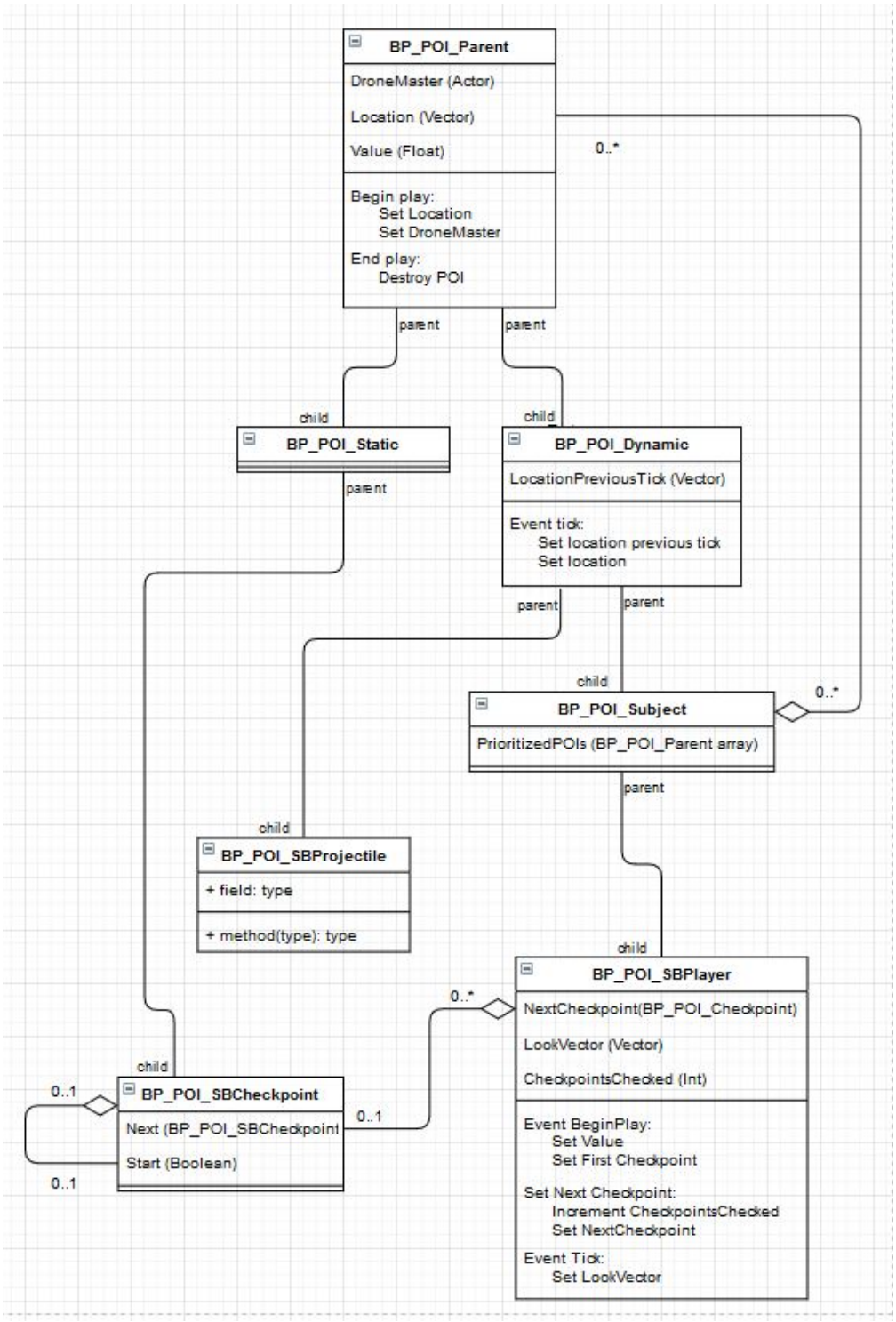


Figure 3.7: Klassediagram for interessepunkter i Setback.

Her er det laget tre spesifiserte POI-barnekomponenter. Den ene er *BP_POI_SBCheckpoint*. Denne komponenten brukes på checkpointene som spillerne skal gjennom. Denne har en egen Boolean for om checkpointet er det første spillerne skal gjennom, og en variabel for Checkpoint-POI-komponenten til neste checkpoint etter seg selv. Checkpoints er statiske, og dermed er komponenten barn av *BP_POI_Static*.

BP_POI_SBProjectile er POI-komponent for prosjektiler. Denne komponenten er kun opprettet og ikke videreutviklet, men er ment for å lage barneklasser til ulike typer prosjektiler som Setback-utviklere vil ha med i systemet. Prosjektiler er dynamiske, og dermed er komponenten barn av *BP_POI_Dynamic*.

BP_POI_SBPlayer er POI-komponent for spillerkarakterene i Setback. Det er disse som er i hovedfokus i spillet og dronene skal følge, så derfor er dette en barnekomponent av *BP_POI_Subject*. Denne komponenten har egne variabler for *BP_POI_SBCheckpoint*-komponenten til det neste checkpointet spilleren skal til, en teller for antall checkpoints spilleren har gått gjennom og en vektor for retningen til spillerens kamera/våpen.

3.3.3 Dronemester

Selve dronesystemet er styrt av en Actor kalt *BP_DroneMaster*. Denne dronemesteren har flere oppgaver:

- **Holde oversikt over interessepunkter** - Interessepunkter i form av POI-komponenter (Se delkapittel 3.3.2) vil over tid oppstå og forsvinne. dronemesteren skal til enhver tid ha en full oversikt over alle POI-komponentene som eksisterer i 3d-miljøet, og kategorisere disse.

  Dette gjøres ved at når et POI oppstår vil det kalle en funksjon hos dronemesteren som tar komponenten som argument, og putter en referanse til den i et array for den typen POI-komponent. Dersom et POI forsvinner, kaller det en annen funksjon som fjerner referansen til den komponenten fra arrayet for den typen komponent.

- **Kontrollere antall droner i 3d-miljøet** - Dronemesteren styrer hvor mange droner det eksisterer i 3d-miljøet ved hjelp av funksjoner for å opprette og fjerne dem, som kalles etter behov. Hvordan disse funksjonene brukes er opp til anvendere av pluginen. Det er en Integer-variabel (MaxDrones) som brukes for å begrense hvor mange droner som eksisterer til enhver tid. etter slik den generiske pluginen funker, vil dronemesteren hver gang et nytt subjekt-POI oppdages kjøre en funksjon som oppretter en ny drone så lenge antallet droner ikke overskrider MaxDrones-variabelen. Den nye dronen blir da gitt i oppgave å følge det nye subjektet.

- **Delegere subjekter til droner** - En drone som opprettes må få tildelt et POI det skal følge av Dronemesteren. Dette blir dronens subjekt, som her betyr at dronen skal følge og filme dette interessepunktet i hovedfokus. Dette gjøres ved å

kalle en funksjon dronen med et POI som argument. Denne funksjonen setter en variabel i dronen (AssignedPOI) til POI-en i argumentet.

Det er meningen at den som anvender pluginen skal bestemme når Dronemesteren tildeler nye subjekter til droner. I den generiske pluginen og den Setback-spesifikke tildeles droner det subjekt-POI-et som fører til opprettelsen av den.

- **Bestemme hvilke POI-er som er relevante i relasjon til *BP_POI_Subject*** - Alle Subjekt-POI-er har et array (*PrioritizedPOIs*) med andre POI-komponenter. POI-ene i dette arrayet er relevante for dronen å filme sammen med subjektet. Dronemesteren må i hvert tick oppdatere dette arrayet hos hver drone.

  Hvordan Dronemesteren velger om en drone skal prioritere et punkt er opp til utvikleren som anvender pluginen. I både den generiske versjonen av pluginen og den Setback-spesifikke versjonen går Dronemesteren gjennom alle Subjekt-POIer og ser på avstanden mellom hver av dem, og dersom de er innenfor en viss avstand fra hverandre puttes de i hverandres array.

  Dette arrayet brukes til å utregne et midtpunkt mellom POI-ene i arrayet som dronen forsøker å holde i kamerabildet sammen med sitt tildelte Subjekt-POI. 3.3.4

3.3.4 Kalkulering av midtpunkt av POI-er

Hver *BP_POI_Subject*-komponent har et array med andre POI-komponenter. Dronen som følger Subjekt-POI-et har som oppgave å forsøke å holde disse POI-ene i kamerabildet sitt. For å forenkle dette problemet blir det utregnet ett enkelt midtpunkt basert på alle disse POI-ene som dronen heller forholder seg til.

For å regne ut midtpunkt er metode for å regne ut geometrisk midtpunkt brukt. Med et sett S av n antall punkter regnes geometrisk midtpunkt M slik:

$$S = \{(x_1, y_1, z_1), (x_2, y_2, z_2), ...(x_n, y_n, z_n)\}$$

$$M = (\overline{x}, \overline{y}, \overline{z}) = \frac{1}{n} * \left(\sum_{i=1}^{n} x_i, \sum_{i=1}^{n} y_i, \sum_{i=1}^{n} z_i \right) \tag{3.1}$$

Med denne metoden blir alle punktene behandlet likt, altså de trekker midtpunktet like mye til seg. I tilfellet for dronesystemet er ikke dette hensiktsmessig. For eksempel kan det ønskes at en spiller skal trekke til seg midtpunktet mer enn et prosjektil.

Dette løses ved å kalkulere en egen faktor for hvert punkt. Alle punkt multipliseres med faktoren sin. n-dividenden, som her er summen av antall punkt, må heller bli summen av alle faktorene. Midtpunkt M Regnes da ut slik basert på punktene S og faktorene F:

$$
\begin{aligned}
S &= \{(x_1, y_1, z_1), (x_2, y_2, z_2), ...(x_n, y_n, z_n)\} \\
F &= \{f_1, f_2, ...f_n\} \\
M = (\overline{x}, \overline{y}, \overline{z}) &= \frac{1}{\left(\sum_{i=1}^{n} f_i,\right)} * \left(\sum_{i=1}^{n} x_i f_i, \sum_{i=1}^{n} y_i f_i, \sum_{i=1}^{n} z_i f_i\right)
\end{aligned} \tag{3.2}
$$

Eksempel for utregning av faktor

Det er meningen at utregning av den multiplikative faktoren skal defineres av den som utnytter seg av pluginen for hver enkelt type POI, slik at de selv kan kontrollere hvordan midtpunktet kalkuleres.

Et eksempel på en utregning av faktor er laget i den Setback-spesifikke pluginen for *BP_POI_SBPlayer*. En annen spiller er mer relevant å følge med på jo nærmere de er spilleren dronen følger. De er også mer relevante dersom en av spillerene sikter på den andre, for da er sannsynligheten for at de påvirker hverandre større.

Her blir da faktoren basert på avstand fra spilleren som dronen følger, samt prikkproduktet mellom spillers siktretning og retning mot annen spiller (her utregnes prikkproduktet for begge spillere, og det største velges). Ligningen blir da slik:

$$
\begin{aligned}
a &= \text{avstand mellom spiller-POI-ene} \\
m &= \text{maks avstand for POI som inkluderes} \\
V_a &= [0, m] \\
d &= \text{Det største dot-produktet} \\
V_d &= [-1, 1] \\
f(a, d) &= 2 * \left(\frac{(d+1)^5}{4}\right) * \left(\frac{m-a}{m}\right)^2
\end{aligned} \tag{3.3}
$$

Faktoren vil da se slik ut:

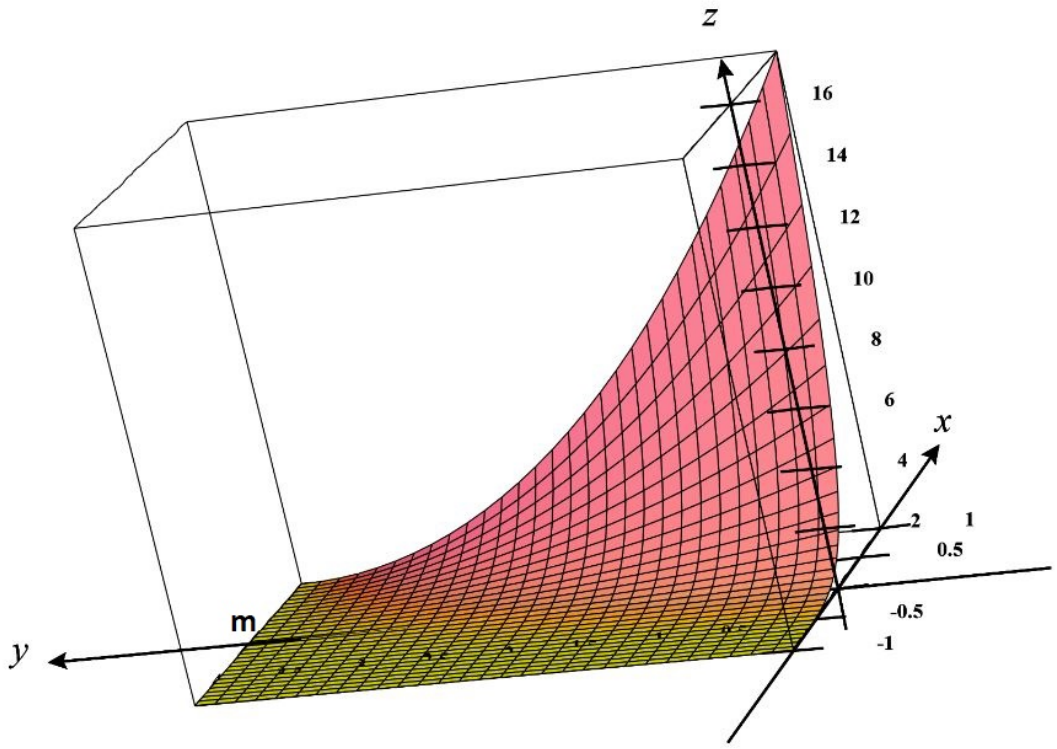


Figure 3.8: Faktor for utregning av midtpunkt, for setback-spiller-POI. Her er a langs y-aksen og d langs x-aksen.

3.3.5 Komposisjonskomponent

En drone har en komposisjonskomponent som bestemmer hva slags komposisjonsrutenett dronen skal forsøke å putte sitt tildelte POI og det utregnede midtpunktet i. Denne komponenten består av to float-tallvariabler, HeightMidpart og WidthMidpart. Disse variablene fungerer slik:

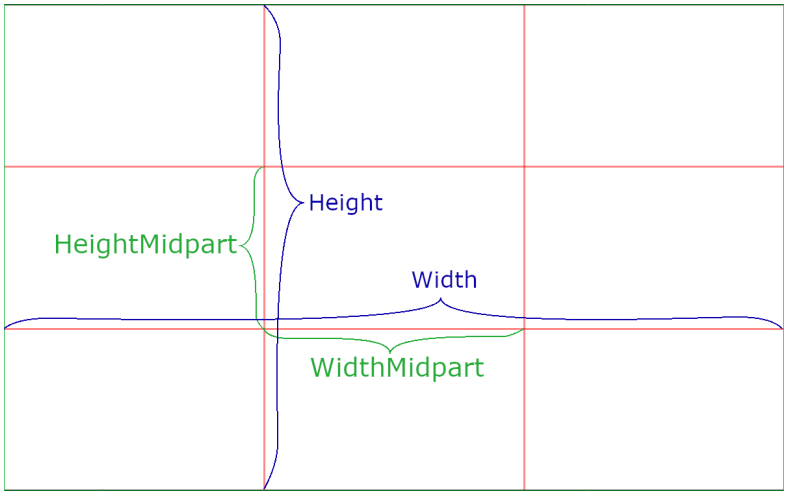


Figure 3.9: Hvordan variablene i komposisjonskomponenten forholder seg til kamerabildet.

Variablene har verdi mellom 0 og 1, etter hvor store de er i forhold til Heigth og Width. Ved å for eksempel definere HeightMidpart og WidthMidpart som 1/3, vil man få et tredelingsperspektiv (slik som i figur 3.9). Ved å for eksempel sette HeightMidpart som 1/3 WidthMidpart som 1/2, vil komposisjonsrutenettet se slikt ut:

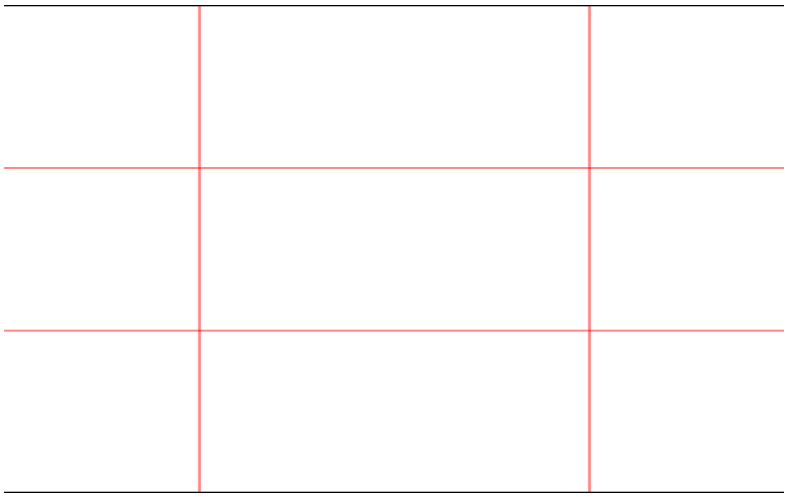


Figure 3.10: Komposisjonsrutenett for HeightMidpart = 1/3 WidthMidpart = 1/2.

3.3.6 Utregning av destinasjon

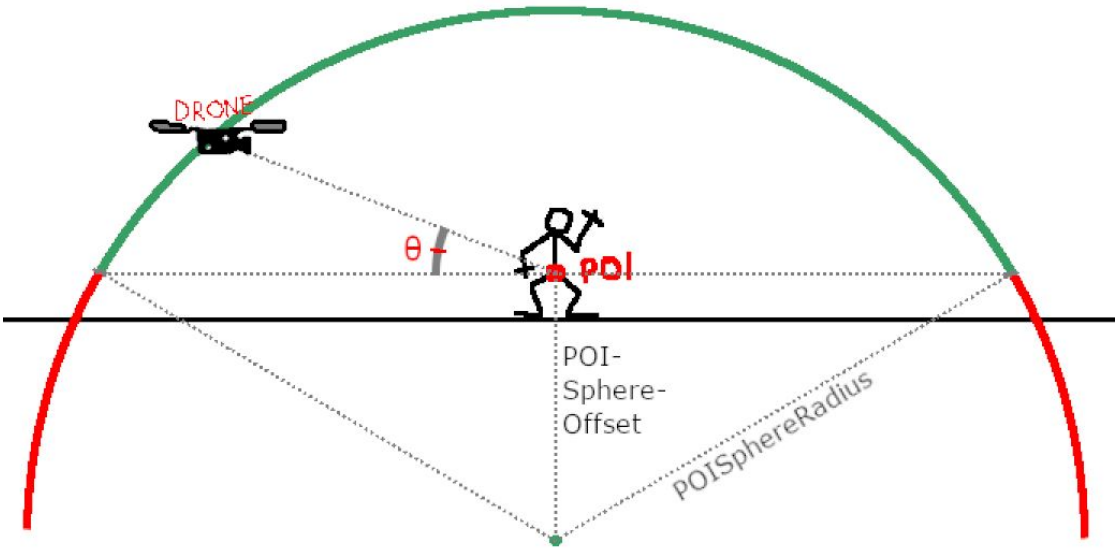


Figure 3.11: Område rundt tildelt interessepunkt hvor dronen vil oppholde seg sett fra siden.

Destinasjonsområdet

Dronen vil alltid være tildelt et POI som den skal følge. Området dronen vil bevege seg mot og prøve å oppholde seg er definert av to variabler som er mulig å justere etter behov. Den ene er en radius, POISphereRadius, og den andre er POISphereOffset. Et punkt blir definert en avstand lik POISphereOffset under POI-et dronen følger, og dronen vil oppholde seg med en avstand lik POISphereRadius fra det definerte punktet. Dermed vil destinasjonspunktet alltid bli definert som et punkt i skallet på en kule definert av disse variablene, som blir det grønne og røde området på figur 3.11.

Vertikal vinkel

Dronen har en variabel for vertikal vinkel mellom seg selv og bakkeplanet relativt til sitt POI (θ i figur 3.11). Ved å endre på denne variabelen vil posisjonen til dronen varieres vertikalt langs destinasjonsområdet, og dronen vil kunne se sitt POI fra forskjellige vinkler.

Med den vertikale vinkelen kan dronen regne ut i hvilken breddegrad langs destinasjonsområdet den vil oppholde seg i. Dette gjøres ved hjelp av cosinussetningen:

$a^2 = b^2 + c^2 - 2ac * Cos(A)$

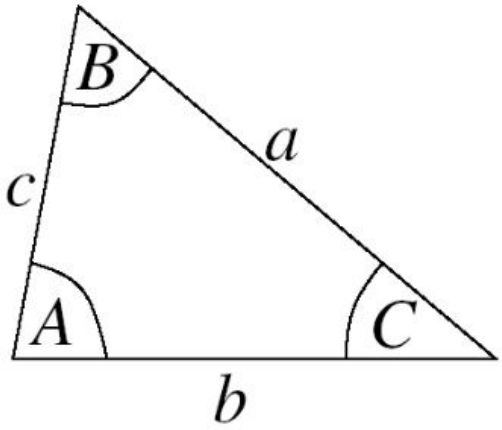


Figure 3.12: variabler i Cosinussetningen.

I dronens tilfelle må vi endre på setningen:

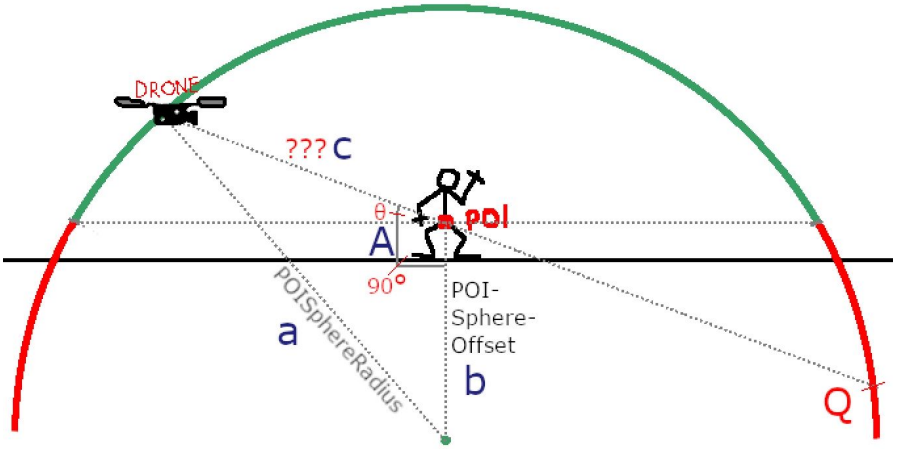


Figure 3.13: Cosinussetningen i forhold til dronen.

Sett etter figur 3.13 er variablene a, b og A kjente ($a = POISphereRadius$, $b = POISphereOffset$, $A = \theta + 90°$), mens c må utregnes. Her får vi en andregradsligning, som regnes ut med abc-formelen (Variabler i abc-formelen kalt x, y og z for å skille med variabler i cosinussetningen):

$$
\begin{aligned}
a^2 &= b^2 + c^2 - 2bc * Cos(A) \\
c^2 - c(2b * Cos(A)) + (b^2 - a^2) &= 0 \\
x &= 1 \\
y &= 2b * Cos(A) \\
z &= b^2 - a^2 \\
c &= \frac{-y \pm \sqrt{y^2 - 4xz}}{2z}
\end{aligned}
\tag{3.4}
$$

Her velges verdien for c basert på den satte vertikale vinkelen. Dersom vinkelen er positiv vil destinasjonen være langs den grønne seksjonen av destinasjonsområdet på figur 3.13, og dermed må den laveste verdien for c velges. Den høyeste verdien vil være avstanden mellom POI og punktet Q. Dersom den vertikale vinkelen er negativ, vil destinasjonen være langs den røde seksjonen av destinasjonsområdet, og da velges den største verdien for c. Med den utregnede verdien for c kan dronen regne ut en posisjon som både er i destinasjonsområdet og forholder seg til den vertikale vinkelen dronen har bestemt å holde seg i.

Opprettholde sikt

Dronen sjekker ved hvert tick om den har sikt til subjektet ved å benytte Line Trace for Objects. Dersom Line Tracen oppdager terreng mellom dronen og subjektet øker den vinkelen til destinasjonen som vist i figur 3.14 til et punkt hvor dronen vil ha sikt igjen. Den satte verdien for θ forblir den samme, mens dronen utnytter en midlertidig vinkel θ_1 som er tilpasset siktblokkeringer.

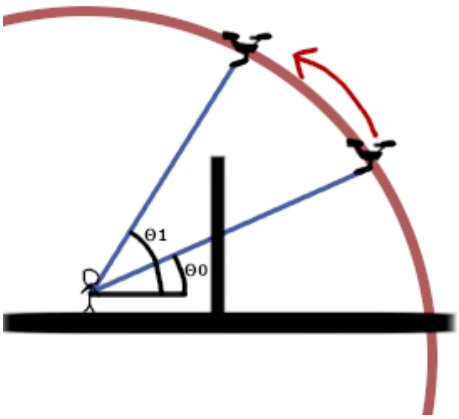


Figure 3.14: Diagram som viser hvordan en drone beveger seg for å opprettholde sikt til subjektet

Destinasjon utregnet langs bestemt breddegrad

Når dronen har regnet ut i hvilken breddegrad destinasjonen skal ligge, regner den ut hvor langs breddegraden destinasjonen kan settes.

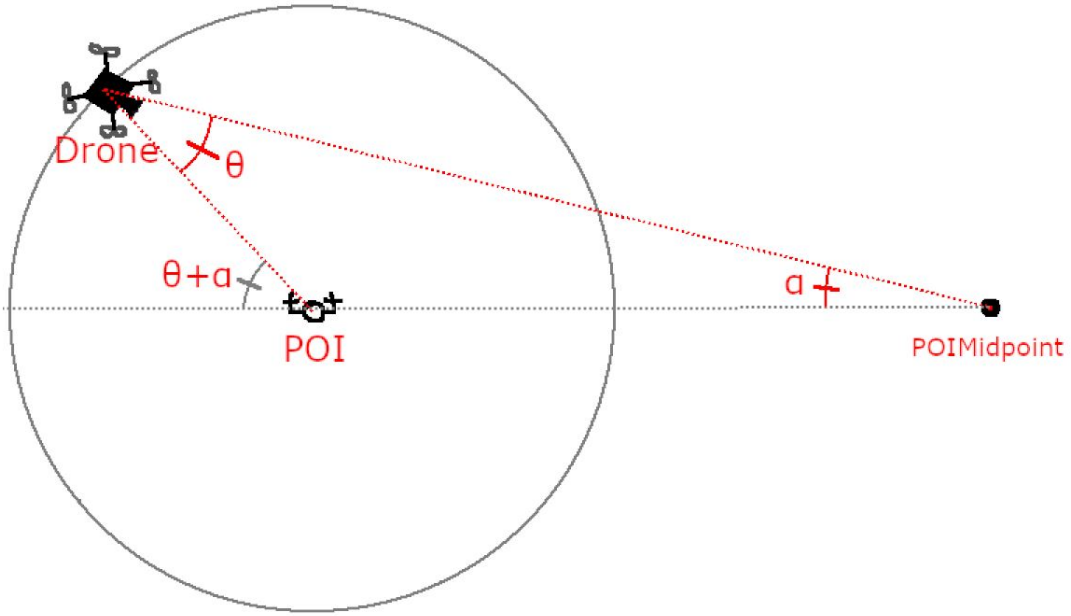


Figure 3.15: Dronen sett ovenfra, sett i forhold til sitt POI og den utregnede POIMidpoint.

Dronen har utregnet et POIMidpoint som det skal forholde seg til sammen med sitt tildelte POI. Basert på komposisjonen og FOV-en til dronen. Basert på komposisjonskomponenten (ref. delkapittel 3.3.5) sin MidpartWidth-variabel og dronens vertikale FOV regner dronen ut den maksimale vertikale vinkelen θ:

$$\theta = MidpartWidth * VerticalFOV \tag{3.5}$$

hvor dronens destinasjon alltid skal ligge et sted på breddegraden hvor vinkelen mellom sitt POI og POIMidpoint er likt eller mindre enn θ. Basert på dette kan dronen regne ut vinkelen α, og dermed finne ut grensene på breddegraden som den må holde seg innenfor, som blir $\theta + \alpha$.

Destinasjonen langs breddegradssirkelen blir til enhver tid basert på hvor dronen er.

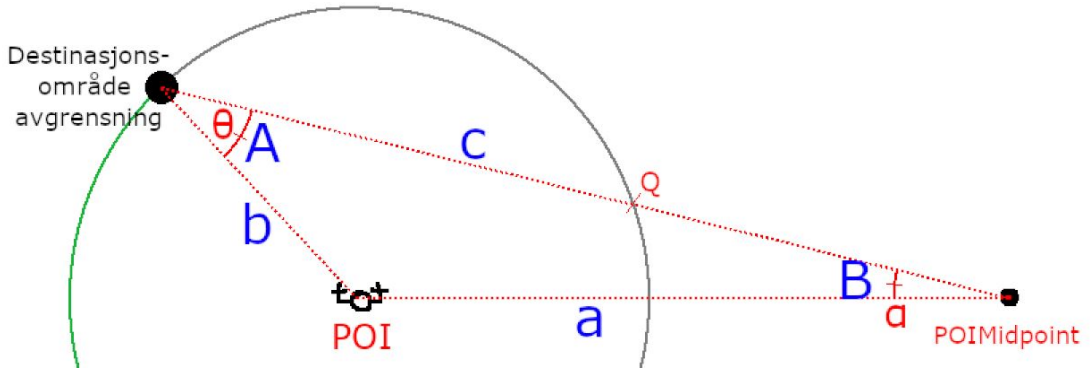


Figure 3.16: Variabler i cosinussetningen for å regne ut α.

For å regne ut vinkel *alpha* utnyttes igjen cosinussetningen. I dette tilfellet er A, a og b kjent, og vi vil regne ut B. For å regne ut B får vi formelen:

$$\begin{aligned} b^2 &= a^2 + c^2 - 2ac * cos(B) \\ B &= acos(\frac{b^2 - a^2 - c^2}{-2ac}) \end{aligned} \tag{3.6}$$

Før B kan regnes ut, må c regnes ut. cosinussetning for dette vil bli slik:

$$\begin{aligned} a^2 &= b^2 + c^2 - 2bc * Cos(A) \\ c^2 - c(2b * Cos(A)) + (b^2 - a^2) &= 0 \end{aligned}$$

$$\begin{aligned} x &= 1 \\ y &= 2b * Cos(A) \\ z &= b^2 - a^2 \\ c &= \frac{-y \pm \sqrt{y^2 - 4xz}}{2z} \end{aligned} \tag{3.7}$$

Her velges den største verdien for c, fordi den lavere verdien vil være for tilfellet hvor b og c møtes i punktet Q.

Basert på avgrensningen utregnet her, vil destinasjonen bestemmes i forhold til hvor dronen ligger i forhold til avgrensningen:

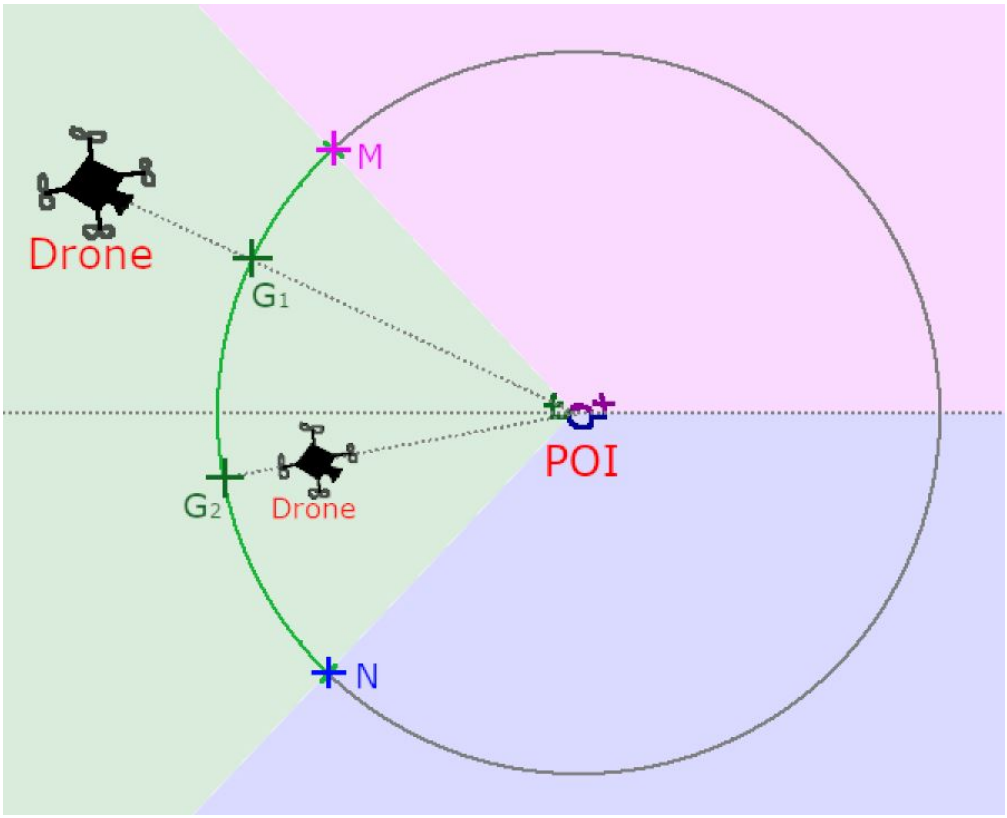


Figure 3.17: Hvor destinasjonen plasseres i forhold til dronens posisjon.

Dersom dronen oppholder seg i det øvre rosa området, settes destinasjonen i punktet M, som er på enden til avgrensningen. Dersom dronen oppholder seg i det nedre blå området, settes destinasjonen i punktet N, som er på den motsatte enden til avgrensningen. Dersom dronen ligger i det grønne området, vil destinasjonen settes i krysset mellom avgrensningen og vektoren mellom dronen og dens POI (her sett to eksempler, G_1 og G_2).

Kollisjonshåndtering mot droner og omgivelser

Når droner kommer innenfor en bestemt avstand fra en annen, vil de da bevege seg fra hverandre. Dette blir gjort på følgende måte:

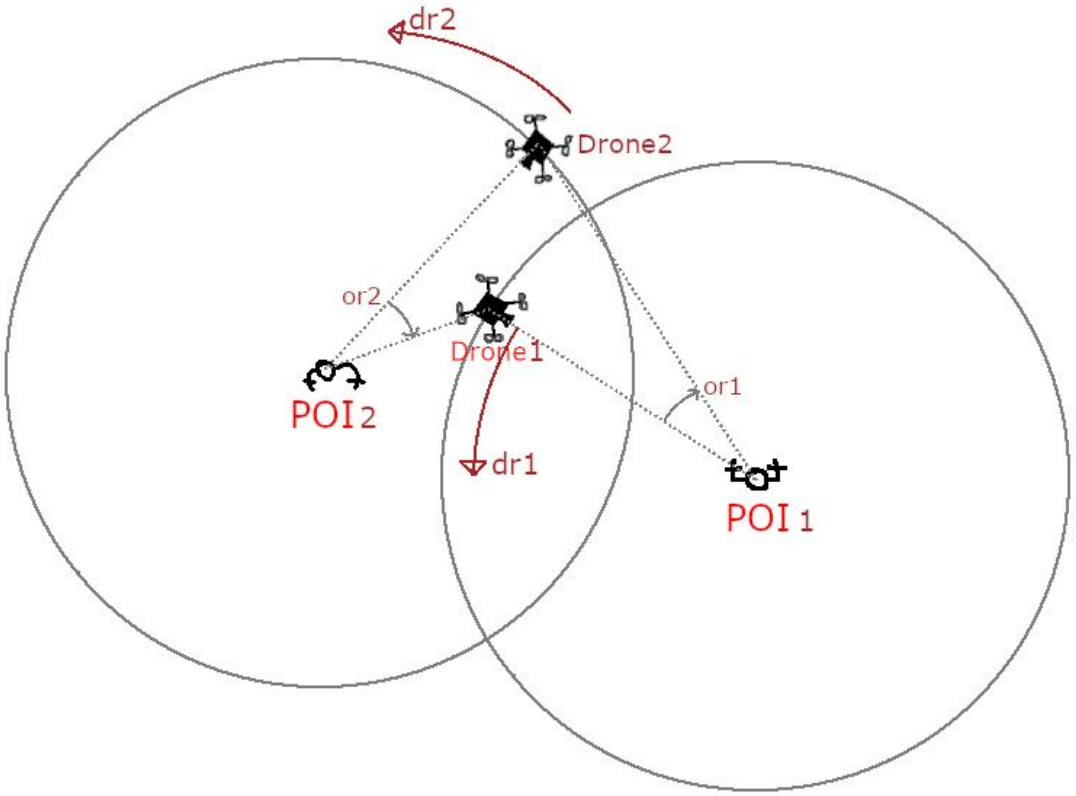


Figure 3.18: Hvordan droner unngår hverandre.

Figur 3.18 er et eksempel hvor to droner er for nære hverandre, innenfor en avstand L_d. I et slikt tilfelle vil begge dronene gjøre tiltak for å unngå den andre. Den enkelte dronen vil da finne ut retningen *or* (hvilken retning den andre dronen er i forhold til dronens POI), og roterer destinasjonspunktet til dronen i retning *dr* langs destinasjonsbreddegraden.

Hvor mye destinasjonspunktet roteres er proporsjonalt med avstanden til den andre dronen. Rotasjonsstørrelsen vil altså avta jo større avstanden blir, helt til avstanden blir lik L_d eller større.

For å unngå omgivelser blir en lignende metode brukt:

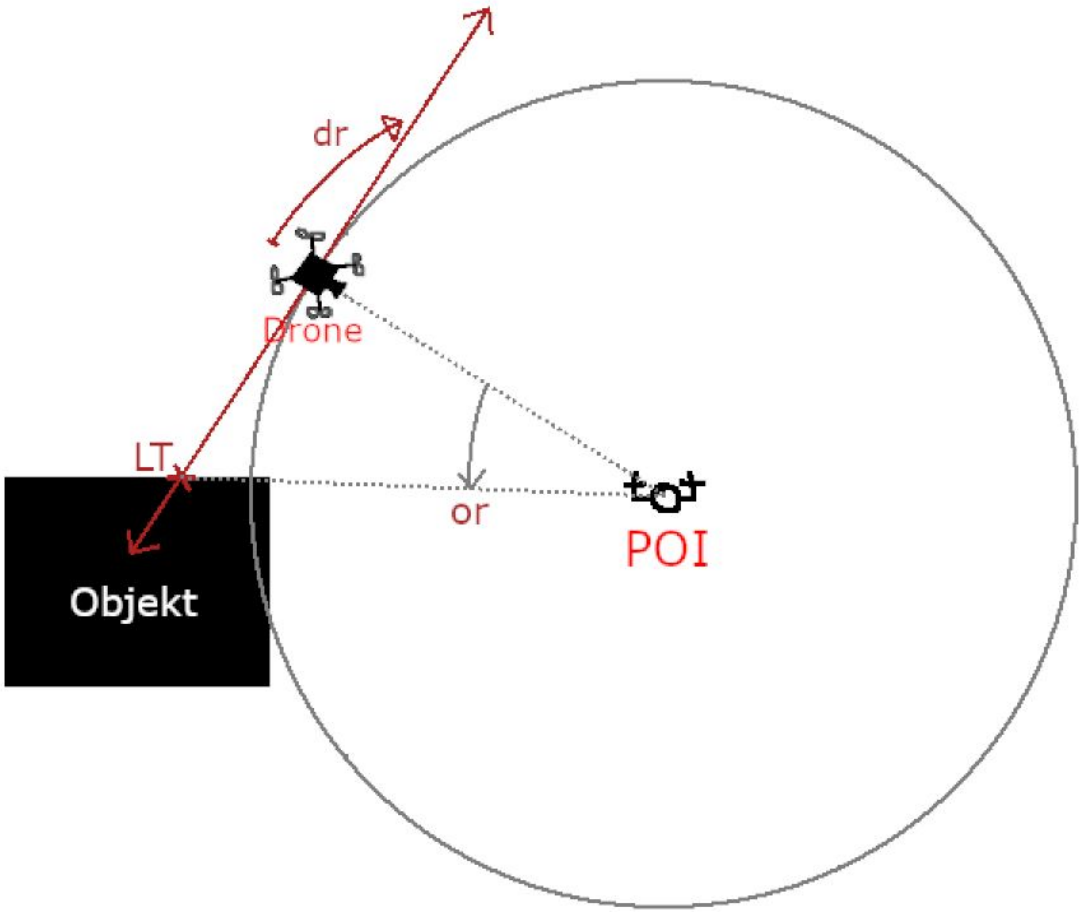


Figure 3.19: Hvordan droner unngår omgivelser.

Figur 3.19 er et eksempel hvor en drone oppdager omgivelser den må unngå. Dronen bruker Line Trace For Objects ut fra begge sidene sine i en definert lengde L_o. Dersom dronen oppdager et objekt (LT) vil dronen rotere destinasjonspunktet i motsatt retning av Line Tracen langs destinasjonsbreddegraden. Her er rotasjonsstørrelsen proporsjonal med avstanden mellom destinasjonen og objektet, og rotasjonsstørrelsen avtar helt til avstanden blir lik L_o eller større.

Alle rotasjonene som regnes ut for dronen adderes sammen så dronen får en total rotasjon før destinasjonen roteres rundt subjekt-POI-et. Dermed vil flere droner nær hverandre spre seg utover.

3.3.7 Dronebevegelse

Dronen vil til enhver tid forsøke å bevege seg mot destinasjonen. Dette gjøres ved hjelp av en node i UE4 som heter "Move To Location". AI-kontrolleren i dronen bruker denne

for å bevege dronen mot et punkt. Grunnet hvordan denne noden fungerer er det ikke hensiktsmessig å sette punktet dronen beveger seg mot til den utregnede destinasjonen.

En utfordring er hvordan dronen beveger seg mot punktet. Med "Move To Location" vil dronen rette bevegelsen sin nokså direkte mot punktet den vil mot, som kan føles unaturlig siden destinasjonen ofte kan være statisk og endre seg fort, så dronen kan få brå fartsendringer som føles unaturlig for en drone. Det er mulig å endre hvor fort dronen snur retningen sin med en variabel kalt "Turning Boost", men dersom denne senkes blir det vanskelig å kontrollere bevegelsen siden dronen deakselererer mindre når den nærmer seg punktet, og heller flyr videre før den retter bevegelsen mot punktet igjen. Denne utfordringen ble løst ved å bruke et annet punkt dronen beveger seg mot. Dette punktet interpoleres mot destinasjonen ved bruk av "VInterpTo", som gjør at punktet dronen følger vil bevege seg jevnt mot destinasjonen, og som følge vil også dronen bevege seg mot destinasjonen i en jevnere bane.

En annen utfordring er at med "Move To Location" vil dronens hastighet synke jo nærmere punktet den beveger seg mot. Som følge vil dronen bevege seg veldig saktenår punktet er nærme, og når destinasjonen flytter på seg vil da dronen ende opp med å henge langt etter, som gir dårlig sikt.

Dette er blitt løst ved å øke avstanden til punktet dronen beveger seg mot. Et nytt punkt plasseres langs linjen som går gjennom dronen og det interpolerte destinasjonspunktet. Hvor på linjen det plasseres er avhengig av avstanden mellom dronen og den interpolerte destinasjonen. Det nye punktet plasseres med en avstand fra dronen lik avstanden til det interpolerte punktet multiplisert med en faktor, i samme retning som det interpolerte punktet.

Faktoren er ment for å kunne justeres, men den faktoren vi kom fram til som gir en mer naturlig dronebevegelse er bestemt av denne funksjonen:

$$\begin{aligned} x &= \textbf{avstand mellom drone og interpolert destinasjon} \\ f(x) &= log_{10}(x+1) \end{aligned} \tag{3.8}$$

som ser slik ut:

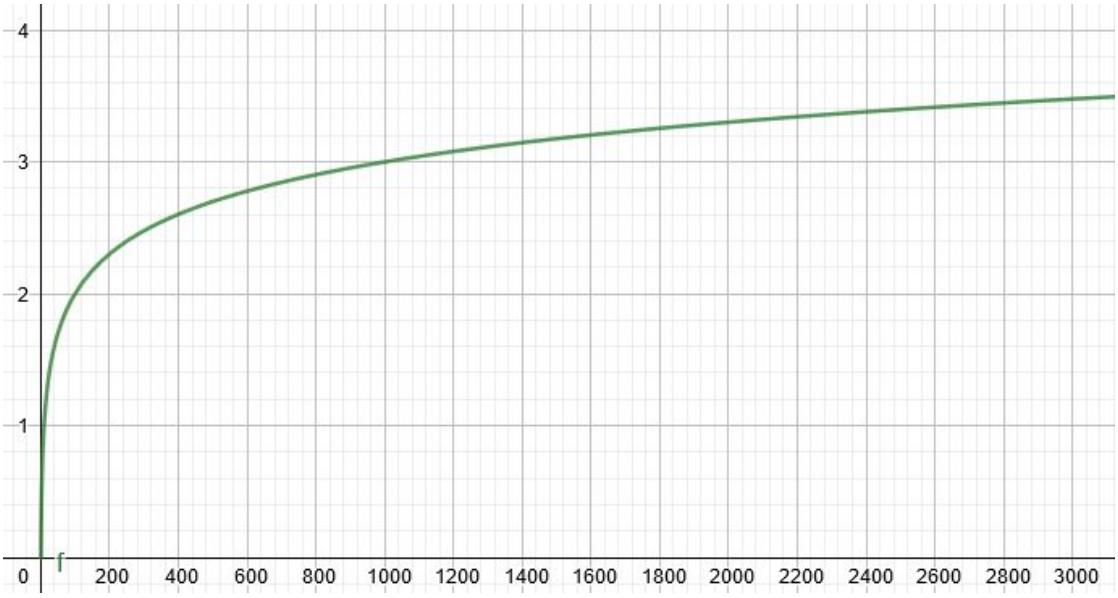


Figure 3.20: Faktoren som multipliseres med avstand mellom drone og interpolert destinasjon.

Avstand til det nye punktet i forhold til den interpolerte destinasjonen vil da se slik ut:

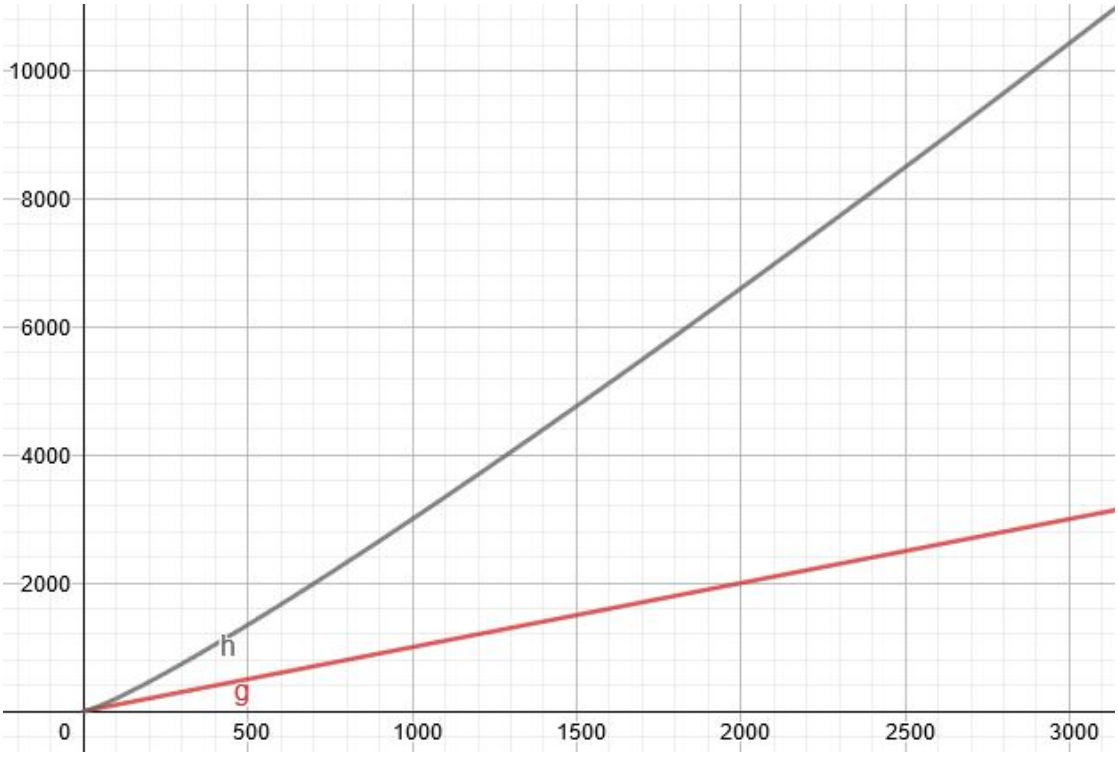


Figure 3.21: g: Avstand mellom drone og interpolert destinasjon.
h: Avstand til det nye punktet.

Den nye avstanden blir større enn den originale. Dermed vil dronen bevege seg med større hastighet mot den interpolerte destinasjonen siden den følger det nye punktet istedet.

3.3.8 Komposisjon

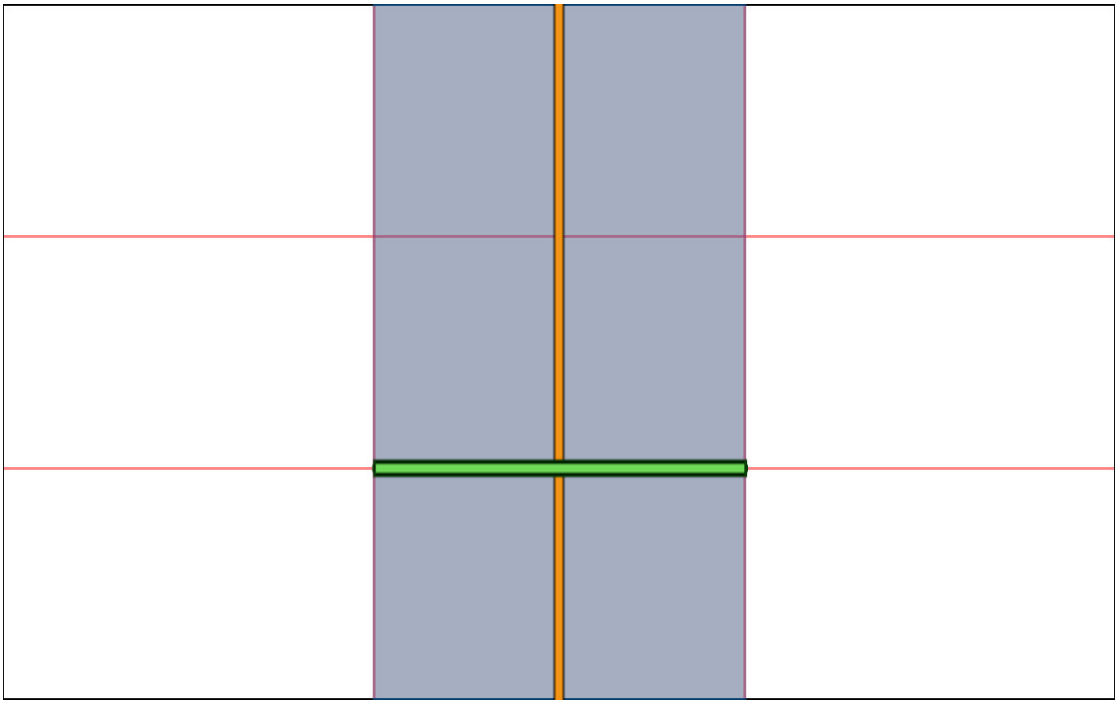


Figure 3.22: Den visuelle komposisjonen og hvordan dronen forholder seg til den.

Dronesystemet bruker i hovedsak tredelingsregelen (ref. kap. 2.1.1) som komposisjonell regel, men har mulighet til å anvende andre komposisjoner etter Komposisjonskomponenten (ref. kap. 3.3.5). Den vil alltid holde subjekt-POI-et langs den nedre grønne linjen i figur 3.22), mens dronen vil bevege seg for å holde POI-midtpunktet innenfor det blå området.
Dette fungerer ved å legge til en komposisjonsrotasjon av Rotation-datatypen (ref. 3.1.4 - subseksjon "Rotation-datatype") på rotasjonsvariabelen fra dronen til spilleren. For å få Subjekt-POI-et langs den grønne linjen settes pitch-rotasjonen til komposisjonsrotasjonen til $\frac{MidpartHeight * VerticalFOV}{2}$ av den vertikale FOV-en.

For å tilpasse komposisjonsrotasjonens Yaw ser dronen om den vertikale vinkelen mellom subjekt-POI-et og POI-midtpunktet, β, er større enn maksvinkelen θ (ref. 3.5). Dersom β er større, settes komposisjonsrotasjonens Yaw til $\frac{MidpartWidth * HorizontalFOV}{2}$ i retning mot midtpunktet. Dersom β er lik eller mindre enn θ, settes Yaw til $\frac{\beta}{2}$. Når β da er mindre vil subjektet og midtpunktet havne med lik avstand til den oransje midtlinjen på figur 3.22 på motsatt side av hverandre.

3.3.9 Droneverdi

Hver drone regner ut en verdi som skal fortelle hvor interessant den er å se ut av i forhold til de andre dronene på banen. Hvordan dette utregnes er ment for den enkelte utvikleren å bestemme.

I både den generiske versjonen og Setback-versjonen av pluginen går dronen gjennom hvert subjekt-POI utenom det den er tildelt, og sjekker om de er innenfor kamerabildet og om det er fri siktlinje til dem. For hvert subjekt det er kamerasikt til, øker dronen siktsverdien sin med subjektets verdi (Value-variabelen i POI-komponenter) summen av verdiene blir variabel v.

Det er begrenset hvor mange bevegende objekter menneskeøyet kan følge med på. En god drone burde derfor ikke ha for mange objekter å følge med på. I følge Gestaltteori (ref. 2.1.2) vil objekter som er nær hverandre oppfattes som en gruppe istedenforindividuelle figurer. Vi bruker standardavvik for å finne ut hvor nærme objektene er hverandre. Standardavvik $\overline{sd}$ regnes ut slik utifra settet med subjekter som dronen sikt til i synsfeltet V_n:

$$\overline{sd} = \frac{1}{n} * \sum_{i=1}^{n} \left(V_i(x)^2 + V_i(y)^2 + V_i(z)^2 \right) \tag{3.9}$$

Ut ifra verdien v og standardavviket $\overline{sd}$ regnes den endelige siktsverdien $ViewValue$ ut slik:

$$ViewValue = v + 100 * \frac{v}{\sqrt{\overline{sd}}} \tag{3.10}$$

3.4 Optimaliseringsproblem

3.4.1 Posisjonering

I et forsøk på å finne en generalisert måte å posisjonere dronene på er det lagd et separat optimaliseringsproblem. Den følgende algoritmen er en modifisert versjon av "Reynolds Flocking" (ref 2.2.1), hvor hver drone i tar hensyn til alle punkter (droner j og interessepunkter k) innen for en gitt radius:

$$D_i = \{drone\ j : j \neq i \wedge \parallel r_{ij} \parallel < r_j^{max}\} \tag{3.11}$$

D_i er et sett av alle droner som er innenfor drones oppfatningsradius r_j^{max}, der r_{ij} er differansen mellom posisjonsvektorene til drone i og drone j, og $\parallel r_{ij} \parallel$ er den euklidiske lengden til vektoren.

$$P_i = \{poi\ k : \parallel r_{ik} \parallel < r_k^{max}\} \tag{3.12}$$

P_i er drone i sitt sett med alle interessepunktene innenfor oppfatningsradiusen r_k^{max}.

Separeringsfaktoren blir brukt til å unngå andre droner slik at de ikke kommer i veien for hverandre, og holder avstand fra interessepunktene. Dette er for å få en synsvinkel inn mot intressepunktene slik at hver drone skal få med seg hva som skjer bak interessepunktet, og lar andre droner holde seg unna slik at de ikke skulle fly i veien og skape problemer for seeropplevelsen. Separeringsfaktoren for en gitt drone er da:

$$a_i^{sep} = -\frac{x^{sep}}{|D_i| + |P_i|}\left(\sum_{j \in D_i} \frac{r_{ij}}{\parallel r_{ij} \parallel^2}\right)\left(\sum_{k \in P_i} \frac{r_{ik}}{\parallel r_{ik} \parallel^3}\right) \tag{3.13}$$

der x^{sep} er den multiplikative faktoren, som bestemmer styrken på regelen. Separeringsfaktoren er basert på avstanden mellom punktene.

Samkjøring gir dronen en bevegelsesretning som tilsvarer bevegelsen til nærliggende punkter slik at dronen kan holde følge med interessepunktene og fly i formasjon med de andre dronene. Samkjørings faktoren blir da:

$$a_i^{sam} = \frac{x^{sam}}{|D_i| + |P_i|}\left(\sum_{j \in D_i} v_j\right)\left(\sum_{k \in P_i} v_k\right) \tag{3.14}$$

der x^{sam} er en multiplikativ faktor. Her finner vi gjennomsnittlig bevegelsesvektor for alle punkter innenfor interesseradiusen og finner gjennomsnittet.

Migreringsfaktoren blir den som passer på at dronen holder seg i nærheten av interesse punktene. Siden den bare har fokus på punkter som er innefor oppfatningsradiusen, trenger ikke dronen tenke på hva de andre intressepunktene, gjør da de er så langt borte at det ikke er gunstig å filme. Den trenger heller ikke tenke på hva andre droner som er langt unna gjør da dette ikke skaper problem for seer opplevelsen. Den blir da definert som:

$$a_i^{sen} = \frac{x^{sen}}{|P_i|} \sum_{k \in P_i} r_{ik} \tag{3.15}$$

med x^{sen} som er med på bestemme styrken på regelen.

3.4.2 Simulasjon

For å simulere hvordan dronen burde posisjonere seg ble den modifiserte flokkalgoritmen kjørt i Python. I denne simulasjonen forholder dronene seg til dynamiske interessepunkter. Her beveger interessepunktene seg rundt i rommet i en samlet klynge, med en justerbar avstand fra hverandre. Dette for å simulere bevegelsen som spillerne har i Setback, og hvordan dronene burde oppføre seg i forhold. Dette er en simplifisert simulasjon, da Setback har terreng som gir spillerne bevegelse i høyden og dronene hindringer å ta hensyn til samt intelligente uavhengige spillere som skal filmes.

I denne simulasjonen kan vi også prøve å regne ut hvilket interessepunkt som er mest interessant å se på. Vi kan også prøve å finne hvilken drone som systemet bør se fra for å få med seg dette punktet og mye av de andre punktene. I Setback kan det for eksempel være interessant å se på spilleren som leder kappløpet, så programmet finner derfor det interessepunktet som leder klyngen av interessepunktene i den retningen flokken beveger seg i. For å velge en drone som systemet burde se utifra, finner systemet den dronen som ligger foran klyngen og kan se tilbake på dette (hoved)interessepunktet og få med seg flest mulige interessepunkt som ligger bak.

Både interessepunktene og dronene er definert som underklasser som blir arvet fra en punkt-klasse. Denne punkt-klassen inneholder tre vektorer som inneholder informasjon om punktets oppførsel i rommet. Posisjon, Fart og Akselerasjon. Et hovedprogram lager et gitt antall interessepunkter og droner, og plasserer de i rommet, så kjører en oppdatering på alle punktene med logikk de har i sin egen klasse om hvordan de skal endre posisjon i rommet.

4 Resultater

Her gjøres det rede for resultatene som følger av arbeidet gjort under prosjektperioden, i henhold til forskningsspørsmålene (ref 1.2), de funksjonelle kravene som ble definert i visjonsdokumentet, og arbeidsprosessen.

4.1 Vitenskapelige resultater

Resultatene i denne seksjonen er direkte knyttet til forskningspørsmålene som ble stilt i kapittel 1.2.

4.1.1 Interessepunkter

Det første forskningsspørsmålet ble definert som "Hvordan kan et system med kameradroner forholde seg til flere dynamiske interessepunkter?". Slik dette produktet tilnærmer seg denne utfordringen er ved å anvende det utarbeidede systemet for POI-komponenter. Med disse er det mulig å definere objekter som interessepunkter ved behov. Det er også mulig å lage ulike typer POI-komponenter for ulike typer objekter, som tillater at de kan behandles annerledes etter hva slags type interessepunkt det er.

I dronemesteren holdes det oversikt over alle interessepunkter og droner. Med denne gjøres da samhandling mellom interessepunkter og droner oversiktlig og kontrollert. Hvilke droner som følger hvilke interessepunkter, samt hvilke interessepunkter som bør filmes sammen, kontrolleres da her. Ved å anvende disse prinsippene, er det mulig for systemet å forholde seg til hvilket som helst sett med interessepunkter.

4.1.2 Oversikt

Et av forskningsspørsmålene var "Hvordan kan systemet skape en helhetlig oversikt over interessepunktene?". I selve pluginen er ikke dette tatt for nøye stilling til, annet enn at det er lagt til rette for anvendere å videreutvikle systemet til å tilpasses sitt prosjekt, ved hjelp av å definere interessepunkter og kontrollere hvilke som filmes. Denne utfordringen er heller forsøkt å svare på ved hjelp av et optimaliseringsproblem utviklet ved hjelp av en simulering av et 3d-miljø i Python.

Simulasjon

Python-programmet simulerer punktene (POI og dronene) og finner posisjonsvektorene lokasjon, fart og akselerasjon over tid. Dette for å finne posisjoneringen til dronene slik at de kan få oversikt over alle interessepunktene. Siden alle punktene er vektorer kan det

enkelt vises i en tre-dimensjonell graf. Her er flere vinkler for å vise dronenes oversikt over interessepunktene.

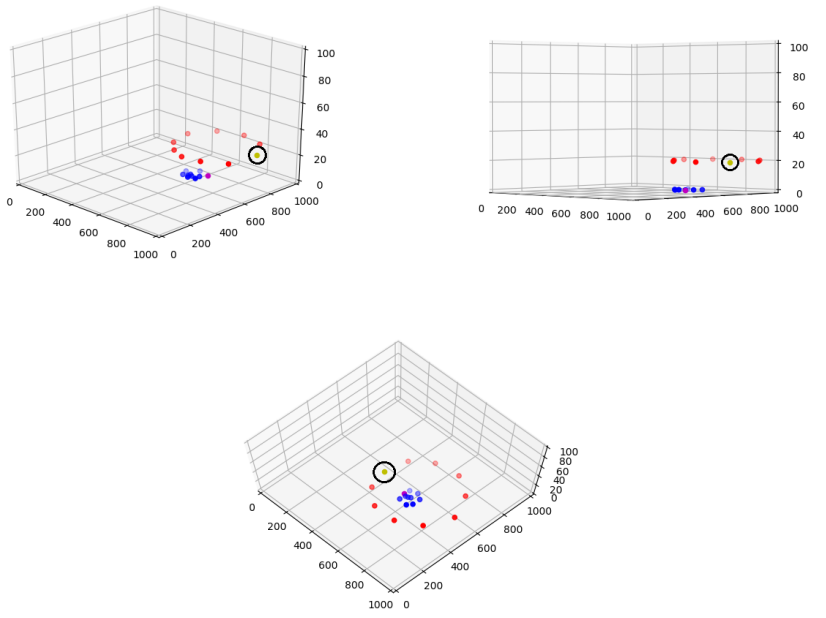


Figure 4.1: Droner i rødt, POI i blått. Det lilla punktet leder interessepunkene, og det gule punktet sirklet i svart er dronen med "best" oversikt

Dette er en simulasjon i sanntid hvor dronene må tilpasse seg de dynamiske interessepunktene for å opprettholde en god oversikt, slik som de måtte ha gjort i Setback. For å visualisere dette på best måte er det laget en video av simuleringen (Gram et al., 2020b). I videoen kan man se at dronene reagerer på hvordan miljøet enderer seg over tid. Det gule punktet hopper fra drone til drone ettersom programmet velger hvilken drone systemet burde se utifra (se 3.4.2).

4.1.3 Visuell Komposisjon

Et forskningsspørsmål var formulert som "Hvordan kan en kameradrone tilpasse seg et sett med interessepunkter for å oppnå en visuell komposisjon?". Dette ble løst ved å definere et subjektpunkt og et midtpunkt av relevante interessepunkter, og å plassere disse to punktene i forhold til en definert komposisjon. I denne seksjonen vises det skjermbilder fra et UE4 template prosjekt. Det vises ett bilde med debug-points og tredelingsrutenett, og ett bilde uten. Det turkise debug-pointet viser posisjonen til

dronens hovedsubjekt. Det lilla punktet viser posisjonen til midtpunktet dronen regner ut.

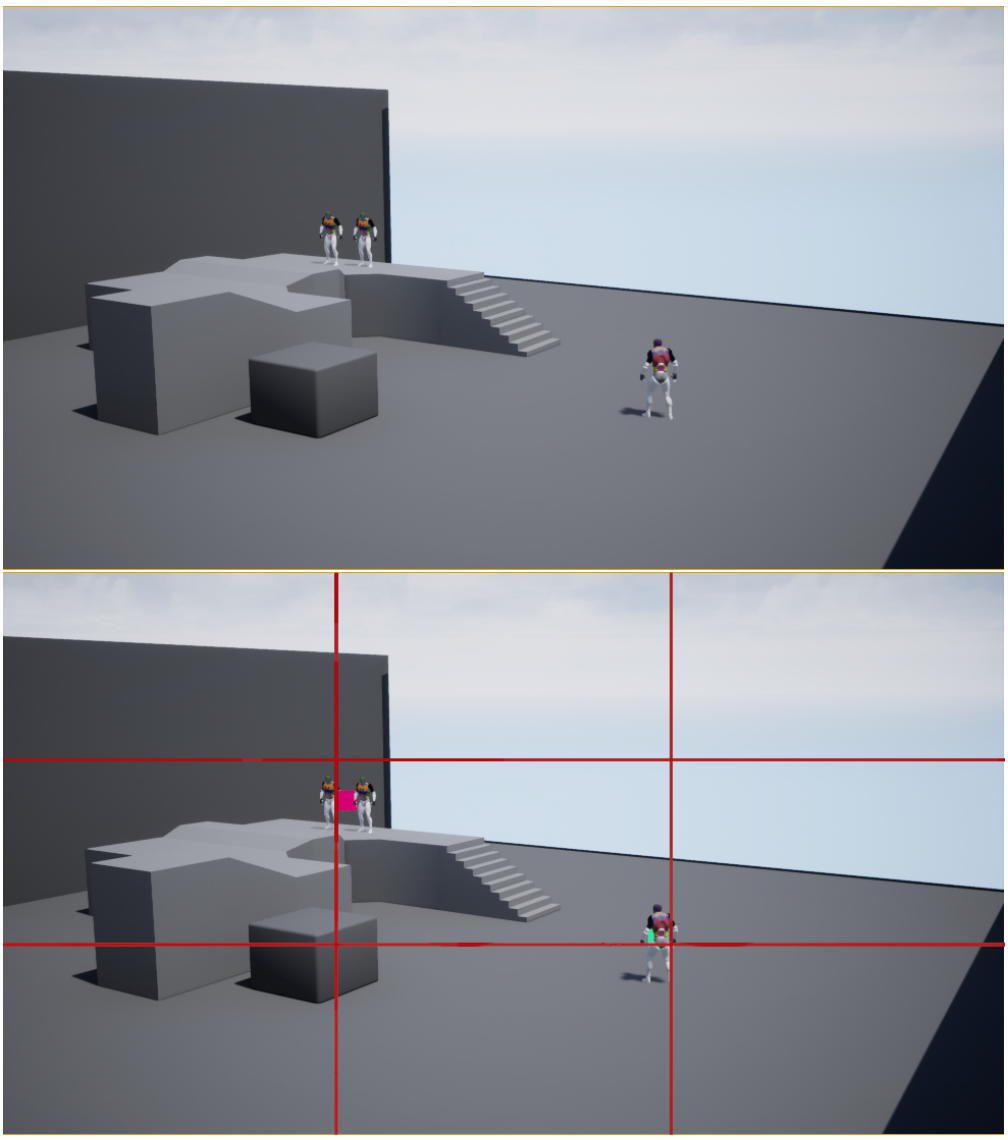


Figure 4.2: Skjermbilde av et UE4 template prosjekt sett gjennom dronen, med tredelingsrutenett og to karakterer

I figur 4.2 ser vi at dronen har plassert subjektet i nedre høyre kryss i tredelingsnettet. Vi ser at midtpunktet, som ligger mellom de to andre karakterene, ikke er plassert på den horisontale linjen da dronen prioriterer å holde synsvinkel og avstand til subjektet.

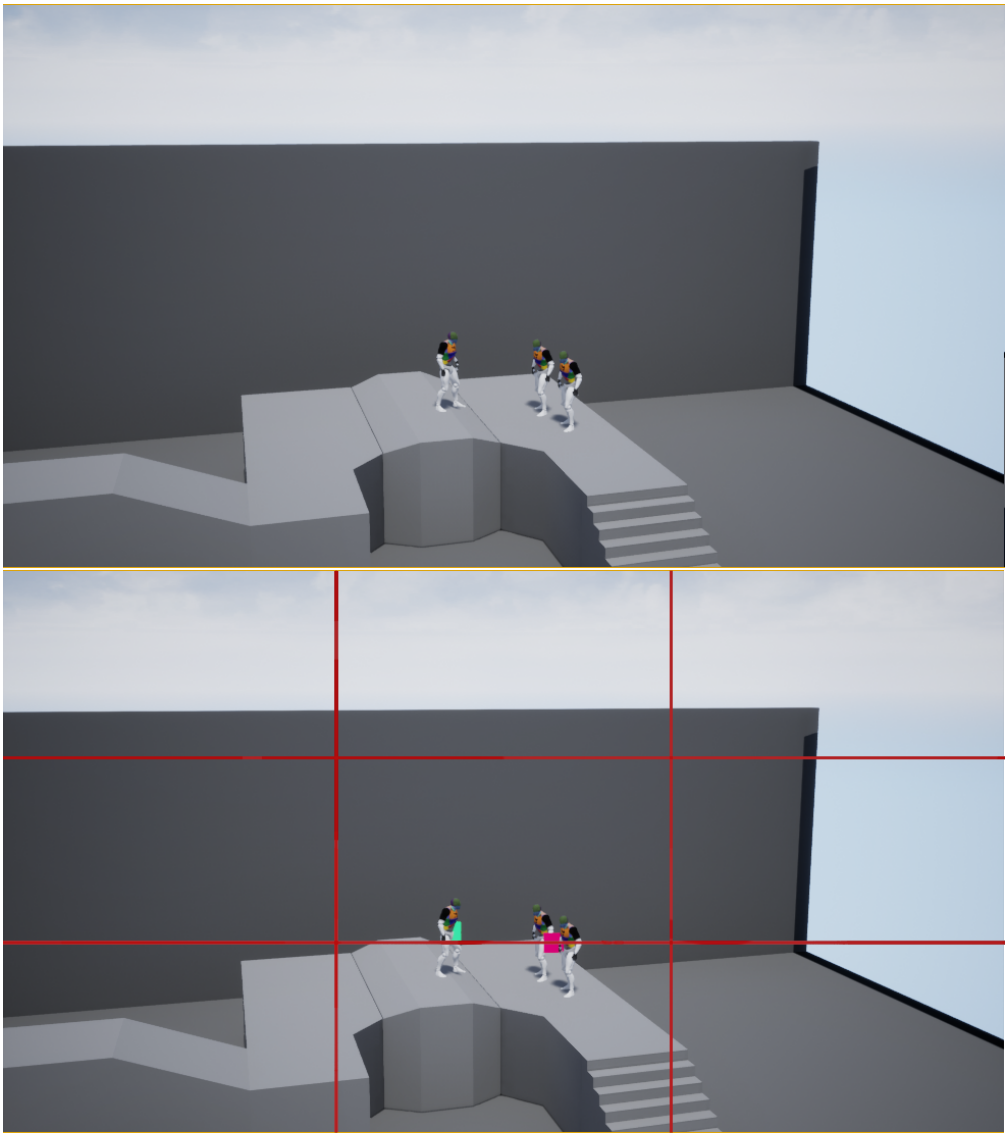


Figure 4.3: Interessepunktene står nærmere hverandre

Når subjektet og de andre karakterene er for nærme til at dronen kan sette begge på de vertikale linjene, ser vi at dronen lager et kompromiss i figur 4.3.

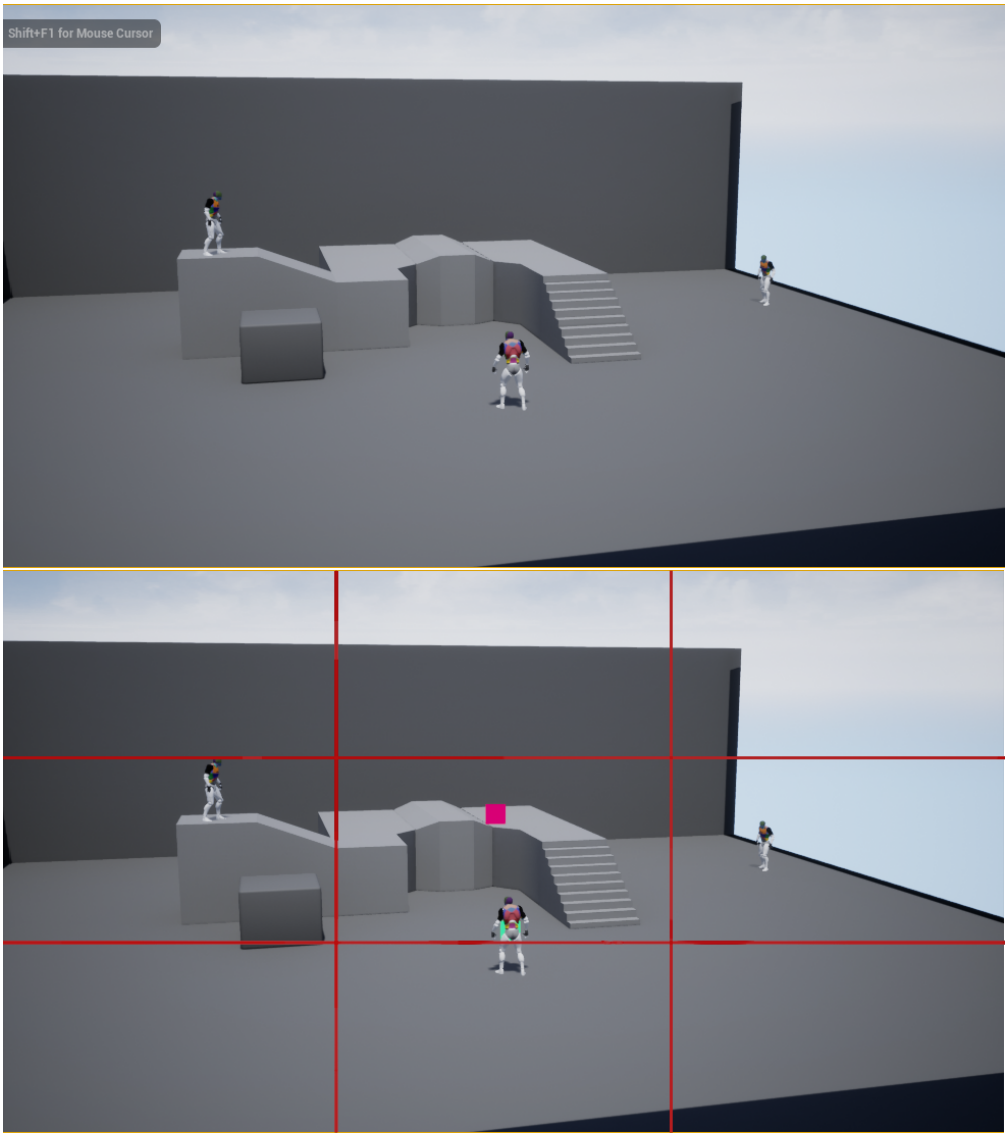


Figure 4.4: Interessepunktene er spredt ut over skjermen

Når det står interessepunkter på hver side av subjektet slik som i figur 4.4, vil dronen plassere subjektet mellom de vertikale linjene i tredelingsnettet. Den bryter altså komposisjonsregelen til fordel for å beholde så mye informasjon i synsfeltet som mulig.

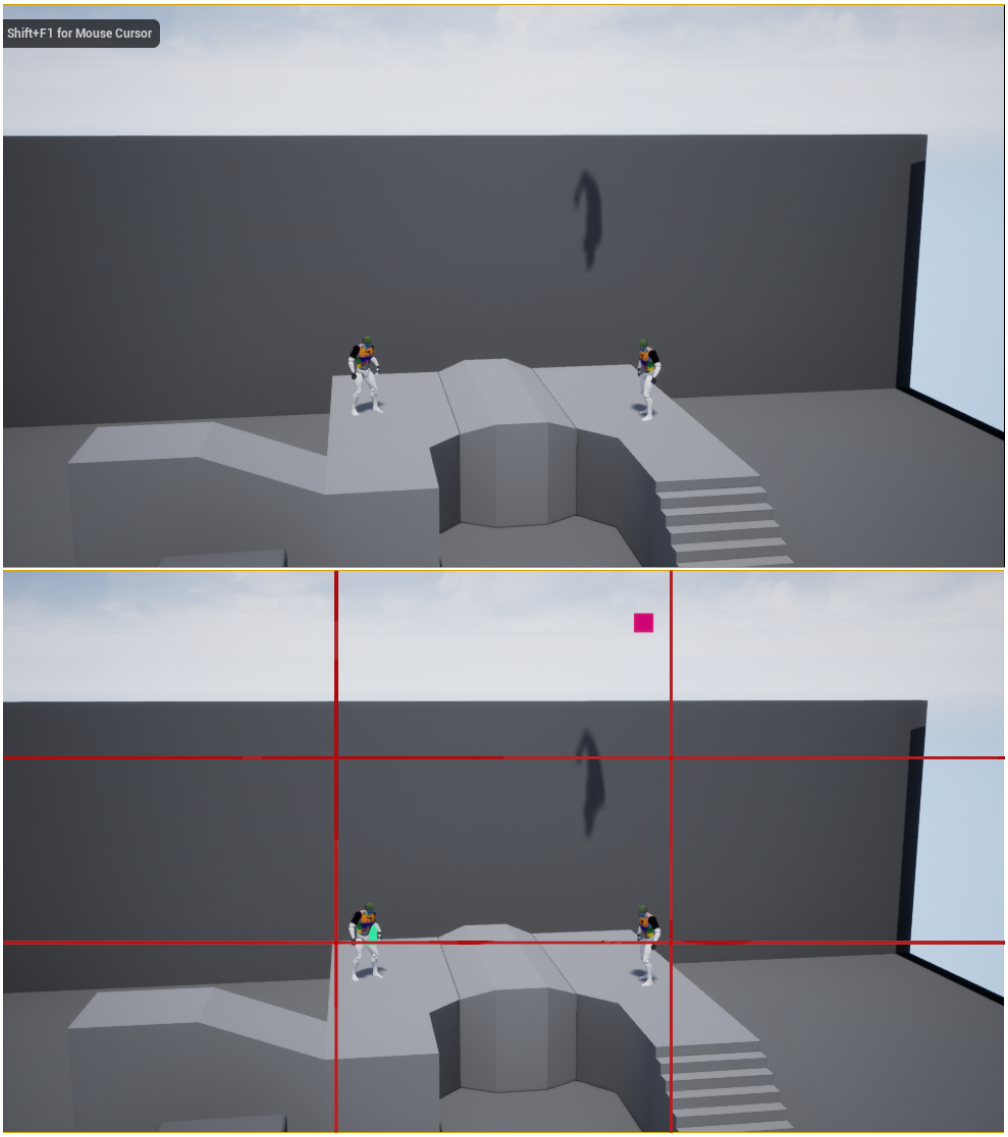


Figure 4.5: Komposisjon når 1 figur har forlatt skjermen (skyggen er synlig på veggen)

I figur 4.5 har en av karakterene forlatt skjermen via toppen. Dronen har ikke funksjonalitet til å justere den vertikale rotasjonen for å få med alle interessepunktene i synsfeltet.

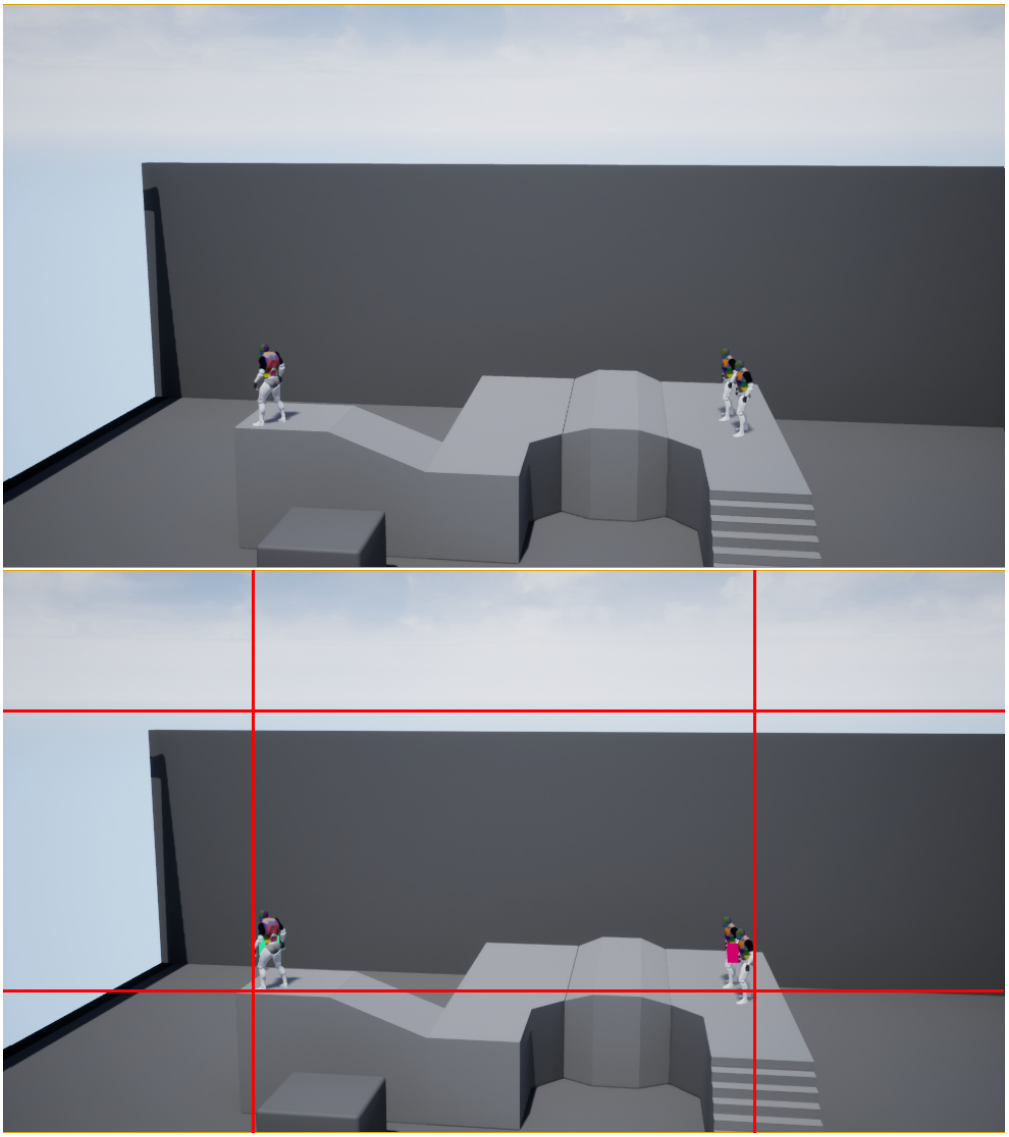


Figure 4.6: Komposisjon når komposisjonsregelen er satt til firedelingsregelen

Dronen kan endres til å benytte seg av andre forhold i komposisjonen. I figur 4.6 er "firedelingsregelen" brukt, hvor dronen prøver å holde subjekt og midtpunkt mellom de to ytterste fjerdelene av rutenettet. Vi ser her at subjektet ligger høyere over den horisontale linjen enn det gjorde når tredelingsregelen ble brukt.

Dersom komposisjonsregelen er for bred, dronens rotasjonshastighet er for lav, og subjektet kan få veldig stor fart er det mulig for subjektet å løpe ut av kameraets synsfelt. Dette kan løses ved å bruke en smalere komposisjonsregel eller øke rotasjonshastigheten til dronen.

4.1.4 Plugin

Et av forskningspørsmålene var definert som "Hvordan kan dronesystemet designes så det kan integreres i ulike prosjekter?". Dette ble løst ved å lage systemet som en UE4-plugin. Dette avhenger av at all kode i pluginen ikke har noen eksterne avhengigheter, og dermed kan den anvendes i andre prosjekter. Den største utfordringen her var hvordan definere actors i et prosjekt som interessepunkter dronen skal filme. For å unngå avhengighet her løste vi dette ved å utvikle et system med POI Actor Components. Actor Components kan festes til hvilke som helst Actors, og når POI-komponentene er definerte i pluginen er alle avhengighetene interne.

En annen viktig utfordring er hvordan tilrettelegge for integrasjon i prosjekter. Pluginen er ment for å videreutvikles og tilpasses det enkelte prosjekt, og da er det spesielt viktig med omfattende dokumentasjon og veiledning til hvordan eventuelle anvendere håndterer dette.

Derfor er det her utarbeidet dokumentasjon av kildekode som skal være ledende for utviklere når de skal videreutvikle. I tillegg til beskrivelser av variabler, funksjoner og Events viser denne dokumentasjonen tydelig til hvilke områder av koden som er spesielt ment for å videreutvikle på egenhånd, ved hjelp av fargede kommentarbokser som viser områder som bør tilpasses.

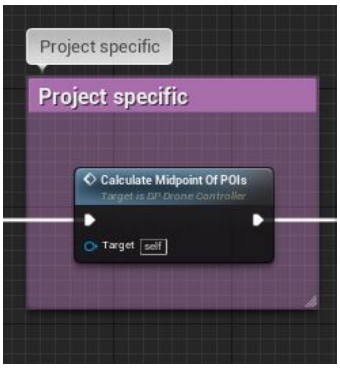


Figure 4.7: Slik er noder som bør tilpasses markert.

Det er også laget en ReadMe-fil i Github-prosjektet (Gram et al., 2020a) som forklarer hvordan en installerer og implementerer pluginen i sitt prosjekt, samt en guide for hva en bør videreutvikle (samme som de markerte områdene i kildekoden), og hvilke betraktninger en må ta for videreutviklingen i hensyn til sitt prosjekt.

4.2 Ingeniørfaglige resultater

De ingeniørfaglige resultatene er de som er knyttet mer spesifikt til utviklingen av produktet og tar for seg måloppnåelsen av de kravene som er satt av oppdragsgiver og

definert i visjon- og kravdokumentene (ref. Vedlegg: Visjonsdokument; Vedlegg: Kravdokument).

4.2.1 Funksjonelle krav

I denne seksjonen er de funksjonelle kravene fra visjons- og kravdokumentene listet opp. De er sortert etter om de er ferdigstilte, uferdige men påbegynt, og funksjonelle krav som ikke er utarbeidet. De påbegynte kravene har også status som forteller om hvor i prosessen de ligger.

Ferdigstilte Funksjonelle Krav

- Systemet skal være en Unreal Engine plugin
- Droner skal være selvstyrte
- Droner må kunne beveges
- Droner skal bevege seg jevnt
- Det skal være mulig å definere interessepunkter for dronene
- Droner må ha en kamerakomponent som man kan se ut av
- Flere droner må kunne oppholde seg i samme miljø
- Droner må kunne følge relevante interessepunkter med kamera, og forsøke å putte dem i en komposisjon
- Droner må kunne følge et interessepunkts posisjon

Funksjonelle krav som er påbegynt, men ikke ferdigstilt/har klart rom for forbedring

- Et sentralisert styringssystem delegerer oppgaver til droner
- Grensesnittet skal vekte de ulike interessepunkter for dronesystemet
  Status: Grensesnittet gir dronene interessepunkter å ta hensyn til basert på avstand. Mer sofistikert vekting gjøres av dronen.
- Droner må ha variabel synsfelt
  Status: Synsfeltet er variabelt i den forstand at det kan endres på, men ingen av funksjonene som er implementert justerer eller trenger å endre synsfelt.
- Droner må unngå kollisjon
  Status: Kollisjonsunngåelse er implementert, men ikke veldig komplekst.
- Droner skal ikke forstyrre for spillet
  Status: Dronene vil naturlig ikke forstyrre spillet da de prøver å holde seg over spillområdet. Det er ikke implementert logikk for å definere spillområdet og be dronene holde seg ute av dette i mer vertikale baner.

- Systemet skal kunne bytte mellom hvilken drone seere ser ut gjennom.
  Status: Bytte av kamera kan ikke pakkes med en plugin, fordi hvordan dronene anvendes må bestemmes for hvert enkelt prosjekt. Det er lagt rede for et eksempel for å gjøre dette på en helt enkel måte i prosjektets ReadMe på Github (Gram et al., 2020a).

- Droner skal kunne vite hvilke andre interessepunkter som er relevante å følge sammen med sitt hovedinteressepunkt
  Status: Dronene tar hensyn til de interessepunktene som er vinklet mot subjektet sitt, eller som subjektet er vinklet mot. Interessepunkter som har interaksjoner vil oftere være vinklet mot hverandre enn bort fra hverandre.

- Systemet skal kunne regne ut hvilken drone det er mest interessant å se ut av
  Status: Dronene regner ut hvor mange interessepunkter som er i sikt og hvor nærme midtpunktet de er. Vi kan ikke påstå at vi kan regne ut den mest interessante dronen å se ut av. Vi tør heller å påstå at vi kan finne en drone som ikke er den minst interessante å se ut av.

- Droner må kunne unngå blokkering av sikt mot subjekt
  Status: Dronen opprettholder sikt mot subjektet ved å øke vinkelen som vist i 3.14. Sikten blir opprettholdt mesteparten av spilltiden. Det vil oppstå flere problemer på baner som er mer vertikale enn Setback banen som ble brukt under utvikling, eller har seksjoner med mye tak og vegger.

Funksjonelle krav som ikke er utarbeidet

- Droner må kunne følge en bane

- Synsfelt justeres etter dronehøyde i forhold til subjekt

- Droner må ignorere små endringer i z aksen der hvor subjektet ikke er i kontakt med bakken

- Droner skal kunne vurdere om det er hensiktsmessig å tilpasse seg mindre visuelle sperringer

- Systemet må estimere fremtidig posisjon til interessepunkter basert på farts- og akselerasjonsvektor

4.3 Administrative resultater

Før eksamen 10.mars hadde vi ukentlig plass på oppdragsgivers kontor hvor vi kunne arbeide, der vi fikk kommunisert med utviklerne av Setback. I resten av semesteret

hadde vi kontakt med oppdragsgiver over Discord, hvor vi hadde møter etter behov.

4.3.1 Scrum

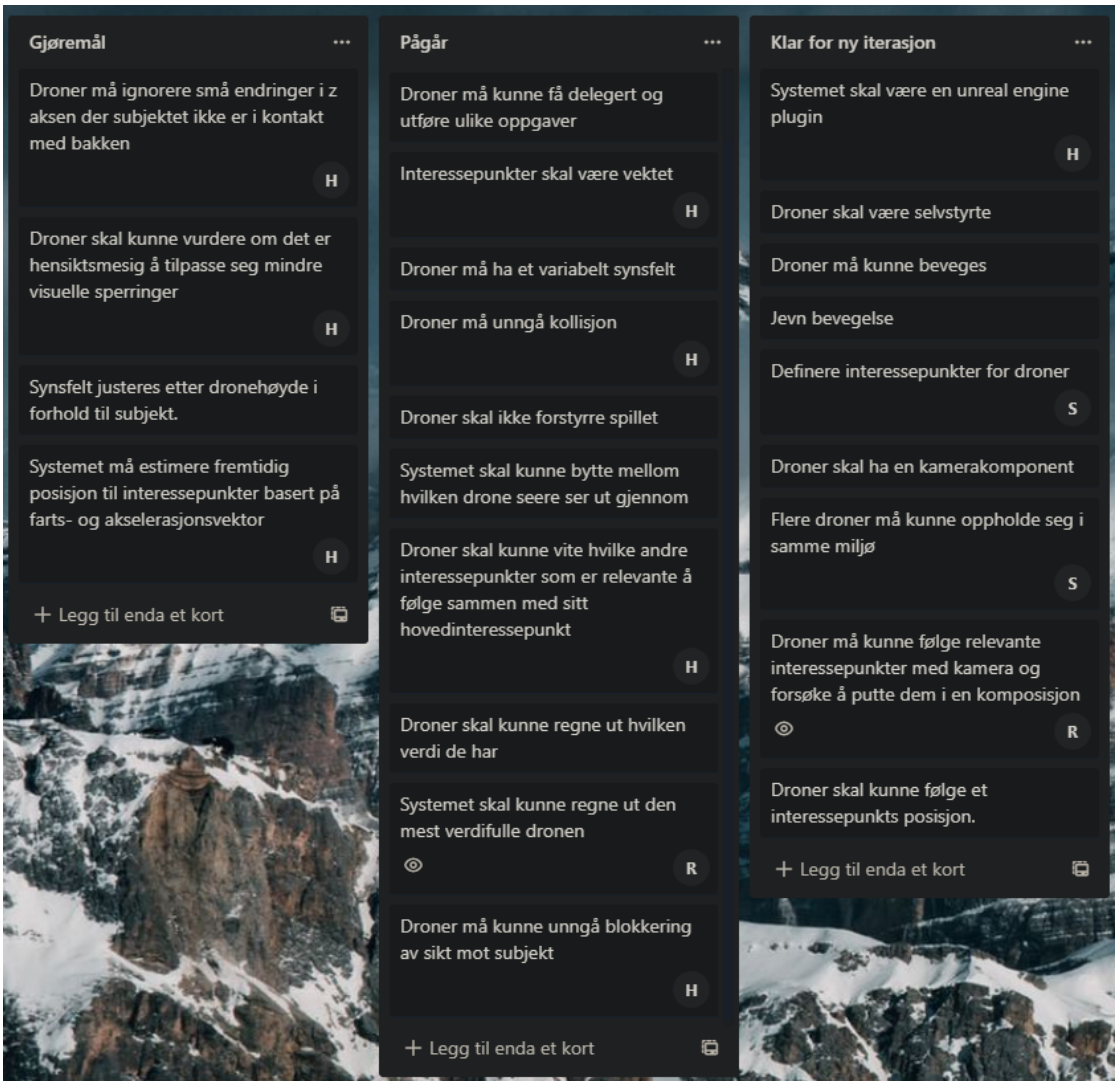


Figure 4.8: Skjermbilde av gruppens Trello

Gruppen har benyttet seg av et Scrum-board (ref. figur 4.8) under gjennomføringen av prosjektet. Det ble moderat brukt og var ikke alltid oppdatert.

Gruppen har ikke planlagt noen sprinter, gjennomført sprint-reviews eller retrospektiver. Det er ikke laget et burndown-chart eller sprint-backlog. User stories ble laget som en del av kravdokumentasjonen (ref. Vedlegg: Kravdokument), disse er ikke benyttet videre i utviklingen.

Etter at campus stengte ble det mer sytematisk med stand-up møter hver morgen over Discord. På disse møtene ble det diskutert hva gruppemedlemmene hadde gjort, hva de skal gjøre videre, og hvilke blokkere det er i veien. Det ble også tatt hånd om administrative oppgaver som å planlegge møter med veilder og oppdragsgiver.

Dronen ble ikke utviklet iterativt i den kapasitet som forventes av et Scrum prosjekt. Det var noe iterasjon når dronen ble omarbeidet for å gjøre ting mer ryddig i UE4 blueprint filene, men disse var ikke en del av noen sprint-planlegging.

4.3.2 Prosjekthåndboken

I starten av prosjektet utarbeidet gruppen et Gant-diagram og anslo hvor mange timer som skulle brukes på hver av de satte aktivitetene. Gruppen undervurderte hvor mye tid det ville ta å utvikle dronen, og overvurderte hvor mye tid som ville gå med til å skrive dokumentasjon for systemet. Det ble også satt av mer tid til å skrive rapport en det som har blitt ført.

Gruppen hadde veiledningsmøter ca. hver tredje uke (ref. Møteinnkallinger i Vedlegg: Prosjekthåndboka). Vi fikk innspill fra veileder om ting vi lurte på angående prosessen og hvor vi var i prosjekt utførelsen.

Aktivitetene

I følge Gant diagrammet skulle planleggingsdelen av prosjektet være ferdige til fristen veileder satt for visjon- og kravdokumenter (28.02.2020). Det har blitt ført timer under planleggingsaktiviteten i uke 21 (18.05.2020-24.05.2020), som er siste uken av prosjektet.

Research delen av oppgaven skulle vare frem til 24.04.2020. Med unntak av 2 timer i uke 19 ble siste de timene på denne aktiviteten ført i uke 17 (20.04-26.04.2020). 200 timer ble anslått, 221 timer ble ført.

Lære Unreal aktiviteten ble satt til å avsluttes med en workshop hos Riddlebit Software som fant sted 21.02.2020. Dette er også den siste datoen det ble ført timer på denne aktiviteten. Gruppen anslo 100 timer, 89 ble ført.

Utvikle drone aktiviteten skulle avsluttes 30.04.2020. De siste timene som ble ført på denne aktiviteten var i uke 19 (04.05.2020 - 10.05.2020). Anslaget var 500 timer, det ble ført 631 timer.

Dokumentasjon aktiviteten skulle vare fra 01.03.2020 til 25.04.2020. Den første timen som ble ført på denne aktiviteten var to måneder senere i Uke 18 (27.04.2020 - 03.05.2020), og den siste timen ble ført i uke 20 (11.05.2020 - 17.05.2020). Det ble anslått 100 timer til dokumentasjon, 39 ble ført.

Det var planlagt at hovedrapporten skulle begynne å bli skrevet 01.04.2020 og avsluttes 20.05.2020. Dette ble ikke påbegynt før to uker senere i uke 16 (13.04.2020-19.04.2020) og det ble utsettelse på fristen på rapporten til 24.05.2020. Det ble anslått 500 timer til denne aktiviteten, 397 har blitt ført.

5 Diskusjon

I dette kapittelet blir resultatene fra kapittel 4 diskutert og vurderes i forhold til hvorvidt disse svarer på forskningsspørsmålene fra kapittel 1.2, og eventuelle områder vi ser behov for forbedring. Arbeidsprosessen vil også diskuteres.

5.1 Forskningsspørsmål

I denne seksjonen tar vi for oss hvert forskningsspørsmål og diskuterer produktet i relasjon til spørsmålet. Det evalueres hvordan vi forsøkte å løse utfordringen, hvorvidt løsningen er god, og eventuelle mangler og utfordringer ved løsningen.

5.1.1 Interessepunkter

Første forskningsspørsmål var definert som "Hvordan kan et system med kameradroner forholde seg til flere dynamiske interessepunkter?".

Måten dronen forholder seg til interessepunkter er i stor grad ved å redusere dem ned til et midtpunkt som den så bruker. Dette gjør at matten for å forholde seg til interessepunktene blir veldig enkel, siden dronen kun må forholde seg til subjektet og midtpunktet. Det er en risiko ved denne løsningen hvor dronen kanskje burde fokusere på noen få spesifikke interessepunkter istedenfor å strekke seg etter alle den har blitt tildelt. I et forsøk på å løse dette problemet er det tilrettelagt for å differensiere mellom ulike typer interessepunkter ved å lage ulike barnekomponenter, slik at disse kan behandles ulikt etter behov. Likevel er det ikke alltid dette vil hjelpe, fordi dronens destinasjonsområde alltid vil være helt avhengig av dets subjekt, som betyr at det er begrenset hvordan dronen kan tilpasse seg andre punkter enn subjektet.

Vi har altså funnet en løsning for hvordan en drone kan forholde seg til flere interessepunkter, men det er ganske klart at det finnes andre løsninger. Vi har sett på en løsning som bruker Boids-algoritmen for å få en spredt dekning over kartet. Med dette er dronenes destinasjon ikke avhengig av noen enkelte interessepunkter, og står mer fritt til å bestemme hvilke punkter det fokuserer på. Dette ville medført en annen måte å forholde seg til interessepunktene på, som vi ikke har fått tid til å utvikle.

5.1.2 Oversikt

Andre forskningsspørsmål var "Hvordan kan systemet skape en helhetlig oversikt over interessepunktene?".

Dronesystemet kan forsøke å oppnå oversikt ved å kontrollere hvor mange droner som jobber, og hvilke subjekter de jobber med. Hvordan dette kontrolleres er mer opp til

den enkelte utvikleren. Hvert prosjekt kan ha helt forskjellige behov i dette området, så derfor er det ikke hensiktsmessig å lage en spesielt utarbeidet måte for å holde oversikt over interessepunkter. Derfor er det heller lagt til rette metoder for å opprette og fjerne droner etter behov, samt tildele droner interessepunkter som kan anvendes etter behov.

Her er avhengigheten til ett enkelt subjekt problematisk igjen. Dersom det ønskes en helhetlig oversikt, er ikke dette nødvendigvis en god løsning, når hver drone vil fokusere på ett enkelt subjekt framfor å prøve å få god dekning over større områder.

Derfor har vi laget en simulasjon for en annen type dekning som baserer seg på "Reynolds flocking" (ref. 3.4.1). Dette er en mer dynamisk måte å posisjonere dronene på, som ikke er avhengig av enkelte subjekter. Hvordan dronene skal oppføre seg blir styrt utifra den modifiserte flokk-algoritmen. Her er det mange variabler som kan være med på å finjustere oppførselen til dronen innenfor destinasjon, bevegelse og fokus. Denne algoritmen kan bli skalert opp til et stort antall droner da hver drone bare trenger å forholde seg til droner og interessepunkter innen for oppfatningsradiusen sin. I tillegg har den en separerings faktor som sprer dronene utover et større området, større en oppfattningsradiusen, men en optimalisering som kan gjøres her er å fordele utregningene over flere prosessorkjerner for bedre ytelsen.

Vi fikk implementert og brukt deler av denne algoritmen der det passet i samkjøring med de andre metodene nevnt i kapittel 3. Men vi valgt å ikke implementere dette programmet inn i systemet da det å finne de optimale variablene viste seg å være for tidkrevende. Her hadde det vært mulig å bruke maskinlæring til å finne de optimale verdiene, men dette kan også være vanskelig da det ikke er noen kontrete begrensninger. Dette kunne blitt fokus på et egent forskningsprosjekt.

Det er en vanlig feil i spillutvikling når det kommer til kamera at utviklere prøver å gjøre for mye av kamera-arbeidet automatisk (Nesky, 2016). I mange tilfeller vil det være hensiktsmessig å være mer direkte når det kommer til å dirigere kameraet. Man kan gjøre en analyse av spillet og banen og manuelt bestemme hvor dronen burde være på kartet i spesifikke situasjoner. I vårt prosjekt ønsket vi å utvikle en plugin, som betyr at dronen skal være bane- og spilluavhengig. Spillet Setback som dronen er designet for var også under utvikling i prosjektperioden, dermed ville det vært urealistisk å forvente at banen og spillet skulle forbli den samme etter prosjektperioden. Banen som gruppen har brukt til testing ble fjernet fra Setback 13/05/2020, som vil si at dronesystemet hadde vært deprekert før prosjektperioden var ferdig om det hadde vært avhengig av den spesifikke banen.

En tydelig begrensning av systemet vårt er hva slags miljøer det fungerer dårlig i. Systemet er utviklet med Setback som testemiljø, på en bane som har lite høydeforskjell, og for det meste helt åpent rom over banen. Dronen er dermed utviklet med dette som basis. Derfor er dronen dårlig til å tilpasse kamera til større vertikale avstander mellom punkter. Måten dronen håndterer siktsblokkeringer gjør at den vil flytte seg oppover for å finne en bedre kameravinkel. Dette vil være problematisk i miljøer hvor det er mange objekter i området over spillfeltet. Dette er likevel tatt med i betraktning under utvikling. Det skal være mulig å nokså lett kontrollere dronens posisjon bedre, siden avstand og vinkel til subjekt kan styres ved å endre på noen enkelte variabler (da først og fremst POISphereRadius og PreferredVerticalAngle). Det kan fortsatt by på

utfordringer, fordi destinasjonsområdet på breddegraden (ref. figur 3.16) kan havne i et uønsket felt, og dette området er viktig for å putte midtpunkt i komposisjon.

5.1.3 Visuell komposisjon

"Hvordan kan en kameradrone tilpasse seg et sett med interessepunkter for å oppnå en visuell komposisjon?" var det tredje forskningsspørsmålet.

Som vist i kapittel 4.1.3 har dronene funksjonalitet for å putte punkter i et komposisjonsrutenett, slik som tredelingsregelen. Med måten vi endte opp med trenger komposisjonen kun forholde seg til to punkter, som er relativt lett for dronen å forholde seg til. Det var snakk om å utvikle funksjonalitet for å finne en optimal plassering for dronen å oppholde seg for å på best mulig måte fange opp interessepunktene i komposisjon. Dette byr på en rekke utfordringer, hovedsakelig hvordan man bestemmer hva som er "optimalt" for dronen og hvordan den kan forholde en rekke interessepunkter til en komposisjon. Derfor valgte vi å fokusere på dette mer i optimaliseringsproblemet.

Fokuset i pluginen ble heller på å holde ett enkelt subjekt i fokus, og forsøke å filme interessepunkter relevante for subjektet. Vi anså dette som også særdeles relevant, fordi i mange sammenhenger er det hensiktsmessig å holde spesiell fokus på ett element framfor en helhetlig oversikt over mange, for eksempel dersom venner vil følge med på hva en enkelt spiller gjør, eller om det i e-sportsammenheng er av interesse å fokusere på enkeltspillere. Vi anså det også som mer aktuelt for mindre prosjekter hvor fokus gjerne er på den enkelte spilleren. Av denne grunn ble det naturlig å sette fokus på enkelte spillere.

Om vi ser på komposisjonsbildene i figur 4.6, ser vi at punktene ikke overlapper helt med komposisjonslinjene, selv om i dette tilfellet skulle det egentlig gjort det. Vi er ikke helt sikre på hvorfor dette forekommer, men vi har en mistanke om at det kan komme av en "Fish Eye"-effekt på kameraet, slik som i figur 5.1. I dette tilfellet vil en vinkelgrad nærmere sentrum av bildet være videre enn en vinkelgrad lenger unna. Likevel anså vi ikke dette som en prioritet, siden vi anså løsningen vi kom fram til som god nok i forhold til tiden som var til disposisjon.

5.1.4 Plugin

Det siste forskningsspørsmålet var formulert som "Hvordan kan dronesystemet designes så det kan integreres i ulike prosjekter?".

At dronesystemet ble utviklet som en plugin har gjort at det ikke har noen eksterne avhengigheter. På grunn av dette kan systemet anvendes i hvilket som helst UE4-prosjekt. Det er testet i flere ulike template-prosjekter som følger med UE4 samt Setback, og implementasjonen fungerer likt i alle. Det er testet av alle i gruppa samt oppdragsgiver fra Riddlebit, og det skal fungere.

I tillegg til å være implementerbart skal også systemet kunne videreutvikles og tilpasses det enkelte prosjekt. Dette er svært viktig, fordi slik dronesystemet er ment for å anvendes er det umulig å lage noen "One size fits all"-løsning. Det kommer tross alt opp til

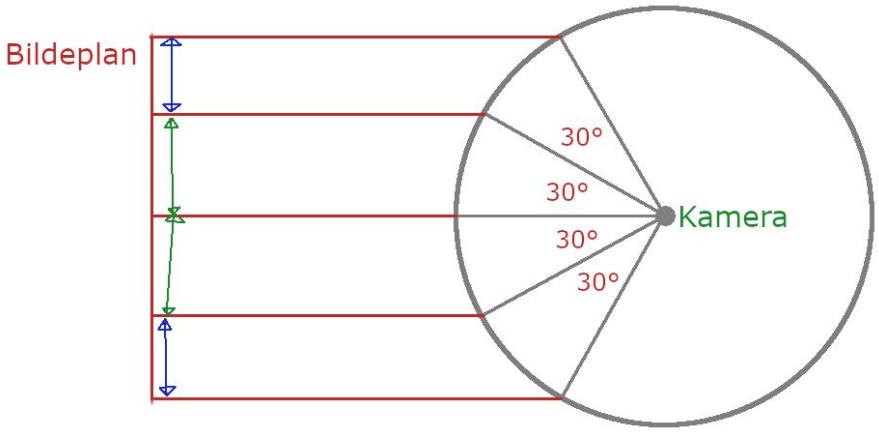


Figure 5.1: Fisheye-effekten forvrenger bilde etter hvor bred vinkel linsen får med seg. De blå feltene og de grønne feltene inneholder alle 30 grader, men de blå blir smalere når de projiseres på bildeplanet.

utvikler og deres prosjekt hva som ønskes at skal filmes og hvordan ulike interessepunkter behandles og prioriteres.

Derfor er systemet utviklet og dokumentert med tanke på dette. All funksjonalitet er oppdelt, organisert og dokumentert. Funksjoner som er viktigst for anvender å tilpasse er tydelig markert, og hvilke betraktninger en bør ta i henhold til disse er lagt til rette for i systemets ReadMe på Github (Gram et al., 2020a), og systemdokumentasjonen (se vedlegg) skal være til hjelp for å forstå systemet nærmere.

Vi har likevel ikke hatt noen brukertester for selve tilpasningsdelen av plugin. Testing av dette ville vært en omfattende prosess, og siden det ferdige testbare produktet ble sent ferdig i en periode med stor arbeidsmengde, har ikke dette blitt prioritert. Pluginen skal derimot anvendes i Setback og videreutvikles av Riddlebit, og når de begynner med dette vil vi få informasjon om hvor lett systemet er å jobbe med.

5.1.5 Profesjonsetiske problemstillinger

For vårt produkt, i forhold til det tiltenkte bruksområdet, er det vanskelig å se for seg noen sosioøkonomiske etiske problemstillinger. Selve produktet er gratis å anvende, og har ingen politiske eller sosiale aspekter.

En problemstilling som kan være relevant er fysisk helse hos seere. Dersom produktet skulle vise seg å være av lav kvalitet, er det ikke utenkelig at det kan volde skade hos seere i form av svimmelhet eller trigge epileptiske anfall. Håpet er at produktet vårt er godt nok til å unngå dette, men det gjenstår å se.

5.2 Ingeniørfaglige resultater

Det er tydelig at gruppen undervurderte mengden arbeid hvert funksjonelle krav ville kreve. I et slikt dronesystem kan man lage de individuelle delene ekstremt kompliserte. Noen av de funksjonelle kravene kunne vært egne oppgaver i seg selv. Vurdere hva som er relevante interessepunkter å se på, vurdere hvilken drone av flere som det er *mest* interessant å se ut av, og blokkering av sikt/kollisjonsunngåelse/veifinning i luften, er alle funksjonelle krav som man kan gå ekstremt i dybden på.

Det var noen krav som ble nedprioritert når vi fikk bedre oversikt over hva vi kom til å ha tid til å oppnå. Da vi bestemte oss for å ha dronen en bestemt høyde i forhold til subjektet ble det å variere synsfelt basert på høyde i forhold til subjektet mindre viktig. Vi bestemte oss for å ikke fjerne kravet, siden det fremdeles kan være nytte for funksjonen dersom en utvikler som bruker dronen som plugin endrer den til å skulle kunne filme

5.2.1 Funksjonelle krav som ikke er utarbeidet

Disse funksjonelle kravene endte opp med å ikke bli inkludert i produktet av ulike årsaker, som gjøres rede for her.

- Droner må kunne følge en bane
  Dette ble nedprioritert, fordi vi anså dette til å ikke være viktig for vårt produkt. Kan være mer relevant i framtiden.

- Synsfelt justeres etter dronehøyde i forhold til subjekt
  Dette ble også nedprioritert. Det er likevel lagt til rette for å implementere, siden justering av synsfelt ikke vil påvirke den visuelle komposisjonen.

- Droner må ignorere små endringer i z aksen der hvor subjektet ikke er i kontakt med bakken
  Dette ble forsøkt, men ble til slutt nedprioritert fordi det vi kom fram til virket unaturlig i vårt system, og dette ikke var spesielt viktig for vårt produkt. Verdt å se videre på.

- Droner skal kunne vurdere om det er hensiktsmessig å tilpasse seg mindre visuelle sperringer
  Siden det ikke var essensielt å utarbeide for sofistikerte metoder for dette for å finne svar til forskningsspørsmålene ble dette nedprioritert.

- Systemet må estimere fremtidig posisjon til interessepunkter basert på farts- og akselerasjonsvektor
  Siden dette var ment å brukes for forrige punkt ble dette også nedprioritert.

5.3 Refleksjon om gruppens arbeid

Som nevnt i kapittel 3.2.2 bestemte vi oss for å vente med å jobbe i henhold til Scrum-prinsipper til etter eksamen i Systemtenkning og Økonomi 18.03.2020. Dette var fordi vi visste at vi ikke kom til å ha en veldig forutsigbar timeplan før dette, med flere forelesninger og obligatorisk opplegg. I ettertid ser vi at det ikke ville vært så vanskelig å sette opp ordentlige sprinter. Vi var nok ikke fullt så erfarne med Scrum som vi trodde, da det hele falt sammen når vi selv hadde ansvaret for at prosessen ble gjennomført. Da smittevernsreglene som sa at studenter ikke skulle oppholde seg på campus ble innført 12.03.2020 måtte gruppen begynne å jobbe hjemmefra. Da ville det vært spesielt gunstig å være mer strukturerte, ha retrospektiver og passe på at vi jobbet effektivt. Når det er sagt føler vi at den vannfallsmetoden vi endte opp med har fungert tilstrekkelig. Selve systemet er nokså lite og oversiktlig, og det ble jobbet mot endelige mål, så vannfallsmetoden var ikke noe spesielt til hindring. Å jobbe iterativt slik som Scrum sier ville hatt mye overhead i et UE4 prosjekt, da to personer ikke kan jobbe på den samme tingen siden Git Version Control ikke fungerer med .uasset-filer. Gruppen klarte stort sett å holde seg til den planen som var satt i starten av prosjektet, og vi er nokså fornøyde med det.

6 Konklusjon

Forskningsspørsmålene som ble oppgitt i kapittel 1 var:

1. Hvordan kan et system med kameradroner forholde seg til flere dynamiske interessepunkter?
2. Hvordan kan systemet skape en helhetlig oversikt over interessepunktene?
3. Hvordan kan en kameradrone tilpasse seg et sett med interessepunkter for å oppnå en visuell komposisjon?
4. Hvordan kan dronesystemet designes så det kan integreres i ulike prosjekter?

Det å lage et dronesystem som skal ha god oversikt og komposisjon, og fungere i flere typer prosjekter, har vist seg å være mer komplisert enn det gruppen først antok. Dersom dette prosjektet skulle blitt utarbeidet til sitt fulleste potensiale hadde dette vært et prosjekt som krever mer enn fem måneder og 3 studenter. Derfor har fokuset endt særlig opp på visse deler av prosjektet, slik som komposisjon og anvendbarhet, mens andre deler ble nedprioritert, slik som kollisjonshåndtering og siktopprettholding. Vi anser likevel produktet vi endte opp med som en god tilnærming til å løse forskningsspørsmålene, og det skal være godt lagt til rette for å videreutvikling. Vi ser derfor mulighetene for et mer utarbeidet og komplekst system basert på vårt system.

Selve systemet for interessepunkter mener vi er et godt grunnlag for videre arbeid. Siden dette tillater å behandle interessepunkter ulikt basert på hva slags type det er, ser vi på det som mulig å anvendes i et hvilket som helst prosjekt.

Hvordan vi har tilnærmet oss å holde en helhetlig oversikt er ikke spesielt utarbeidet i selve UE4-pluginen. Systemet slik det er nå fokuserer ikke på å få fullstendig oversikt, men fokuserer heller på noen bestemte subjekter. For å forsøke på at systemet har full dekning må det være like mange droner på banen som det er spillere. Dette garanterer fortsatt ikke fullstendig dekning, og selv om systemet regner ut hvilken drone som har best oversikt, betyr ikke det at systemet kan garantere at en enkelt drone har spesielt god oversikt over kartet. Følgende mener vi at resultatene fra optimaliseringsproblemet kan være en mulig alternativ løsning som er bedre på dette feltet.

Når det gjelder komposisjon har løsningen vår visse svakheter, særlig når interessepunkter som forsøkes å filme er betydelig spredt. Vi mener likevel at dette er et godt alternativ for å oppnå komposisjon som virker naturlig.

At systemet er utviklet som en plugin har ført til at systemet er lett å implementere i alle typer prosjekter, men det er ikke nødvendigvis alle prosjekter som det vil fungere i. Selv om dette ikke var målet, ser vi fortsatt at det er visse svakheter som fører til at systemet fungerer dårlig i prosjekter som kunne ha hatt nytte av det, da særlig miljøer

av vertikal natur og miljøer med mye obstruksjoner over spillområdet. Likevel ser vi for oss at det er lagt opp for å videreutvikle systemet for å løse disse problemene samt å legge tilfunksjonalitet.

Vi konkluderer med at det er mulig å lage et system som dekker våre forskingspørsmål, selv om vi ikke har fullstendige svar på alle forskningsspørsmålene og ser områder hvor det burde viderearbeides. Vi ser altså på produktet som en god prototype for et dronesystem som kan videreutvikles.

6.1 Videre arbeid

I videre arbeid på dette prosjektet er det som nevnt tidligere tydelig flere felter som burde videreutvikles. Først og fremst anser vi det å implementere prinsippene fra optimaliseringsproblemet som mest interessant. Dette ser vi for oss som et interessant alternativ for å oppnå så god dekning som mulig, noe vårt system ikke tar spesielt hensikt til.

Kollisjonshåndtering og pathfinding var stort fokus i starten av prosjektet men falt ned på prioritetslisten når vi fikk bedre forståelse for problemet eller kom fram til noe nyttig over lengre tid. Dette er likevel en viktig del av systemet, og pathfinding i luften basert på kollisjonsunngåelse og opprettholding av sikt er et felt som burde utforskes videre. Dette ville nok vært et prosjekt i seg selv.

Slik som systemet er nå vil sikt på baner med tak og/eller mye vertikalitet være en utfordring. Derfor anser vi et mer allsidig system for å opprettholde sikt vil være nødvendig for å lage en mer allsidig drone.

En enkelt drone har ikke veldig god oversikt over hvor bra bildet den filmer er. Det er forsøkt å lage en metode for å bestemme dette, men å vurdere hvor bra sikten egentlig er er jo også veldig subjektivt. Kan man lære dronen å gjøre en ordentlig vurdering på hvor godt bildet den filmer er? Og kan man bruke dette til å utvikle en drone som kan predikere hvordan den burde filme for å konstruere gode komposisjoner som også gir god oversikt over spillbanen?

Vi mener at det burde utforskes flere alternative metoder for å oppnå komposisjon i bildet. Vi anser metoden vi har kommet fram til som en god tilnærming, men vi anser også variasjon som viktig, og derfor kan det bli ensformig å kun bruke denne metoden.

Bibliography

Bellard, M. (2019). Environment design as spatial cinematography: Theory and practice, *Game Developers Conference 2019*.

Bjørseth, P., Dao, K. and Ellevold, L. (2019). *Bruk av maskinlæring til å predikere fremtidige interessepunkter i et dataspill*, Norges teknisk-naturvitenskapelige universitet, Gløshaugen, Trondheim.

Epic Games, I. (2018). Cinematic depth of field method, `https://docs.unrealengine.com/en-US/Engine/Rendering/PostProcessEffects/DepthOfField/CinematicDOFMethods/index.html`. Hentet: 2020-19-03.

Gram, H.-J., Valen, R. S. and Andresen, S. M. (2020a). Github for vår generiske plugin. **URL:** *https://github.com/Hans-Jeiger/SpectatorDrones*

Gram, H.-J., Valen, R. S. and Andresen, S. M. (2020b). Video av dronesimulasjonen. **URL:** *https://youtu.be/phMS8jekB4E*

Johan Wagemans, James H. Elder, M. K. S. E. P. M. A. P. M. S. and von der Heydt, R. (2012). A century of gestalt psychology in visual perception i. perceptual grouping and figure-ground organization, **138**(6): 1172–1217.

Katz, S. D. (1991). *film directing shot by shot: visualizing from concept to screen*, Michael Wiese Productions, Studio City, California.

Krages, B. P. (2005). *Photography: the art of composition*, Allworth Press, New York, NY.

Nesky, J. (2016). 50 game camera mistakes, *Game Developers Conference 2014*.

Reynolds, C. W. (1987). Flocks, herds, and schools: A distributed behavioral model, *SIGGRAPH '87 Conference Proceedings*, Association for Computing Machinery, Anaheim, California, pp. 5–7.

Ross, S., Melik-barkhudarov, N., Shankar, K. S., Wendel, A., Dey, D., Bagnell, J. A. and Hebert, M. (2012). Learning monocular reactive uav control in cluttered natural environments, *IEEE International Conference on Robotics and Automation*, IEEE, RiverCentre, Saint Paul, Minnesota, pp. 1765–1772.

Sadeghi, F. and Levine, S. (2017). Cad2rl: Real single-image flight without a single real image, *Conference: Robotics: Science and Systems 2017*, Vol. 13.

Saska, M., Báča, T., Thomas, J., Chudoba, J., Preucil, L., Krajník, T., Faigl, J., Loianno, G. and Kumar, V. (2016). *System for deployment of groups of unmanned micro aerial vehicles in GPS-denied environments using onboard visual relative localization*, Vol. 41, Autonomous Robots.

Saska, M., Chudoba, J., Precil, L., Thomas, J., Loianno, G., Tresnak, A., Vonasek, V. and Kumar, V. (2014). Autonomous deployment of swarms of micro-aerial vehicles in cooperative surveillance, *2014 International Conference on Unmanned Aircraft Systems, ICUAS 2014 - Conference Proceedings*, pp. 584–595.

Yannakakis, G. N. (2012). Game ai revisited, *Proceedings of the 9th Conference on Computing Frontiers*, IT University of Copenhagen, Rued Langgaards Vej 7, pp. 1–6.

Ordliste

open-source Åpen kilde kode, at hvem som helst kan studere, endre og distribuere koden. iii

scrum Agil arbeidsprosess. 13, 48, 49, 55

tick Hver oppdatering av spillverdenen i UE4. For eksempel i spill med 60 FPS har generelt 60 ticks i sekundet.. 16, 19, 25

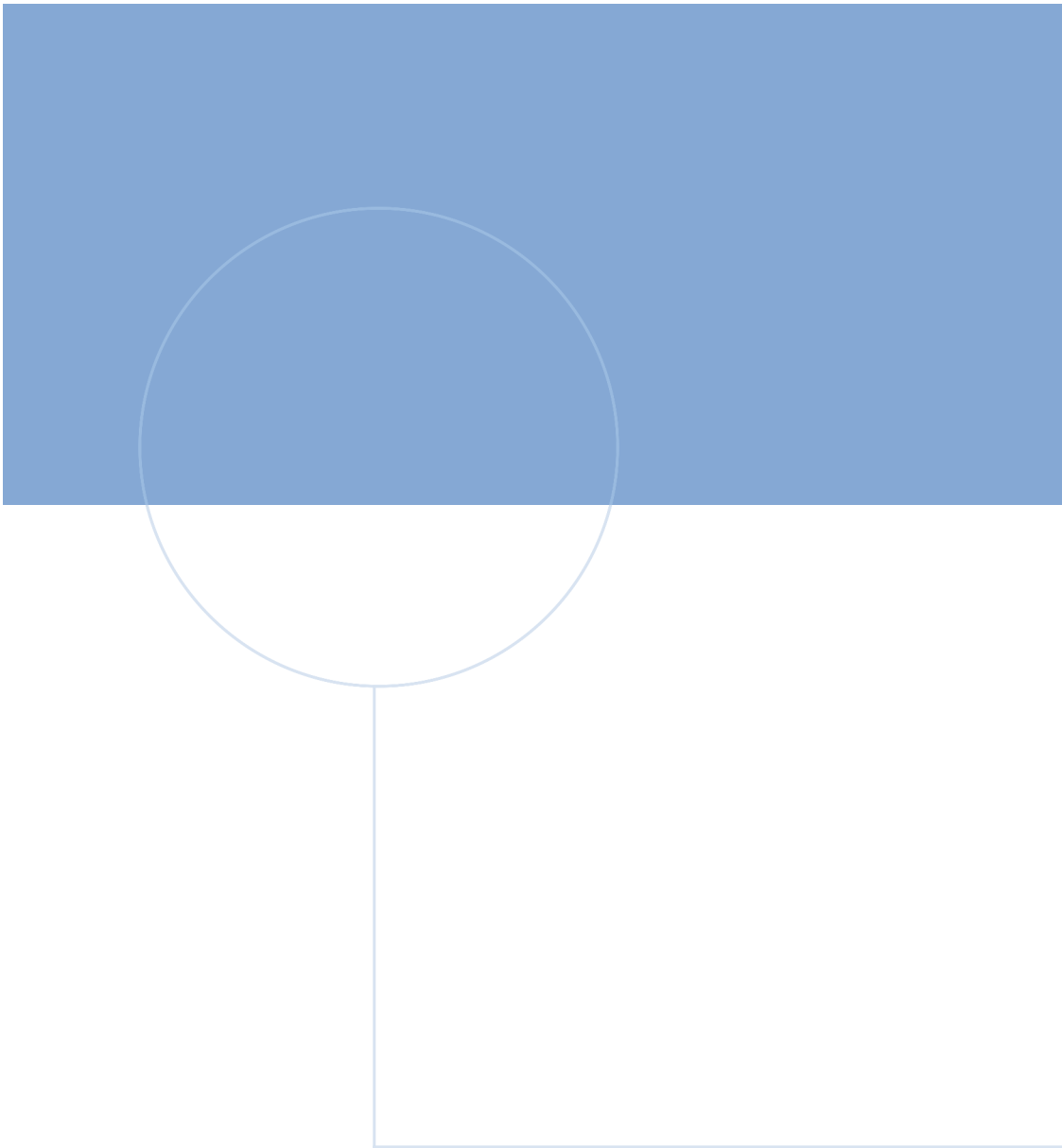
Unreal Engine Unreal Engine 4, en spillutviklingsmotor fra Epic Games, skrevet i C++. iii, 9, 61

Forkortelser

FOV (Engelsk) Field of View/(Norsk) Synsfelt. 26, 34

POI (Engelsk) Point of Interest/(Norsk) Interessepunkt. 14, 15

UE4 Unreal Engine. 8, 9, 11, 13, 15, 30, 39, 40, 45, 49, 52, 55, 56



NTNU
Kunnskap for en bedre verden