Masteroppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Det humanistiske fakultet
Institutt for musikk

Alexander Polanco Olsen

Teknisk Lyddesign i Dataspill

Hvordan utvikle et omgivelsesbasert lydsystem for en tredimensjonal spillverden

Masteroppgave i Musikkteknologi
Veileder: Trond Engum
Mai 2022

NTNU
Kunnskap for en bedre verden

Alexander Polanco Olsen

Teknisk Lyddesign i Dataspill

Hvordan utvikle et omgivelsesbasert lydsystem for en tredimensjonal spillverden

Masteroppgave i Musikkteknologi
Veileder: Trond Engum
Mai 2022

Norges teknisk-naturvitenskapelige universitet
Det humanistiske fakultet
Institutt for musikk

NTNU
Kunnskap for en bedre verden

Sammendrag

Denne masteroppgaven tar for seg hvordan man kan lage et lydsystem som gjør det enklere å implementere omgivelseslyder i en tredimensjonal dataspillverden. Oppgaven består av en praktisk og en skriftlig del. Oppgavens praktiske del har bestått av å utvikle en prototype av et omgivelsesbasert lydsystem. Den skriftlige delen av oppgaven presenterer relevant teori og praksis i forbindelse med lyd i tredimensjonale dataspill, og knytter dette opp mot arbeidet som er gjort i utviklingen av det omgivelsesbaserte lydsystemet.

Abstract

This master's thesis presents a way of creating a sound system that enables easier implementation of environmental sounds in a three–dimensional computer game. The thesis combines a practical component and a written report. The practical component is delivered as a prototype of an environmental sound system. The report covers relevant theory and common practices related to audio and sound in three-dimensional computer games. Furthermore, it connects these theoretical aspects to the work done in the development of the environmental sound system.

Forord

Gjennom mine fem år som student ved musikkteknologi på NTNU har jeg fått muligheten til å tilegne meg kunnskap innenfor et bredt spekter av ulike retninger i forbindelse med lyd og musikk. Tidlig i utdanningsløpet ble det likevel klart at min største interesse lå innenfor det audiovisuelle mediet, spesielt dataspill. I løpet av studietiden min har jeg blitt oppmuntret og utfordret til å utvikle min faglige kompetanse relatert til lyd i spill. Som et resultat av dette har både bachelorprosjektet og masterprosjektet mitt omhandlet dette temaet. Jeg vil anse meg selv som heldig som har fått muligheten til å kunne bruke disse prosjektene til å fordype meg i noe som jeg oppriktig brenner for. Dette gjelder denne oppgaven mest av alt. Det er noe spesielt med å kunne levere en masteroppgave etter fem år, som i sin reneste form er et sluttprodukt av min interesse og lidenskap. Min største takk går derfor til NTNU, som tilbyr dette enestående studieprogrammet.

I arbeidet med denne oppgaven er det mange mennesker som har hatt stor betydning for meg. Jeg vil først og fremst takke min veileder Trond Engum, for hans faglige og akademiske kompetanse. Han har alltid tatt seg tiden til å hjelpe og virkelig stilt opp når jeg har hatt behov for det. I tillegg til dette har han helt siden jeg skrev bacheloroppgaven min uttrykt stor tiltro til mine evner og oppmuntret meg i alle aspekter ved arbeid relatert til lyd i dataspill. Jeg vil takke Henriette Jenssen ved IO Interactive for gode samtaler og tilbakemeldinger i forbindelse med planleggingen av innholdet i dette prosjektet. Jeg vil også takke Marianne Austvik ved Digit Game Studio, for muligheten til å oppleve arbeidslivet i spillbransjen og alt det innebærer. Til slutt vil jeg takke familien min, som alltid har støttet meg i alle mine interesser og hjulpet med gjennom hele prosessen. De har vært grunnmuren som jeg har lent meg på og dette prosjektet hadde ikke vært mulig å gjennomføre uten dem.

Alexander Polanco Olsen

Trondheim 10. mai 2022

Innholdsfortegnelse

FIGURER: XII

1. INNLEDNING 1

1.1. OVERSIKT OVER INNLEVERT MATERIALE 1

1.2. MOTIVASJON OG BAKGRUNN 2

1.3. PROBLEMSTILLING 3

1.3.1. PROBLEMSTILLINGENS NØKKELBEGREP 4

1.4. OPPGAVENS OPPBYGNING 6

2. TEORI 8

2.1. PROGRAMVARER 8

2.1.1. SPILLMOTORER 8

2.1.2. MELLOMVARER 12

2.2. IMMERSJON 16

2.3. GRUNNLEGGENDE BEGREP 17

2.3.1. GENERELLE BEGREP INNENFOR SPILLUTVIKLING 17

2.3.2. BEGREP DIREKTE RETTET MOT LYD I SPILL 18

2.3.3. BEGREP INNENFOR AKUSTIKK OG FYSIKK 21

3. GENERELLE LYDPRAKSISER OG LYDKONVENSJONER I 3D SPILL 26

3.1. TREDIMENSJONALE AUDIO EMITTERE 26

3.2. ATMOSFÆRESYSTEM 28

3.2.1. TODIMENSJONALE ATMOSFÆRESYSTEM 28

3.2.2. TREDIMENSJONALE ATMOSFÆRESYSTEM 30

3.3. LYDUTBREDELSESSYSTEM 30

3.3.1. OBSTRUKSJON 31

3.3.2. OKKLUSJON 32

3.3.3. ROMKLANG 34

3.3.4. UTBREDELSESVEIER 34

4. OMGIVELSESBASERT LYDSYSTEM 37

4.1. UNDERSYSTEM FOR STATISK TREDIMENSJONALE AUDIO EMITTERE 38

4.1.1. CULLING 38

4.1.2. FORHINDRE LIKE AVSPILLINGSPUNKT 40

4.1.3. OPPSETT I DET OMGIVELSESBASERTE LYDSYSTEMET 43

4.2. UNDERSYSTEM FOR ATMOSFÆRELYDER 44

4.2.1. TODIMENSJONALE ATMOSFÆRELYDER 44

4.2.2. TREDIMENSJONALE ATMOSFÆRELYDER 49
4.3. UNDERSYSTEM FOR LYDUTBREDELSE 53
4.3.1. ROM & PORTALER 53
4.3.2. OBSTRUKSJON 57
4.3.3. OKKLUSJON 62
4.3.4. ROMKLANG 70
5. DISKUSJON/EVALUERING 74
5.1. OPPSUMMERING 74
5.2. EVALUERING 75
5.2.1. UTBEDRINGER OG VIDERE ARBEID 76
5.2.2. TILGJENGELIGHET OG BRUKSOMRÅDER 78
6. LITTERATURLISTE 79
7. SPILL 85
8. FILMER/SERIER 86
9. VEDLEGG 87
9.1. BILDEFILER 87
9.2. VIDEOFILER 90
9.3. SPILLFILER 91
9.4. PROSJEKTFILER 92
9.5. NAVIGASJONSVEILEDNING FOR PROSJEKTET 93

Figurer:

Figur 1: Eksempler på ulike spillmotorer. 9
Figur 2: Utsnitt av brukergrensesnittet i UE4. 10
Figur 3: Eksempel på kode i UE4, ved bruk av blueprints. 11
Figur 4: Oversikt over bruk av mellomvarer i store og mindre titler. 13
Figur 5: Utsnitt av brukergrensesnittet i Audiokinetic Wwise 14
Figur 6: Illustrasjon av det omgivelsesbaserte lydsystemet sin rolle. 15
Figur 7: Attenuasjonsradius til en audio emitter.. 18
Figur 8: Illustrasjon av en attenuasjonskurve. 19
Figur 9: Eksempel på kodeutsnitt som sender beskjed til Wwise om å aktivere en event. 20
Figur 10: Eksempel på hvordan RTPC kurvene i Wwise kan se ut.. 21
Figur 11: Representasjon av refleksjoner 22
Figur 12: Illustrasjon av de ulike rom modene. 23
Figur 13: Illustrasjon av tidsforsinkelse mellom direktelyd, tidlige og sene refleksjoner. 24
Figur 14: Visuell representasjon av diffraksjon. 24
Figur 15: Visuell representasjon av akustisk overføring 25
Figur 16: Eksempel på mengde audio emittere som kan plasseres i et område. 27
Figur 17: Eksempel på oppsett av et Sound Shape Tool. 29
Figur 18: Visuell representasjon av Random FX 30
Figur 19: Visuell representasjon av obstruksjon. 32
Figur 20: Visuell representasjon av okklusjon. 32
Figur 21: Figur som viser hvordan rooms og portals i Wwise ser ut. 36
Figur 22: Utdrag fra koden som definerer reglene for kontroll av en audio emitter.. 40
Figur 23: Illustrasjon av en dobbeltradius for atmosfærelyder. 41
Figur 24: Utsnitt av Seek – funksjonen i Wwise 42
Figur 25: Utsnitt av spillobjektet for en statisk tredimensjonal audio emitter. 43
Figur 26: Eksempel på en statisk tredimensjonal audio emitter 43
Figur 27: Illustrasjon av emitteren til 2D - atmosfærelyder dersom spilleren er utenfor regionen 45
Figur 28: Illustrasjon av emitteren til 2D – atmosfærelyder dersom spilleren er innenfor regionen 45
Figur 29: Illustrasjon av direksjonalitetskonseptet for de todimensjonale atmosfærelydene. 48
Figur 30: Utsnitt som viser valg for om atmosfærelydene skal behandles ulikt eller ikke. 49
Figur 31: Utsnitt fra Wwise som viser eksempel på randomisering av tredimensjonale atmosfærelyder.. 50
Figur 32: Utsnitt av det interaktive aspektet ved undersystemet for atmosfærelyder.. 51
Figur 33: Utsnitt av Rom og Portaler i det omgivelsesbaserte lydsystemet 54

Figur 34: Eksempel på en generisk første – persons karakter i et tredimensjonalt spill. 58
Figur 35: Eksempel på verdiene man kan skrive inn for obstruksjon i et rom. 59
Figur 36: Eksempel på endring av parameter som et resultat av obstruksjon gjennom en RTPC 60
Figur 37: Utsnitt av kurvene for obstruksjon og okklusjon i Wwise sitt innebygde system..... 61
Figur 38: Illustrasjon av okklusjonsberegning for ett rom..... 65
Figur 39: Illustrasjon av beregningsmetoden for okklusjon utprøvd under utviklingen av Hitman 66
Figur 40: Illustrasjon av hvordan portalens tilstand endrer beregningsreglene for okklusjon. 67
Figur 41: Illustrasjon av hvordan okklusjon beregnes for flere rom. 69
Figur 42: Illustrasjonene av hvordan lydutbredelsesveiene endres ut ifra portalenes tilstand..... 69
Figur 43: Illustrasjon av image source (speilbilde) metoden 72
Figur 44: Representasjon av beregningene for å finne avstanden til lyden (tidlige refleksjoner)..... 73
Figur 45: Flytskjema for implementasjonen av romklang i undersystemet for lydutbredelse. 73

1. Innledning

1.1. Oversikt over innlevert materiale

I denne masteroppgaven vil det bli levert to typer arbeid; en skriftlig tekst og et praktisk arbeid i form av en prototype av et utviklet lydsystem. Teksten vil bli levert i et standard PDF format og i henhold til NTNU sine retningslinjer.

Det praktiske arbeidet vil bli levert som prosjektfiler med tilhørende kildemateriale for kode, lydfiler og spillobjekt. Hovedmateriale for systemet finnes i den vedlagte mappen ved navn *Vedlegg,* og heter *Vedlegg/UE4 – Wwise Prosjekt/Env_SoundSystem/Env_SoundSystem.uproject.* Prosjektfiler for lydarbeid finnes i vedlegget *Vedlegg/UE4 – Wwise Prosjekt/Env_SoundSystem/Env_SoundSystem_WwiseProject/ Env_SoundSystem_WwiseProject.wproj*. For å åpne prosjektfilene kreves det at man har installert Unreal Engine 4 og Audiokinetic Wwise. Anbefalte versjoner for dette prosjektet er Unreal Engine 4.27.2, og Wwise 2021.1.6.7774. Det kreves at man bruker *Windows 10* som operativsystem. Det er også lagt med en veiledning for å enklere kunne finne frem til ulike deler av det nevnte hovedmateriale i Unreal Engine 4. Denne ligger i *vedlegg 9.5* og heter *Navigasjonsveiledning for prosjektet*. Det anbefales å bruke denne første gangen man åpner prosjektet i Unreal Engine 4.

Det er også vedlagt videoklipp, et spill og bilder som forklarer og viser det praktiske arbeidet. Dersom man ikke har mulighet til å installere de nødvendige programvarene eller åpne prosjektfilene, kan disse vedleggene brukes som en alternativ måte å oppleve det praktiske arbeidet på. For videodemonstrasjon av hovedfunksjonen og resultat, se vedlegg *Vedlegg/Filmer/Env_Sound_System_Concept_Walkthrough_W_Music.MP4* og *Env_Sound_System_Implementation_Example_W_Audio.MP4.* For en interaktiv opplevelse av oppgavens konsept, se vedlegg *Vedlegg/Environmental Sound System – Playable Version/WindowsNoEditor/Env_SoundSystem.exe.* Bilder av kode, system og oppsett finnes i vedlegget *Vedlegg/Bilder.* Spillfilen vil kun være kompatibelt med operativsystemet *Windows 10*.

Selve spillet og musikken som er utviklet og produsert i forbindelse med denne oppgaven er laget selv. I tillegg vil alt av det grafiske innholdet og diverse lydklipp enten være egenprodusert eller kjøpt, og eies av meg. Dette betyr at alt innhold i denne oppgaven er tillatt å publisere og bruke, uten risiko for rettighetsbrudd. Det ønskes at gjennomgangen av det praktiske arbeidet oppleves i så høy oppløsning som mulig. Dette gjelder lyd og bilde. Det anbefales derfor å bruke headset eller øretelefoner for lytting, og visuell fremvisning på egnet skjerm; enten PC – skjerm, eller TV.

1.2. Motivasjon og bakgrunn

Helt siden jeg var liten har videospill vært en stor del av livet mitt. Jeg har kjempet meg gjennom gravplasser og dystre sletter i MediEvil (Sony Entertainment, 1998) og utforsket vakre landskap i The Witcher 3 (CD Projekt Red, 2016), hvor jeg jeg brukte timer på å leve meg inn i historier som var så velregisserte og gjennomførte at jeg kunne miste all fatning om tidsbegrep. Med min økende interesse for lyd oppstod det senere en ny fascinasjon for hvilken rolle denne kunne ha i spill og hvordan den kunne resultere i at man levde seg mer inn i handlingene som fant sted, og tilsynelatende ble mer knyttet til karakterene og den verden de befant seg i.

Denne fascinasjonen ble bare større, og førte etter hvert til et ønske om kunne forstå hvordan det var mulig å lage en hel verden av lyd, på en liten CD eller i en datamaskin. Dette utviklet seg naturligvis til at jeg selv ønsket å lage lyd til spill. Jeg lastet derfor ned et testspill, tok frem en mikrofon og begynte å ta opp lyder. Uten å ofre det en tanke, antok jeg at lydene i spillet skulle høres ut og oppleves akkurat slik de gjorde i virkeligheten og at dette var tatt til høyde for i programmet da lydene ble lagt inn i spillet. Hvorfor skulle de ikke det? De gjorde jo det for alle de store spillene som ble utviklet med samme programvare? Slik var det dessverre ikke. Det forsøket endte derfor i flere timer med å prøve å forstå hvorfor lyder i et hus var like tydelige uavhengig om spilleren var inne i eller utenfor det. Av kjappe internettsøk la jeg merke til at dette som regel ble løst gjennom en eller annen form for programmering, som jeg på det tidspunktet anså som avansert på bakgrunn av min begrensede erfaring med slikt. Dette skulle vel ikke være nødvendig? Jeg ville jo bare lage lyder og legge dem inn i spillet, ikke programmere. Det måtte jo finnes en enklere måte å gjøre det på?

Til min store frustrasjon, viste det seg at det ikke var det. Tilsynelatende var det noen eksisterende programvarer og verktøy som gjorde det tekniske aspektet litt enklere, men det ble mer og mer klart at det ikke var noen vei rundt programmeringen. Jeg aksepterte at dette var et konsept jeg måtte bli bedre kjent med for å kunne få til det jeg ville, men hvor skulle jeg derimot starte? Jeg innså også at i tillegg til dette, hadde jeg ingen forståelse av hvordan jeg kunne bygge opp og forme en lydverden slik det var gjort i favorittspillene mine. Hvordan ulike lyder var satt sammen og utgjorde komplekse og varierte lydlandskaper tilsvarende realistiske omgivelser, var noe jeg på det tidspunktet nærmest anså som magi. Det ble derfor innlysende at jeg også måtte lære om praksiser og konsepter i forhold til lyd i spill, og hvilke roller disse kunne ha.

Overraskende nok, var det ikke slik at denne informasjonen var enkel å finne. Det viste seg at det var vanskelig å få tak i utfyllende informasjon om kjente praksiser innenfor temaet, og selv om det eksisterte utallige småleksjoner på ulike nettsider om hvordan man for eksempel la på fotsteg – lyder i spill, var det ingen som ga en detaljert og helhetlig oversikt over hva som er vanlig å gjøre, hvordan man gjør det og hvorfor. I tillegg ble det klart at de eksisterende leksjonene ikke tok til høyde for mer helhetlige og sammensatte løsninger, men kun fokuserte på ett enkelt konsept som ikke kunne brukes i større sammenhenger. Jeg oppdaget også at en del av denne kunnskapen var vanskelig å få tak i hvis man ikke allerede var en del av bransjen, og gjerne kjent i miljøet. I noen tilfeller, virket det nesten som om ulike spillselskap ønsket å «verne» om sine fremgangsmåter, og holde informasjon og kunnskap for seg selv.

1.3. Problemstilling

Alt som blir beskrevet i underkapittelet ovenfor fungerer som motivasjon for det prosjektet som presenteres i denne masteroppgaven. På bakgrunn av dette ønsker jeg i større grad å *frigjøre lydarbeidet i spill fra kjente tekniske begrensninger.* Dette gjøres ved å presentere et alternativ som gjør det mulig for en lyddesigner å *enklere integrere lyd i en programvare* gjennom metoder som baserer seg på, og som har blitt til som et resultat av praksiser og konvensjoner i denne bransjen. Alternativet som blir presentert bør *ikke begrenses av kun en programvare*. Dette har som mål å *gi mer tid til det kreative aspektet ved lyddesign*. I tillegg ønskes det at prosjektet kan fungere som

et læringsverktøy for alle som ønsker å bli bedre kjent med praktisk bruk av lyd i dataspill. Det har også som mål å kunne fungere som et *verktøy for allerede erfarne mennesker i bransjen*. Dette fordi at det i denne oppgaven presenteres løsninger som kan bygges videre på, endres og utvikles ut ifra hva behovet er. Målene som er presentert i dette avsnittet danner grunnlaget for denne masteroppgavens problemstilling, som lyder som følger:

Hvordan lage et lydsystem som gjør det enklere å implementere omgivelseslyder i en tredimensjonal spillverden, basert på lydkonvensjoner og lydpraksiser innen dataspill.

1.3.1. Problemstillingens nøkkelbegrep

For å kunne forstå problemstillingen vil det her redegjøres for viktige nøkkelbegrep. Selv om begrepsforklaringer blir presentert senere i oppgaven, vil nøkkelbegrepene forklares så tidlig som mulig, slik at man får en tilstrekkelig forståelse av oppgavens presenterte problemstilling.

I spillutvikling vil man som regel møte på begrepet *system*, i ulike sammenhenger. Hva det faktisk betyr og innebærer vil i noen grad variere, men en god definisjon kan være at et system kan sees på som en kombinasjon av ulike mekanismer som mottar noe, og resulterer i noe annet (Boards To Bits Games, 2018). Et eksempel på dette kan være et system for hvordan karakteren i et spill skal bevege seg på ulike måter. I dette tilfellet vil man ofte lage et større system som består av mindre mekanismer og system for hvordan spilleren går, løper, stopper og hopper. På den måten har man laget et bevegelsessystem for karakteren. Mindre system som er en del av et større et, vil i denne oppgaven omtales som *undersystem.* I spillutvikling består som regel en stor andel av implementasjonsløsningene av en form for systembygging. For å enklere skille mellom det lydsystemet som blir utviklet i denne masteroppgaven og andre system, vil det ut ifra problemstillingens ordlyd bli omtalt som et *omgivelsesbasert lydsystem.*

For å kunne forstå begrepet *omgivelseslyder*, er det først nødvendig å redegjøre for noen andre begrep som er etablerte for lyd i ulike medier, hvorav de fleste har opphav fra film. *Lydbilde,* er et

samlet begrep for hele det lydlige aspektet i film og spill (Buhler et al., 2010, s. 430). *Lyddesign,* er et begrep som har fått utallige definisjoner; noen skiller lyddesign fra andre lydlige elementer som dialog og musikk (Gjerdsjø, 2014, s. 10), mens andre ser på lyddesign som altomfattende i alle aspekter av mediets lyd, inkludert musikk og dialog (Anderson, 2018). I denne oppgaven benytter vi oss av den første definisjonen. Lyddesignet kategoriseres deretter i *effektlyd* og *kontentum (*Gjerdsjø, 2014, s. 10). Effektlyder er lyder som tilhører konkrete hendelser, som fotsteg eller slag, mens kontentum kan beskrive lyden av omgivelsene (Ibid.). Disse kan være atmosfærelyder, vindkast, eller fuglekvitring. Disse utelukker ikke nødvendigvis hverandre. *Atmosfærelyder,* også betegnet som *ambiens,* går som regel under (og er i noen tilfeller det samme som) kontentum og beskriver lyder som er tilknyttet en atmosfære i lydverdenen. Det er ofte lyder som ligger i bakgrunnen av det totale lydbildet (Chattopadhyay, 2017, s. 354). Eksempler på dette kan være lyden av vind, eller stille vann ved en brygge. Både kontentum og atmosfærelyd kan ha den felles egenskapen at vi ikke nødvendigvis ser lydkilden (Nygård, 2016, s. 9), også kalt *off – screen* lyd (Chion, 1994, s. 208).

Ifølge Eli Nygård, er *omgivelseslyder* er et samlebegrep for ordene atmosfærelyder, effektlyder og kontentum (Nygård, 2016, s. 13). Det blir nevnt at dette begrepet ikke begrenser en lyd til å være ubetydelig i bakgrunnen, slik man kan tolke kontentum og atmosfærelyder ut ifra definisjonen ovenfor. Dette vil passe godt med beskrivelsen av lydene som er tiltenkt dette prosjektet. Selv om lydsystemet som presenteres i denne masteroppgaven kan legge til rette for stor bruk av atmosfære og kontentum – lyder, vil det ikke være begrenset til det. Lyder som kanskje tar større oppmerksomhet, og tilhører konkrete hendelser kan derfor uproblematisk benyttes i det omgivelsesbaserte lydsystemet. I tillegg til dette vil begrepene atmosfærelyder og kontentum ofte *kun* omtales som atmosfærelyder i spillbransjen. På grunn av dette vil oppgavens bruk av begrepet atmosfærelyder eller atmosfæresystem, også kunne inneholde kontentum. Dette blir aktuelt for kapittel 3.2 og 4.2.

Hvordan lyd skal høres ut og brukes i et spill er bestemt av ulike og varierende faktorer og valg. Disse kan være estetiske, tekniske, eller basert på ulike visjoner for hvilken rolle lyden skal ha. Det kan også være et resultat av hvordan spillverden ser ut. Lyd under vann vil for eksempel ha

andre egenskaper enn lyd over vann. Dersom man ønsker å lage et spill hvor handlingen er satt i en undervannsverden, kan dette påvirke hvordan man vil at lyden skal høres ut. Et annet eksempel kan være at dersom man i et spill for eksempel styrer en robot, så vil kanskje alle lydene som lages fra den være mekaniske og unaturlige, i motsetning til om man styrer et menneske. Ettersom at hvordan spill-lyd høres ut er et resultat av slike ulike faktorer og valg, kan det være vanskelig å finne praksiser og konvensjoner for alle typer lyder for alle typer spill.

Det som derimot vil være felles for de fleste spillverdener og audiovisuelle fiksjonsunivers er at de består av lyder som beskriver omgivelsene hvor en handling utspiller seg. Selv om en undervannsverden kan høres annerledes ut enn en mer realistisk spillverden, vil begge ha lyder som tilhører omgivelsene. Det samme gjelder en spillverden hvor man styrer enten en robot eller et menneske. Selv om lydene vil være annerledes, vil *bruken av omgivelseslyder* være felles og anvendbar for flere typer spill. Bruken av omgivelseslyder gjelder ikke bare for spillmediet, men også film. Dette reflekteres blant annet i litteratur og lydpraksiser, hvor spesielt atmosfærelyder og kontentum brukes i lyddesignet til disse mediene. Omgivelseslydene bidrar også til at et fiksjonsunivers oppleves mer ekte og sammenhengende (Bridgett, 2013), noe som gjør at man lever seg mer inn i det. Dette fenomenet skal forklares i kapittel 2.2. Innholdet i disse to avsnittene er grunnen til at det i denne masteroppgaven har blitt valgt å fokusere på omgivelseslyder.

1.4. Oppgavens oppbygning

Denne masteroppgaven består av en praktisk og en skriftlig del. Den praktiske delen består av et arbeid hvor jeg har laget en *prototype* av et lydsystem som kan brukes som et verktøy for enklere implementasjon av omgivelseslyder i spill, basert på, og som et resultat av lydpraksiser og lydkonvensjoner i spillbransjen. Prosjektfilene for det omgivelsesbaserte lydsystemet er inkludert som vedlegg. Det er også vedlagt et kort spill som forklarer og viser ulike konsept ved det omgivelsesbaserte lydsystemet. Dette spillet inneholder også en lydlagt testbane for å vise det i sin helhet. I tillegg er det lagt med en videogjennomgang av dette spillet og en videodemonstrasjon av hvordan systemet brukes og settes opp i praksis. Det er vedlagt bilder og filmer av kode, spesifikke konsept og ulike oppsett av systemet. Det er også lagt med en veiledning for å enklere

kunne finne frem til ulike deler av prosjektet i programvaren som blir benyttet. Disse kan finnes i kapittel 9, *vedlegg.* Vedleggene vil bli referert til og forklart i større detalj underveis i den skriftlige delen av oppgaven. Bruken av videoklipp fører til enklere demonstrasjon av systemet ettersom man kan referere til en spesifikk tidsmarkering, og filtypen (.MP4) støttes av flere ulike enheter. I den skriftlige delen av oppgaven vil derfor videoklippene henvises til i større grad enn de vedlagte prosjektfilene og spillet.

Denne teksten utgjør den skriftlige delen av oppgaven. Her vil det først bli redegjort for teoretisk bakgrunn i form av relevante begrep, forklaringer av ulike programvarer og teoretiske konsept. Dette gjøres for å gi en grunnleggende forståelse av terminologi, ulike aspekter ved lyd i audiovisuelle medier og ulike fenomen innenfor akustikk som brukes i forbindelse med oppgaveløsningen. Deretter vil det bli gitt en gjennomgang av etablerte løsninger og praksiser for lyd i spillbransjen. Det som blir presentert her vil fungere som utgangspunkt for de løsningene som brukes i det praktiske arbeidet. Til slutt vil innholdet oppsummeres, og det vil reflekteres rundt eget arbeid og hva som kan gjøres videre i fremtidig utvikling av prosjektet.

2. Teori

I dette kapittelet skal det gis en faglig oversikt over relevante programvarer, teori og begrep, slik at man får en god grunnforståelse for innholdet i oppgaven. Denne grunnforståelsen vil være viktig å inneha, ettersom innholdet som blir presentert i kapittel 3 og utover bygger på det som blir redegjort for i dette kapittelet. Det som blir presentert her vil altså være viktig for å kunne forstå de generelle lydpraksisene og konvensjonene i tredimensjonale spill, og valg og begrunnelser som er tatt i forbindelse med utviklingen av det omgivelsesbaserte lydsystemet.

2.1. Programvarer

For å danne en grunnforståelse for innholdet i denne oppgaven, kan det først være gunstig å kjenne til hva man bruker for *å lage* dataspill. Grunnen til dette er fordi at lydarbeid i spillbransjen ofte innebærer en mye større tilnærming til ren spillutvikling enn man kanskje skulle tro. Mye av lydarbeidet i spill består med andre ord av teknisk arbeid, så vel som kreativt arbeid. I en analyse av de mest brukte termene for stillingsutlysninger for lyddesignere i spillselskap, omhandler rundt 50% av de implementasjonsegenskaper (Schmidt, 2020). I tillegg til dette krever all form for implementasjon av lyd i spill en eller annen form for interaksjon mellom den som implementerer lyden og programvaren for spillutvikling. Ettersom programvarer er essensielle for utviklingen av spill, vil det bli redegjort for noen programvarer og gitt en beskrivelse av hvilke roller og funksjoner de har i en spillutviklings - sammenheng.

2.1.1. Spillmotorer

En *spillmotor*, eller *spill – engine*, er i korte trekk en programvare som muliggjør utviklingen av dataspill. Tidligere har dataspill blitt utviklet fra bunnen av gjennom programmering og kode, ofte som et resultat av begrenset teknologi. Dette førte til at spilldesign på 80 – tallet hovedsakelig bestod av hardkodede regelsett med begrensede antall nivå og grafikkdata. Rundt midten av 1990 – tallet skjedde derimot en økning i popularitet i spillverden, i forbindelse med introduksjonen av tredimensjonale spill, og da gjerne «first person shooter» spill. Dette er en spillsjanger som går ut på at man styrer en karakter hvor perspektivet oppleves fra karakteren sitt synsfelt, derav begrepet

første - person. Ofte vil man i slike spill benytte seg av en eller annen form for skytevåpen for å uskadeliggjøre eventuelle fiender.

Populariteten til denne spillsjangeren førte til et behov for å skape et raskere utviklingsforløp. Dette resulterte i at man lisensierte deler av den selvlagde programvaren til andre utviklere, slik at de kunne bruke det for sine egne grafiske formål, karakterer og innhold (Andrade, 2015, s. 1). Slike programvarer ble med andre ord mindre spesifikke for ett spill, og mer fleksible slik at de kunne gjenbrukes og skreddersy innhold innenfor sine egne rammeverk.

I dag finnes det et stort antall spillmotorer. Mange kan brukes mot betaling, som en standard programvare, mens andre er begrenset til egne spillselskap (som regel fordi de har laget det selv). Flere av disse programvarene har fellestrekk, ofte i hva de kan gjøre og muligheten de gir. Disse blir eksemplifisert i neste avsnitt. De vil likevel ha noen forskjeller som kan være avgjørende for hvilken man bør velge når man skal utvikle et spill. En stor andel av disse forskjellene finner man i filosofien utviklerne deres har i forhold til spillutvikling; noen spillmotorer krever ikke kunnskap om programmering, noen er basert på populære web – teknologier og noen kan ha åpen kildekode og videreutvikles av andre brukere (Christopoulou & Xinogalos, 2017, s. 21).

Figur 1: Eksempler på ulike spillmotorer (hentet mars 15, 2022 fra https://www.linkedin.com/pulse/10-criteria-adopting-game-engine-oluwaseye-ayinla).

De funksjonalitetene som en spillmotor vanligvis har, og som oftest inngår i alle er: *Grafisk rendering*, som behandler alt av grafikk i spillet; *fysikkmotor*, som brukes til å designe og simulere virkelighetsbaserte aktiviteter, reaksjoner og bevegelser; *animasjon*, som brukes til å livliggjøre grafisk design; *kunstig intelligens*, som håndterer spillets oppførsel og logikk; *lydmotor*, som håndterer alt av lyd og musikk; *streaming*, eller *networking* som støtter flerspilling sammen over internett (Andrade, 2015, s. 2). Det finnes flere populære spillmotorer. Eksempler på disse kan være *Unity* (Unity, u.å.), *Unreal Engine 4* (Epic Games, u.å.b), *ARKit* (Apple, u.å.) og *GameMaker* (Gamemaker, u.å.).

2.1.1.1. Unreal Engine 4

Ettersom problemstillingen tar for seg utviklingen av et lydsystem for å enklere implementere omgivelseslyder i en tredimensjonal spillverden, er Unreal Engine 4 (UE4) spillmotoren som har blitt valgt for å utvikle dette. Det er som nevnt tidligere mange likheter mellom spillmotorer, og man kan som regel oppnå samme resultat med flere av dem. Det er likevel noen spesifikke grunner til at denne programvaren har blitt valgt i forbindelse med oppgaven.

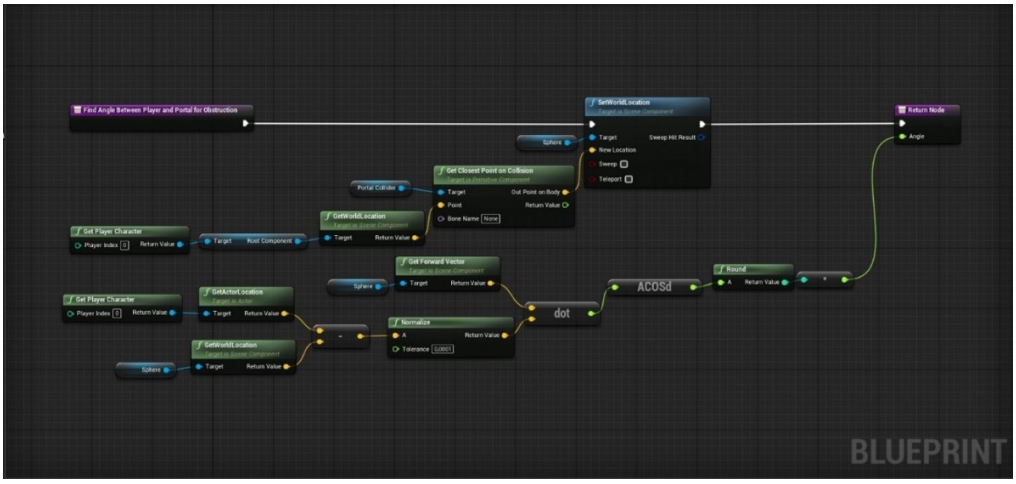
Figur 2: Eget utsnitt av brukergrensesnittet til UE4.

Til å begynne med, er dette en spillmotor som egner seg veldig godt for tredimensjonale spill. Ettersom jeg i grove trekk ønsker å lage et omgivelsesbasert lydsystem for tredimensjonale spill,

var UE4 mer egnet til dette da det har bedre grafisk fidelitet, enn for eksempel Unity, som har større fokus og flere verktøy for 2D grafikk. Unreal Engine 4 er en av de mest brukte tilgjengelige spillmotorene for utviklingen av tredimensjonale spill, på grunn av dets grafiske fokus (Buckley, 2021).

Ettersom dette er en mer populær programvare for tredimensjonale spill, vil det omgivelsesbaserte lydsystemet kunne tilføre mer for brukergruppen og lyddesignere som jobber med denne sjangeren dersom det utvikles i Unreal Engine 4. I tillegg har denne spillmotoren et stort nærvær blant *AAA* selskap. *AAA* selskap er en samlebetegnelse for de største og mest etablerte spillselskapene. Større relevans for sjanger og type spill er da en av årsakene til at jeg har valgt å bruke denne spillmotoren.

UE4 bruker i utgangspunktet programmeringsspråket C++. Dette kan for noen klassifiseres som et mer avansert kodespråk, enn for eksempel C#, som blir brukt i spillmotoren Unity. Likevel tilbyr UE4 en alternativ måte å utvikle på, som heter «visual scripting», eller visuell programmering. I spillmotoren kalles dette for «Blueprints» og består i all hovedsak av noder med innebygde funksjoner som man kan koble sammen. Dette er et godt alternativ da man kan gjøre det samme som i C++ eller C#, men med mindre kodeferdigheter, noe som passer godt da jeg anser mine programmeringsferdigheter som begrenset.



Figur 3: Eget utsnitt av kodeeksempel i UE4, ved bruk av blueprints.

En annen årsak til at jeg har valgt denne spillmotoren for dette prosjektet, er fordi jeg selv har mye erfaring med implementasjon og lydarbeid i det. Dette gjør det mulig å effektivt utvikle og feilsøke, ettersom programvaren er kjent. I tillegg til dette har jeg også et godt nettverk å lene meg på hvis det skulle være behov for hjelp, med utviklere som har lang erfaring med denne spillmotoren.

2.1.2. Mellomvarer

I spillbransjen er det ikke uvanlig at mange selskap benytter seg av spillmotorens egen lydmotor for implementering av lyd, da dette ofte er et bedre økonomisk alternativ enn å skaffe lisenser for andre programvarer. Likevel er det mange som velger å jobbe med lyd i såkalte *mellomvarer.* Mellomvarer er programvarer som fungerer som et mellomledd mellom et lydopptaks – og behandlingsprogram (DAW) og spillmotoren. Disse gjør det mulig å enklere implementere lyd i spillmotorene. Noe av eksemplene på hva disse kan gjøre er: *kontrollere lydegenskaper* (som for eksempel frekvensinnhold) slik at spillkoden ikke trenger å gjøre det, *støtte for sanntid DSP effekter* (digital lydprosessering), *sørge for modulering og automasjon* basert på spill – parameter (se begrepet *RTPC* i kapittel 2.3.2), *innebygde profileringsmuligheter* (for sjekk av CPU og lagring) og *støtte for VR* (Virtual Reality). I all hovedsak er en mellomvare et verktøy for enklere implementasjon av lyd i en spillmotor.

2.1.2.1. Fordeler og ulemper

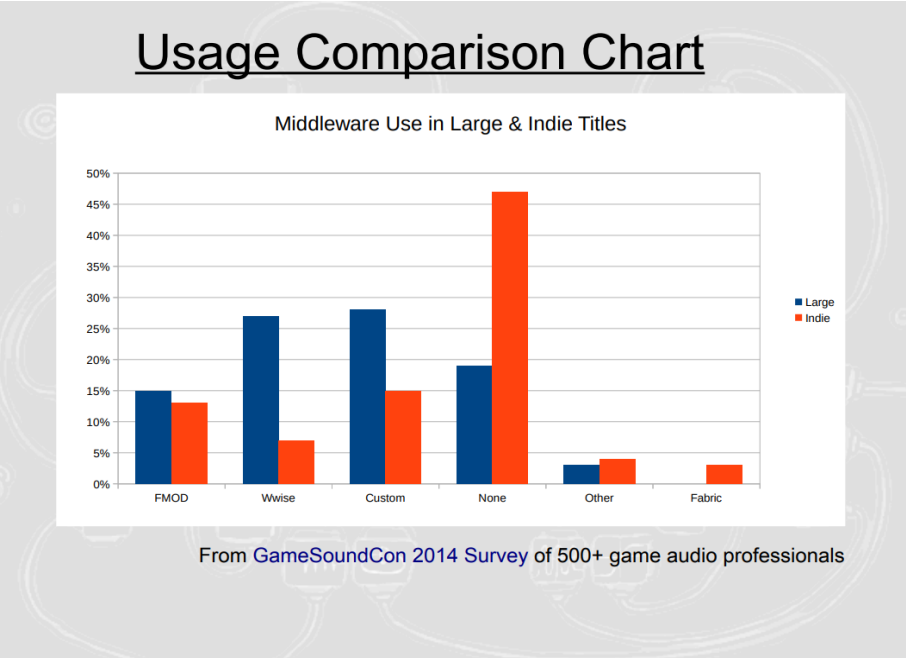
Fordeler som mellomvarer gir spillutviklere er blant annet bedre komprimering av lydfiler og avspillingsmulighet rett fra harddisken, uten å først måtte laste det inn i maskinens minne. Det gir også enklere mulighet for arbeid med interaktiv lyd, som kan bli jobbet med i forkant eller parallelt med selve spillutviklingen. En av de største fordelene med mellomvarer er likevel at de sørger for mindre kodenødvendighet for lydintegrering, altså enklere lydimplementasjon.

Ulemper med mellomvarer er at de kan inneholde produksjonsfeil (*bugs*), som kan være vanskelige å løse, ettersom man ikke har tilgang til kildekoden. De vil også ha flere begrensninger enn rene

kodebaserte lydløsninger, ettersom de består av et sett med forhåndsprogrammerte regler. I tillegg til dette må man kjøpe en lisens for å kunne publisere spillet sitt med en slik mellomvare. Mellomvarene har ofte ekstra utvidelser, eller «plugins», som også koster penger. Det må presiseres at selv om mellomvarer fjerner noe av behovet for programmering og gjør det litt enklere å implementere lyd i en spillmotor, *betyr det ikke at den fullstendig fjerner nødvendigheten for programmering*. Mellomvarer trenger alltid å kommunisere med spillmotoren, ettersom den responderer på hva som skjer i spillet. Dette vil alltid skje gjennom kode.

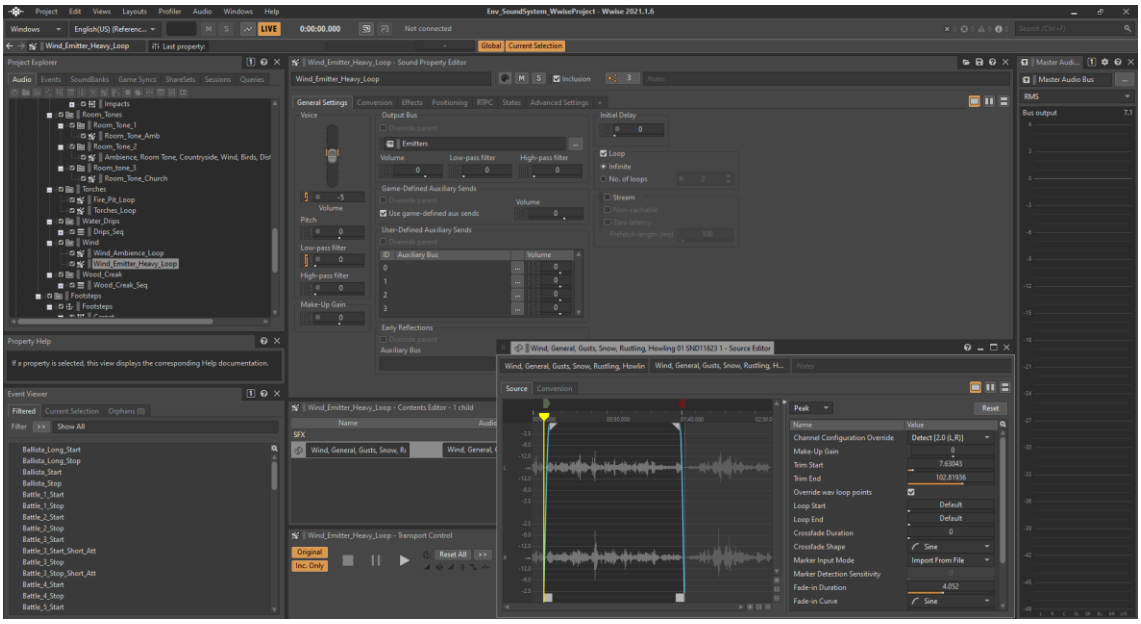
2.1.2.2. Audiokinetic Wwise

Audiokinetic Wwise er en veldig populær mellomvare i spillbransjen. Ifølge en undersøkelse fra GameSoundCon i 2014 bruker mesteparten av AAA selskapene som regel enten egenlagde mellomvarer eller Wwise (Paul, 2015).



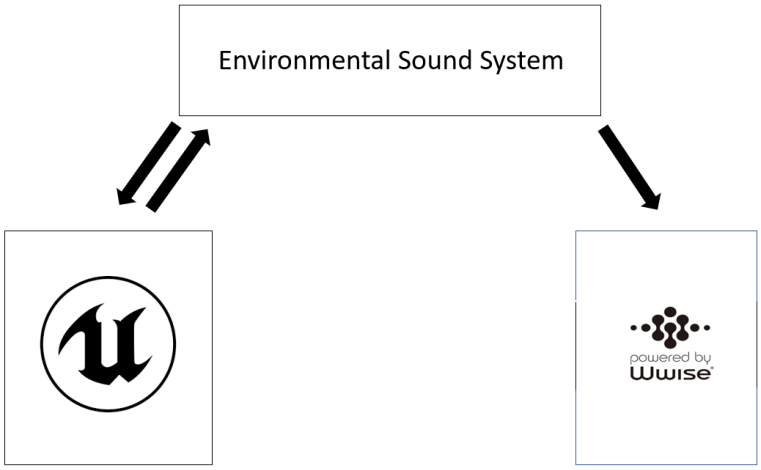
Figur 4: Oversikt over bruk av mellomvarer i store og mindre titler (Hentet mars 16, 2022 fra https://videogameaudio.com/FullIndie-Apr2015/GameAudioMiddleware-FullIndie-SchoolOfVideoGameAudio-LPaul-Apr2015.pdf).

I forhold til funksjonalitet er ikke Wwise veldig ulik andre mellomvarer på markedet. Selv om Wwise tillater litt mer kompliserte oppsett enn for eksempel *FMOD* (Firelight Technologies, u.å.), vil mesteparten av funksjonaliteten til disse mellomvarene være like. Wwise har derimot noen eksterne funksjoner i form av plugins og utvidelser som kan være nyttige å benytte seg av, som for eksempel *Wwise Reflec*t (beregninger av tidlige lydrefleksjoner). I tillegg har Wwise en egen funksjon for behandling av lydutbredelse og virtuell akustikk, som forklares mer i kapittel 3.3.4.



Figur 5: Eget utsnitt av brukergrensesnittet til Audiokinetic Wwise.

I denne oppgaven brukes Audiokinetic Wwise som mellomvare. Hvilken rolle denne har i forbindelse med det omgivelsesbaserte lydsystemet kan i grove trekk forklares slik: Det omgivelsesbaserte lydsystemet består av kode i spillmotoren som mottar ulik data og informasjon ut ifra hva spilleren gjør og hvor spilleren befinner seg. Denne informasjonen blir sendt videre til Audiokinetic Wwise og styrer aktivering av lydhendelser (forklares nærmere i kapittel 2.3.2) og endringer av ulike parameter. Disse lydhendelsene og parameterne kan være satt til å for eksempel spille av ulike lyder, endre volumnivå eller filtrere ulike frekvenser. Hvordan mellomvaren i praksis blir brukt i sammenheng med det omgivelsesbaserte lydsystemet vil forklares i kapittel 4.



Figur 6: Egen illustrasjon som viser hvordan det omgivelsesbaserte lydsystemet sin rolle i UE4 jobber med mellomvaren. Man skriver inn ulike verdier i systemet som bestemmer et sett med koderegler i spillmotoren. Spillmotoren bruker disse kodereglene til å registrere ulik data basert på spillerens handlinger og posisjon. Denne dataen blir gjennom systemet sendt til Wwise, som igjen bruker dataen til for eksempel å endre verdier for ulike parameter, som volumnivå.

I innledningen ble det nevnt at det omgivelsesbaserte lydsystemet ikke burde være begrenset til én programvare. Dette systemet vil være bundet til Unreal Engine 4 som spillmotor, men det skal være anvendbart og kompatibelt med flere ulike mellomvarer. Det skal også være kompatibelt med UE4 sin egen lydmotor. De fleste mellomvarer og lydmotorer har de felles egenskapene at de kan styre ulike parametere, håndtere lydhendelser og ha noen former for DSP – effekter. Ettersom all kode i det omgivelsesbaserte lydsystemet gjøres i spillmotoren, og kun brukes for å styre lydhendelser og parameterendringer, er det derfor ikke behov for noen ekstra funksjonalitet i mellomvaren enn hva som er grunnleggende og felles for de fleste mellomvarer. Systemet kan derfor brukes med UE4 sin egen lydmotor, i tillegg til de fleste mellomvarer.

Årsaken til at denne mellomvaren har blitt brukt, er fordi det har blitt et verktøy som jeg har god kjennskap til og mestrer på et høyere nivå enn andre mellomvarer. I tillegg kan det nevnes at personlige preferanser i forhold til profileringsverktøyet, brukergrensesnitt og arbeidsflyt er grunner til at denne mellomvaren er valgt for dette prosjektet.

2.2. Immersjon

Ettersom tekniske verktøy for både spillutvikling og lydimplementasjon har blitt presentert, vil oppgaven nå ta for seg et mer teoretisk konsept som er viktig for å bedre kunne forstå vår tilnærming til spill. Dette konseptet er veldig tilstedeværende i moderne spillutvikling og er et viktig element innen spillbransjen, som fortsetter å vokse (Brooks, 2021).

En av grunnene til at denne oppgaven baserer seg på utviklingen av et lydsystem for en *tredimensjonal* verden, og en av flere årsaker til at dette lydsystemet består av de ulike konseptene og praksisene som blir presentert i kapittel 3 er på grunn av *immersjon*. Dette er et begrep som forklarer følelsen av å være til stede i en virtuell verden gjennom stimuli av alle sansene (Ermi & Mayra, 2005, s. 4). Det er strengt tatt ikke begrenset til en virtuell verden, da samme følelse kan oppnås gjennom for eksempel å lese en bok, eller høre på lydbok, men den generelle tanken er at man føler seg som en del av det fiksjonsuniverset som blir formidlet, gjennom bruk av sansene.

Dette er et tema som er mye diskutert, og mange ulike meninger og definisjoner finnes om dette begrepet. Noen vil erstatte begrepet med *romlig årvåkenhet* (spatial awareness), som defineres som å eksistere når innhold i media oppfattes som ekte og gir en følelse av å være til stedet i universet det er satt i (Madigan, 2010). Begrepet «flow» er også brukt i disse sammenhengene (Balaguer – Ballester et al., 2018, s. 4). Noen mener også at disse begrepene er ulike og at begrep som for eksempel romlig årvåkenhet kun er en del av immersjonsbegrepet (Ibid.). Ettersom disse skilnadene ikke er kritiske å etterfølge for oppgaveløsningen, vil begrepene immersjon og romlig årvåkenhet forklare samme fenomen: *Følelsen av å være til stede i en fiksjonell* (da dette ikke trenger å være virtuelt) *verden gjennom stimuli av sansene og oppfatte innholdet som ekte*.

2.3. Grunnleggende begrep

Problemstillingen tar utgangspunkt i å utvikle et lydsystem som gjør det enklere å implementere omgivelseslyder i dataspill. For oppgaveløsningen kreves det derfor en viss kjennskap til ulike begrep som omhandler ulike aspekter ved ulike felt. Disse vil være *generelle begrep innenfor spillutvikling*, *begrep direkte rettet mot lyd i spill*, og relevante *begrep innenfor akustikk og fysikk*. Disse vil bli gjennomgått slik at man bedre forstår generelle praksiser og konvensjoner i tredimensjonale spill, i tillegg til hvorfor og hvordan den praktiske delen av oppgaven har blitt løst.

2.3.1. Generelle begrep innenfor spillutvikling

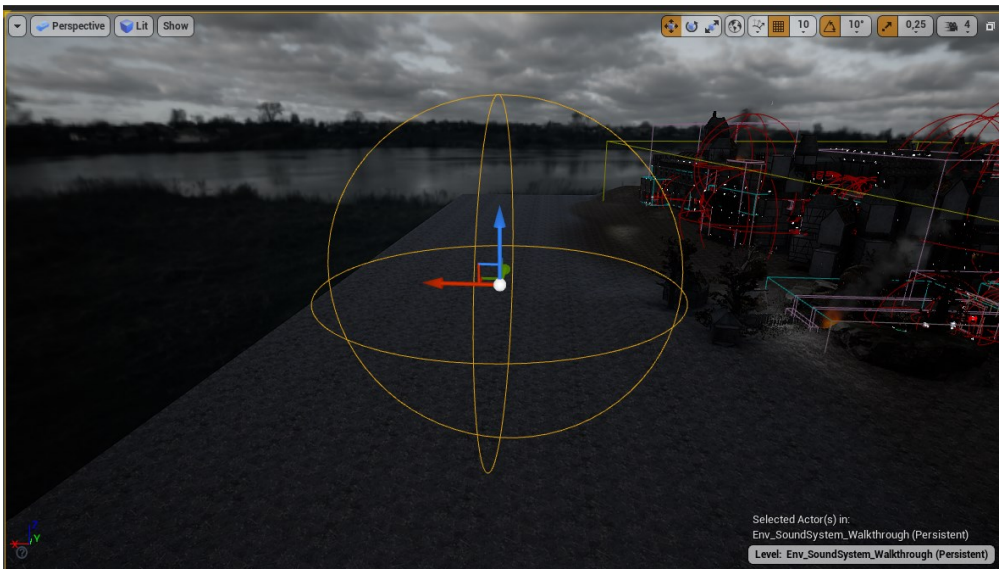
Implementering eller *implementasjon* er et ord som allerede er mye brukt i denne oppgaven, og som generelt har stor betydning for dette prosjektet. I spillsammenheng betyr dette hvordan man integrerer, plasserer eller gjør noe til en del av spillverdenen. Selv om implementasjon er et generelt begrep som benyttes for alle disipliner innen spillutvikling (grafikk og lignende), vil vi i denne oppgaven fokusere på implementasjon av lyd. I all hovedsak vil dette si hvordan vi får en lyd inn i spillverdenen og hvordan vi får den til å oppføre seg slik vi vil.

I dokumentasjonen til CryEngine, beskrives begrepet *Culling*, i tredimensjonal grafikk som å midlertidig fjerne eller avvise/frastøte alle typer objekt som ikke bidrar til det endelige bildet (CryEngine, u.å.). Det er med andre ord en metode for å kontrollere objekt eller element som ikke regnes som nødvendige for en maskin å bruke prosesseringskraft på. Et eksempel kan være hvis et objekt står så langt unna spilleren at det ikke er noen hensikt i å visuelt kunne se det, kan man midlertidig fjerne deler av objektets grafikk for å unngå unødvendig prosesseringsbruk. Culling er ikke et begrep som kun inngår i tredimensjonal grafikk. Det kan brukes om alle objekter i spill, også lydobjekt. De fleste spillmotorer og mellomvarer har ulike former for innebygde culling – system.

2.3.2. Begrep direkte rettet mot lyd i spill

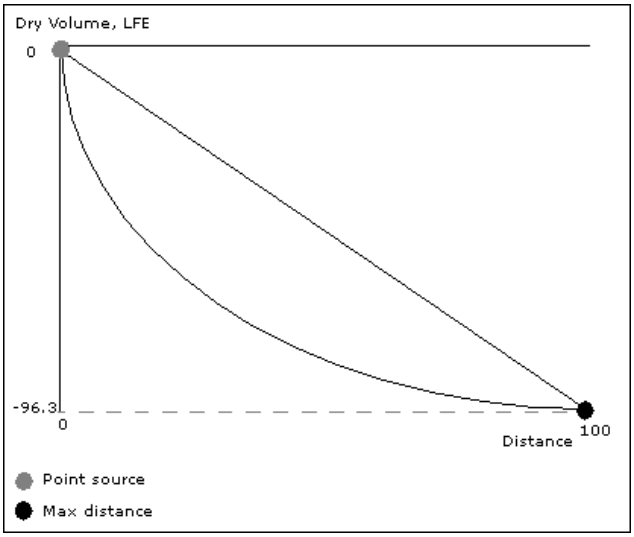
Innenfor spillutvikling vil man som regel bli kjent med noe som heter en *Audio Emitter*, i forbindelse med lyd. I korte trekk beskriver dette et objekt i spillet hvor lyden kommer fra (Jacobsen, 2018). Et annet ord for dette kan være en lydkilde. I denne oppgaven vil hovedsakelig begrepet Audio Emitter eller bare *emitter* benyttes, men bruken av ordet lydkilde vil også forekomme.

For å simulere naturlig uttoning av lyden idet spilleren flytter seg lenger unna lydkilden, brukes ofte et begrep som heter *Distanced - Based Attenuation,* eller *Attenuasjon* (Audiokinetic, u.å.a). Her vil posisjonen til spilleren i forhold til lydkilden påvirke lydintensiteten til lydkilden. Dette er som regel definert av en radius, som man selv kan bestemme. Jo nærmere senter av radiusen man kommer (nærmere lydkilden), jo mer øker intensiteten, og motsatt jo lenger unna man befinner seg. Radiusen markerer også punktet hvor lyden går fra å være hørbar til ikke - hørbar.



Figur 7: Eget utsnitt av eksempel på attenuasjonsradius til en audio emitter. Den hvite sfæren i midten representerer senterpunktet til radiusen hvor lyden er på sitt høyeste, volummessig. Kantene til radiusen viser hvor lyden er på sitt laveste.

Hvordan denne attenuasjonsradiusen faktisk påvirker lyden er definert av en serie kurver som styrer ulike verdier for lydparameter som for eksempel volum eller lavpass filter. I Wwise, kan lyddesigneren eksempelvis bestemme hvilke lydparameter som skal påvirkes og alle kurvene til de ulike lydkildene.

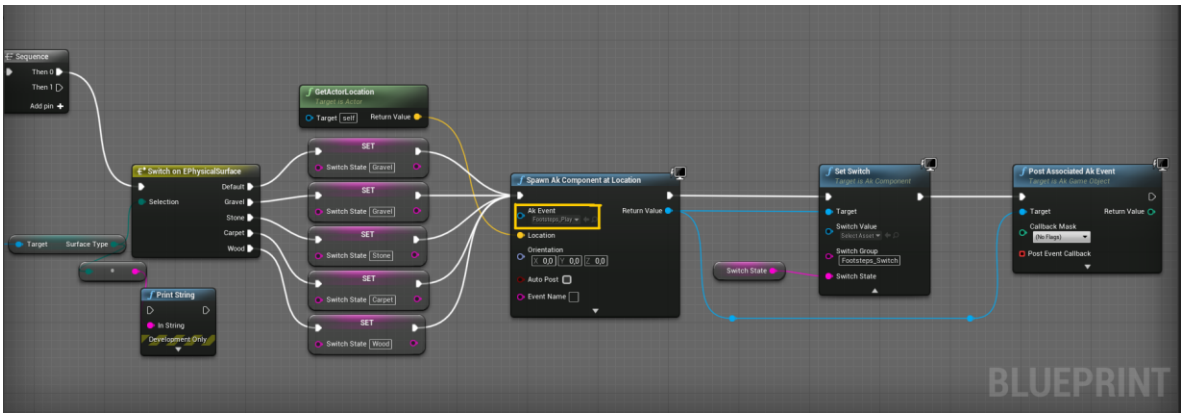


Figur 8: Kurve som viser hvordan volum kan påvirkes av spillerens distanse til objektet. Grå sirkel er lydkildens plassering, og svart sirkel er likt endepunkt på attenuasjonskurven (Hentet mars 17, 2022, fra https://audiokinetic.com/library/edge/?source=Help&id=applying_distance_based_attenuation).

En lydkilde trenger ikke å ha en sfærisk form for attenuasjon, som vist i *figur 7*. Den kan også være formet som en kjegle. Dette er avhengig av hvilken oppfattet *direksjonalitet* eller *direktivitet* lyden har. I all hovedsak går dette ut på hvor den oppfattede posisjonen til lyden er i forhold til lytteren. Hvis man står i et rom og musikk spilles fra en høyttaler, vil man ofte høre nøyaktig hvor lyden kommer fra. Dette vil være et resultat av flere fenomen. Et av dem omhandler lokalisering av lyd. Dette går blant annet ut på hvordan vi bruker tidsforsinkelse og nivåforskjell mellom ørene for lokalisering av lydkilder (Ohmori & Yamada, 2010, s. 513). I det nevnte eksempelet kan enda en påvirkning av direktiviteten til lyden være et resultat av høyttalerens oppbygning, da lavere frekvenser som regel er mer omnidireksjonelle (ikke – direktive), enn hvis driveren er lik eller større enn lydens bølgelengde (P. Svensson, forelesningsnotat, TT3010, 2018).

I spillverdenen er direksjonalitet ofte direkte forbundet med tredimensjonale lyder, som da er en lyd med en virtuell plassering i en tredimensjonal verden, og som oppleves som direksjonell. For dette formålet pleier lydformatet å være *mono*. Todimensjonale lyder i denne sammenhengen er det motsatte. Dette vil være lyder som ikke oppfattes som om de kommer fra en spesiell plass i spillverden. Lydformatet for disse lydene vil være *stereo*.

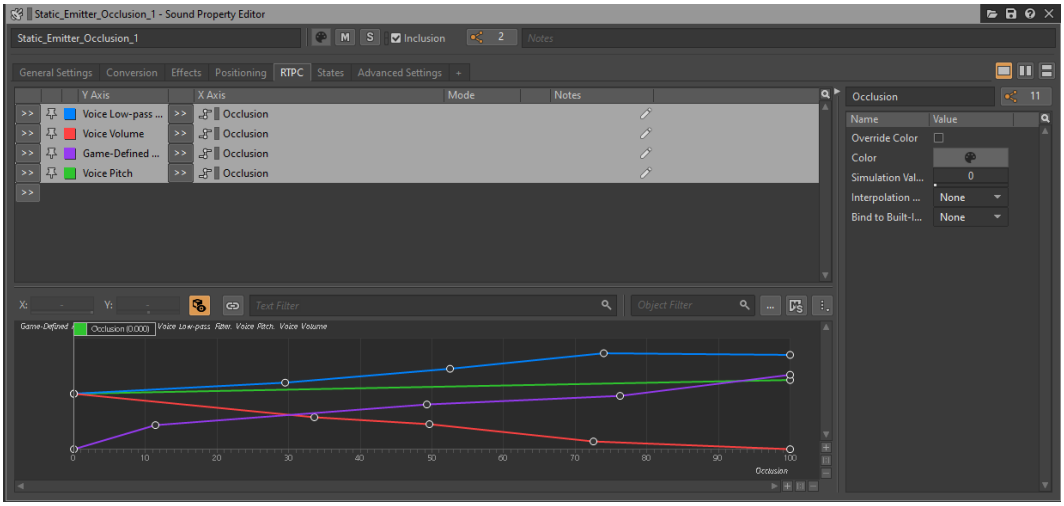
Events, er et begrep i forbindelse med mellomvarer og lydmotorer som kan brukes for å styre lyden i spillet. En event kan tilføre ulike handlinger til lydobjekt, som å spille av en lyd, stoppe en lyd eller aktivere parameterendringer (Audiokinetic, u.å.e). Et annet ord for event kan være *lydhendelse,* som er et begrep brukt tidligere i oppgaven. Etter at en event er laget i for eksempel Wwise, kan den brukes i spillet ved å bli «kalt på» av spillkoden når den trengs. Hvis vi ser på fotsteg - koden for bevegelsessystemet i *figur 9*, vil det hver gang spilleren beveger seg bli sendt en beskjed til Wwise som sier at det skal aktiveres en event som heter «Footsteps_Play». I Wwise har vi i dette eksemplet bestemt at denne eventen skal spille av en fotsteg – lyd når den aktiveres.



Figur 9: Eget eksempel på kode som sender beskjed til Wwise om å aktivere en event.

En *RTPC,* forkortet for *Real – Time Parameter Control,* gjør akkurat det navnet tilsier. Den kan modifisere og endre egenskaper til et lydobjekt dynamisk, basert på informasjon fra spillet i sanntid (Audiokinetic, u.å.c). I Wwise kan man for eksempel lage en RTPC med en kurve som øker mengden romklang på et lydobjekt jo lenger unna objektet spilleren befinner seg. I spillmotoren

kan man da kontinuerlig beregne avstanden fra spilleren til lydobjektet, og sende en beskjed til Wwise om at den må påføre en viss mengde romklang basert på den avstanden som er beregnet.



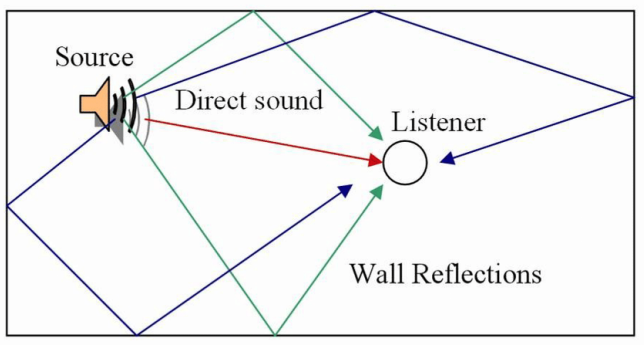
Figur 10: Eget eksempel på hvordan RTPC kurvene i Wwise kan se ut. Hver kurve representerer en parameter med lik fargekode. Rød kurve tilsvarer endring for volumparameter som også er markert rødt. Endringen langs kurven styres av en RTPC som heter Occlusion.

2.3.3. Begrep innenfor akustikk og fysikk

Lydutbredelse eller *lydforplantning* er begrep innenfor akustikk og fysikk som beskriver hvordan lydbølger beveger seg gjennom ulike medier fra en kilde til en lytter (Elnoshokaty, u.å., s. 4). Gjennom denne utbredelsen kan bølger bli *reflektert*, bøyd (*diffraksjon*) eller overført (*transmission) (*Henderson, u.å.). Uten å gå inn i alt for store detaljer rundt det fysiske aspektet ved disse akustiske fenomenene, skal det i dette underkapittelet forsøkes å gis en kort av forklaring av hva disse begrepene innebærer. Det kan være verdt å merke seg at det i mange tilfeller også er snakk om et begrep som heter *refraksjon,* innenfor lydutbredelse (Ibid.). Dette begrepet er ikke relevant for oppgaven, og vil derfor ikke inkluderes i begrepsforklaringene i dette underkapittelet.

Refleksjoner i denne sammenhengen er når lydbølger ikke blir absorbert eller overført idet de treffer et nytt medium (ofte en overflate), men endrer retning i krysningspunktet mellom to medier slik at de returnerer til mediet det originalt oppstod i (Ibid.). Når en lyd oppstår, vil den først treffe

lytteren direkte (direktelyd), og kort tid etterpå vil den samme lyden nå lytteren fra forskjellige reflekterende overflater, så lenge det ikke er snakk om et fritt felt (oppstår når det ikke er noe som har reflekterende egenskaper) (Rossings, 2002, s. 527). Romklang, er et eksempel på fenomen bestående av flere refleksjoner.



Figur 11: Representasjon av refleksjoner (Hentet mars 19, 2022 fra https://www.researchgate.net/publication/265928273_Auditory_Room_Size_Perception_for_Real_Rooms/figures?lo=1).

Et begrep rundt refleksjoner som er viktig å forstå er *etterklangstid (RT)*. Dette er definert som tiden det tar for lyden å senkes med 60 dB (Ibid. s. 530). Denne tiden vil også være ulik for forskjellige frekvensbånd. Etterklangstiden kan derfor være ulik for lave og høye frekvenser, for ulike materialer (Ibid. s. 531). Hvilke etterklangstider en lyd får er blant annet basert på rommets størrelse og den totale absorpsjonen (Ibid.). Dette kan regnes ut gjennom *Sabines ligning:*

$$RT = 0.161 * \frac{V}{A}$$

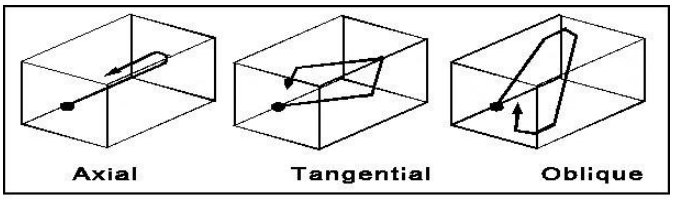
Tallet *0.161* er en konstant, *V* er rommets volum og *A* er det totale absorpsjonsarealet i rommet (Ibid. s. 532). Denne versjonen av ligningen tar ikke høyde for luftabsorpsjon.

Et annet konsept er *Rom Moder.* Disse er naturlige resonanser i rommet som et resultat av rommets geometri (Ibid. s. 556). Dette gjelder hovedsakelig for lavere frekvenser (under 200 – 300 Hz), og består av *Axiale, Tangentiale* og *Oblique* moder (Ibid.). De resonnerer langs dimensjonene som omhandler lengde, bredde og høyde (Ibid.). Dette gjelder hvilke overflater refleksjonene treffer,

men for vår del er det viktigst å vite om Axiale moder ettersom de lagrer mer akustisk energi fordi lydbølgene reiser lengre mellom refleksjoner (Ibid.). Disse modene innebærer refleksjoner mellom to parallelle overflater, og er illustrert i *figur 12*. Den opplevde effekten av rom moder kan være en økning av lydstyrken for spesifikke frekvenser og forvrengning av lyden. Et eksempel på dette kan være at når man er nærme en vegg, vil man høre en oppfattet økning av lydens lavere frekvenser. Rom moder finner man ved formelen:

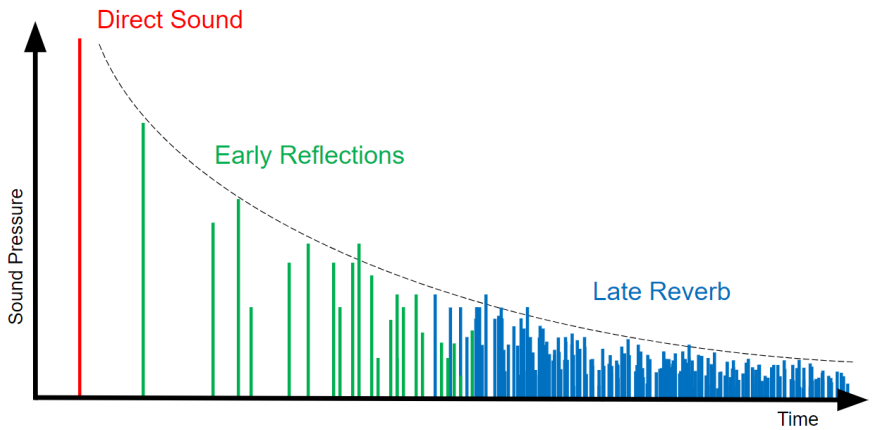
$$f_{lmn} = \frac{c}{2}\sqrt{\left(\frac{l}{l}\right)^2 + \left(\frac{m}{w}\right)^2 + \left(\frac{n}{h}\right)^2}$$

Konstanten *c* er lydens hastighet, *l, m, n* er heltall og *l, w, h* er lengde, bredde og høyde til rommet (Ibid.).



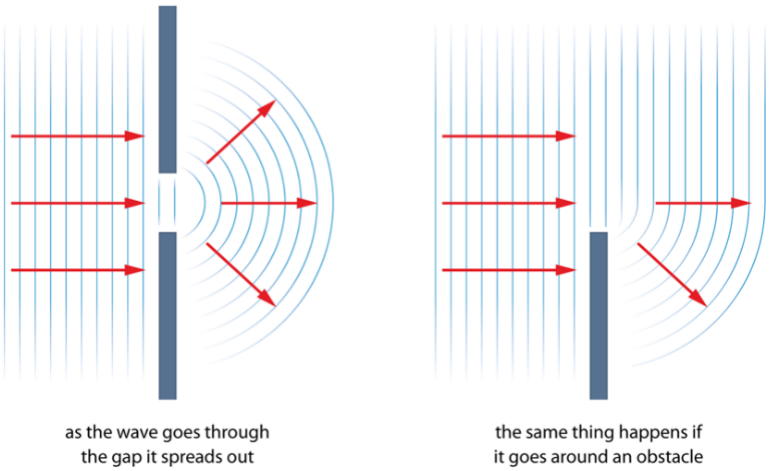
Figur 12: Illustrasjon av de ulike modene (Hentet mars 20, 2022 fra https://www.gikacoustics.com/what-are-room-modes/).

Enda et begrep som har tilknytning til romklang og refleksjoner er begrepet *Pre – Delay.* Begrepet i seg selv brukes mest i forbindelse med digitale romklangeffekter, men beskriver likevel et fenomen innen akustikk. Pre - Delay representerer tiden det tar for lydens tidlige refleksjoner å nå lytteren etter at direktelyden har nådd lytteren (Sinclair, 2020, s. 105). Dette er illustrert i *figur 13.* De tidlige refleksjonene er gode indikatorer for romstørrelse (Ibid, s. 219).



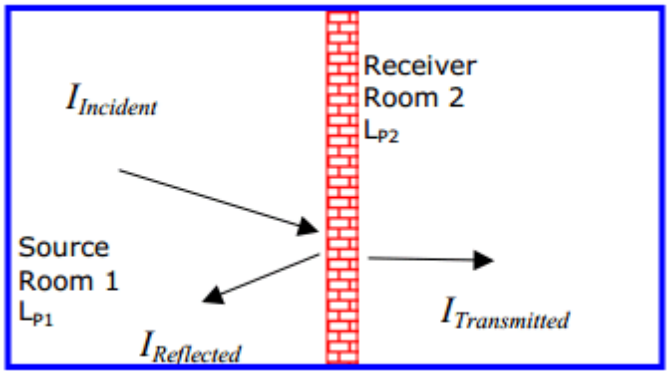
Figur 13: Illustrasjon av tidsforsinkelse mellom direktelyd, tidlige og sene refleksjoner (Hentet mars 20, 2022 fra https://blog.audiokinetic.com/image-source-approach-to-dynamic-early-reflections/).

Diffraksjon, er spredningen av bølger i det de passerer en åpning, eller bøyningen av bølger rundt små objekt (Buffoni, 2020). Bøyningen av bølger rundt et objekt skjer når bølgelengden er mindre eller tilnærmet lik bølgelengden til den aktuelle lyden. Høye frekvenser vil som regel ikke bøye seg rundt objekt, men bli absorbert, ettersom de har korte bølgelengder. Lavere frekvenser har derimot mye lengere bølgelengde, og passerer derfor i større grad uforstyrret rundt et objekt (Lakka et al., 2018, s. 34). Når bølgelengden er tilnærmet lik dimensjonene til et objekt, vil bølgen bruke kantene til objektet som et slags senterpunkt, hvor det skapes en ny bølgefront med samme frekvens, men med lavere intensitet.



Figur 14: Visuell representasjon av diffraksjon. (Hentet mars 20, 2022 fra https://roguephysicist.com/KS4/core/waves/wave_behaviour/diffraction.png).

Transmission, eller *acoustic transmission*, er et begrep som beskriver overføringen av lyd mellom ulike materialer. Når en lyd treffer skillet mellom to materialer vil en del av lyden bli absorbert, en annen del reflektert tilbake i rommet og noe overført til det nærliggende rommet gjennom veggen (Cheng et al., 2017, s. 33). Et eksempel på dette kan være er hvis man står i et avlukket rom og hører lyder som kommer fra et nærliggende rom eller naborom. Dette kan også omtales som *akustisk overføring*.



Figur 15: Representasjon av akustisk overføring (Hentet mars 21, 2022 fra https://www.researchgate.net/profile/Wei-Tan-13/publication/316579745/figure/fig1/AS:494485098696704@1494905842110/Basic-concept-of-sound-transmission-1.png).

3. Generelle lydpraksiser og lydkonvensjoner i 3D spill

I denne delen av oppgaven skal det bli gjort rede for generelle lydpraksiser og lydkonvensjoner i tredimensjonale dataspill. Dette er først og fremst konvensjoner og praksiser relatert til omgivelseslyder og deres karakteristikk (hvordan de høres ut) som et resultat av spillverdenens oppbygning (spillgeometri) og utseende. Det finnes riktignok flere lydkonvensjoner og lydpraksiser i tredimensjonale spill enn hva som nevnes i dette kapittelet, men ettersom de ikke har direkte forbindelse med det praktiske arbeidet som er gjort, vil de ikke redegjøres for i oppgaven.

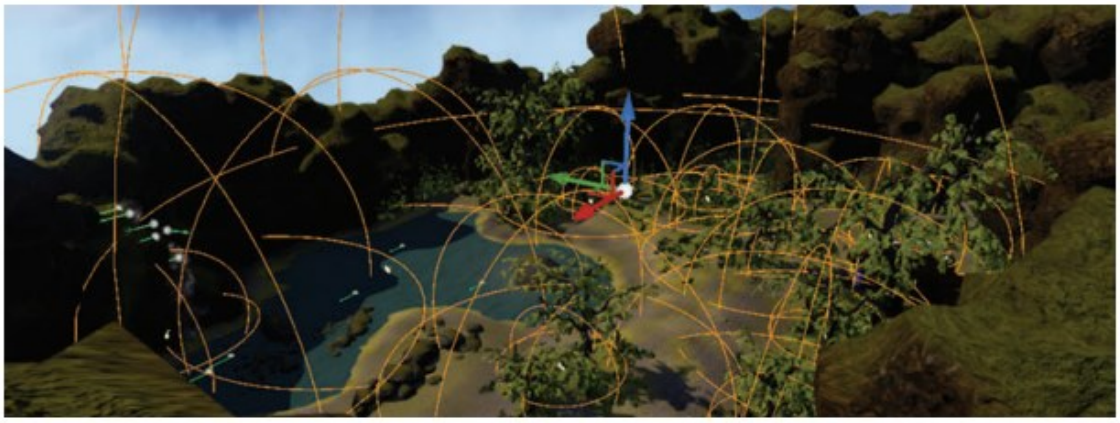
3.1. Tredimensjonale audio emittere

I tredimensjonale spill er det ofte vanlig at lyder i spillverdenen oppleves som om de kommer fra spesifikke lydkilder. Det trenger ikke alltid være slik, men i de fleste tilfeller vil dette være naturlig ettersom det representerer en mer nøyaktig tilnærming til virkeligheten. Disse kalles gjerne for «source loops», da de ofte består av en gjentagende lydsekvens som kommer fra en spesifikk plass i spillverdenen (Raybould & Stevens, 2015, s. 31). Disse gjentagende sekvensene trenger ikke å være sømløse og uavbrutte. De kan også være separert av pauser, for eksempel etter at lydsekvensen er spilt ferdig. Slike tredimensjonale lydkilder er med på å bygge opplevelsen av en levende lydverden og atmosfære. Dette bidrar til å øke den immersive opplevelsen.

I forbindelse med implementasjon av tredimensjonale lydkilder, finnes det noen konvensjoner som skal presenteres i dette underkapittelet. Måten en lydkilde som regel lages på, er ved å plassere en *audio emitter* ut i spillverdenen. Denne representerer lydkilden og har en bestemt *attenuasjonskurve* for å kontrollere hvordan lyden påvirkes som et resultat av distansen mellom kilden og lytteren. Denne prosessen blir beskrevet i detalj av Raybould og Stevens i deres bok *Game Audio Implementation - A Practical Guide Using The Unreal Engine.* Dette blir også understreket i dokumentasjonen til UE4 og Audiokinetic Wwise, hvor selve audio emitteren omtales som enten en *Ambient Sound Actor* (Epic Games, u.å.a) eller en *AkAmbient Sound Actor* (Audiokinetic, u.å.f). I all hovedsak blir denne audio emitteren knyttet opp mot en event som starter avspillingen av en bestemt *source loop*. Audio emittere kan være både statiske og dynamiske, men

i forbindelse med omgivelseslyder er det statiske audio emittere som er vanligst å bruke. Slike audio emittere blir som regel aktivert når spillet eller den aktuelle banen starter, og spilles av kontinuerlig.

I en spillverden er det ikke uvanlig at en bane består av store mengder audio emittere. Dette trengs ofte for å skape variasjon, og nok lydkilder til å danne et helhetlig og levende lydbilde for det universet hvor spillets handling utspiller seg i.



Figur 16: Eksempel på mengder audio emittere som kan plasseres ut på et lite område (Hentet mars 24, 2022 fra Raybould & Stevens, 2015, s. 33).

Når mengden audio emittere øker, vil maskinen bruke mer prosesseringskraft på å håndtere dem. Dersom man bruker Wwise kan man blant annet oppleve noe som kalles for *voice starvation* (Audiokinetic, u.å.g). Dette betyr i all praksis at lyd enten ikke spilles av, eller at den hakker. Hva det heter og hvordan det faktisk oppleves varierer ut ifra hvilken mellomvare og spillmotor man bruker, men prinsippet er det samme; prosesseringsbruken blir for stor, og maskinen klarer ikke å håndtere det slik den skal (ettersom den har mange andre prosesser som også skal prioriteres). Det er flere måter man kan forhindre dette på. Som nevnt i kapittel 2.3.1, er culling en metode som kan benyttes for å unngå unødvendig ressursbruk. Dette kan for eksempel gjøres direkte i Wwise ved å kontrollere hvor mange instanser av en lyd som kan spilles av samtidig, eller ved å endre

oppløsningen til de aktuelle lydfilene (Ibid.). Både Wwise og UE4 har sine egne metoder for å håndtere dette.

3.2. Atmosfæresystem

Et av de vanligste lydsystemene man implementerer i et spill er atmosfæresystem (ambience – system). Et vanlig konsept her er å dele spillet inn i ulike lydregioner. Disse regionene representerer områder hvor man forventer å høre forskjellige atmosfærelyder, ofte med egenskaper og karakteristikker som er spesifikke for det området (Dirrenberger, 2018, s. 183). Bruken av tredimensjonale lydkilder (som nevnt i kapittel 3.1) i slike regioner er en måte å bygge atmosfære på i spill. Dette kalles også for *tredimensjonal atmosfære* eller *3D – Ambience.*

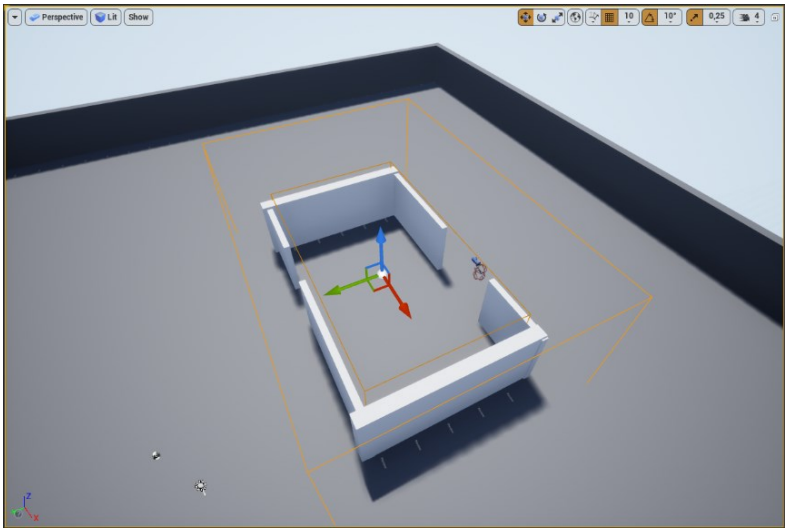
3.2.1. Todimensjonale atmosfæresystem

I nesten alle områder, vil man fra virkeligheten forvente å høre en eller annen form for lyd. Til og med i et tomt rom kan man oppleve å høre lyder; såkalt *rom – lyd* (Raybould & Stevens, 2015, s. 26). Dette er en type lyd som er karakteristisk for det rommet eller området man er i. Eksempler på dette kan være ventilasjonsbråk, svak vindsus, eller elektrisk støy. Disse lydene er ofte veldig subtile og har en dronelignende karakteristikk. Rom – lydene vil vanligvis ikke være direksjonelle, og vil bli mer diffuse jo lenger unna rommet man er (Ibid.). På mange måter kan man se på disse lydene som omringende for lytteren og virke som om de ikke har en åpenbar lydkilde. For at lyden skal kunne bli mer diffus jo lenger unna man er, brukes som regel en attenuasjonskurve med senterpunkt for hele området, og ytterpunkt for radiusen litt utenfor. Ofte vil lydene som brukes for et slikt system være todimensjonale (stereo) eller surroundbaserte. I denne oppgaven vil det kun brukes stereo i denne sammenhengen.

Måten slike områder som regel defineres på er ved å plassere bokser av ulike former i spillverdenen (formen på boksen er som regel lik formen til området eller regionen). Disse representerer området for en spesifikk todimensjonal atmosfærelyd eller romlyd. Dirrenberger beskriver dette som et

Sound Shape Tool, eller lydutformingsverktøy (Dirrenberger 2018, s. 186). Denne boksen brukes da til å aktivere brukerdefinerte atmosfærelyder når spilleren befinner seg innenfor denne boksen.

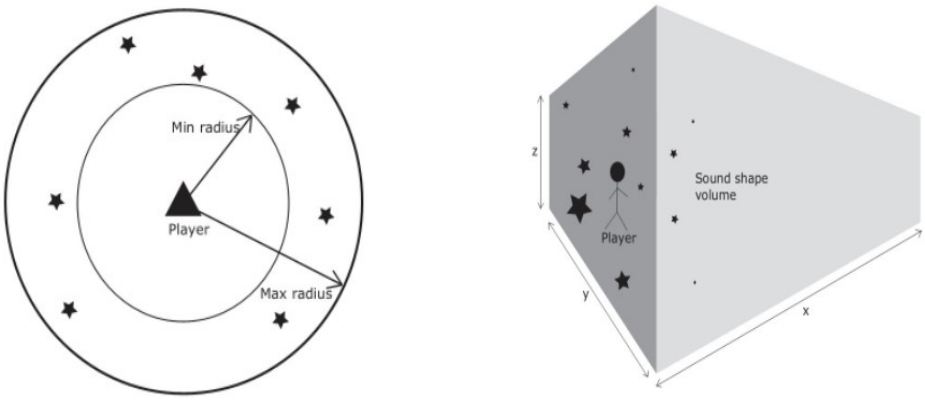
En annen måte å bruke disse boksene på er uten noen form for attenuasjon, men definere et tidsintervall på hvor lang inn og uttoning lydene skal ha når de starter og stopper. Disse kan være enda enklere å sette opp, men *fører til at man mister avstandsbasert inn – og uttoning for tidsbasert inn – og uttoning*. Et eksempel på når dette kan være ugunstig er hvis spilleren befinner seg i et inngangsparti og går ut hoveddøra (og da ut av området med den spesifikke rom - lyden for inngangspartiet). Lyden fra inngangspartiet vil da stoppes med en uttoning som varer et visst antall sekunder. Dersom spilleren kjapt beveger seg ut døra og videre, kan denne uttoningen virke naturlig i henhold til den oppfattede avstanden mellom spilleren og døråpningen, ettersom avstanden øker idet lyden avtar. Dersom spilleren derimot går ut døra og stopper rett ved døråpningen, vil lyden avta selv om spilleren står i ro og befinner seg rett ved siden av rommet hvor lyden kommer fra. I denne situasjonen ville man forventet å fortsatt høre lyden fra inngangspartiet, bare svakere. Dette skaper en mer urealistisk opplevelse, og kan i verste fall føre til at immersjonen blir brutt.



Figur 17: Eget eksempel på oppsett av et Sound Shape Tool for 2D – atmosfærelyder med attenuasjonskurve for inn – og uttoning av romlyd.

3.2.2. Tredimensjonale atmosfæresystem

Det er også vanlig å bruke alternative måter på å skape tredimensjonale atmosfærelyder, enn å plassere mange audio emittere rundt om i en spillverden. Dirrenberger kaller denne metoden for *Random FX* (Ibid. S. 187). Dette brukes for å forbedre det immersive aspektet ved å spille av lyder i et gitt område, i en gitt radius rundt spilleren og med en viss tilfeldighetsfrekvens (Ibid.). Slike lyder kan for eksempel være vindkast, insektlyder eller fuglekvitring. Disse blir spilt av i et definert område rundt spilleren. I kapittel 4.2.2 skal vi se på en versjon av denne måten å lage tredimensjonale atmosfærelyder på.



Figur 18: Figur til venstre viser lydplassering med en min/maks radius. Figur til høyre viser tilfeldige lyder spilt av innenfor et gitt område (Hentet mars 28, 2022 fra Dirrenberger, 2018, s. 188).

3.3. Lydutbredelsessystem

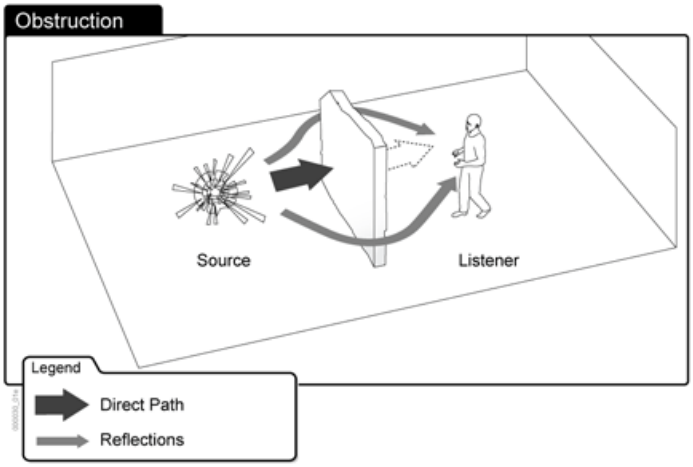
I kapittel 2.3.3 ble det nevnt et begrep som omhandler lydens forplantning i rom eller omgivelser, såkalt *lydutbredelse*. I spill blir måten lyd beveger seg gjennom ulike medier (i ulike omgivelser) på, ofte simulert gjennom metoder som gir en tilnærmet lik opplevelse som det virkelige fenomenet. Det brukes ofte ikke løsninger som helt eksakt beregner hvordan lyd sprer seg i rom. En av årsakene til dette er at verdensdimensjonene i spill kan være urealistiske. Hovedårsaken er derimot at det kreves enorme ressurser for å beregne nøyaktig lydutbredelse i sanntid (Raybould & Stevens, 2015, s. 227). Tidligere har det blitt redegjort for noen begrep innenfor akustikken som

inngår i lydutbredelsesaspektet. I de neste avsnittene skal vi se på hvordan disse kan bli overført til en spillverden og vanlige praksiser innenfor utviklingen av et lydutbredelsessystem.

I terminologien for spill-lyd vil man ofte møte på begrepene *obstruksjon* (obstruction) og *okklusjon* (occlusion). Disse begrepene kan i enkelte tilfeller bli definert på ulike måter, men den generelle forståelsen av konseptene rundt dem er like; Obstruksjon skjer når *direktelyden fra en lydkilde blokkeres av en gjenstand*, men *refleksjonene fra denne lyden finner en vei rundt gjenstanden og treffer lytteren* ettersom lytteren og lydkilden befinner seg i samme miljø eller omgivelse (Harris, 2018, s. 300). Okklusjon skjer derimot når lytteren og lydkilden befinner seg i ulike omgivelser (for eksempel separate rom), som fører til at *både direktelyden og refleksjonene blir blokkert* (Ibid.). Hvordan man skal sette disse begrepene opp mot akustiske fenomen er ikke alltid like lett. En generell beskrivelse fra Nathan Harris (utvikler i Audiokinetic) er likevel at obstruksjon kan beskrives ved hjelp av *diffraksjon,* da det beskriver hvordan lyd filtreres og dempes når det bøyes rundt et objekt (Ibid.). Okklusjon beskrives ofte gjennom fenomenet *akustisk overføring*, hvor lyd går gjennom objekter og overføres til et annet materiale (Ibid.).

3.3.1. Obstruksjon

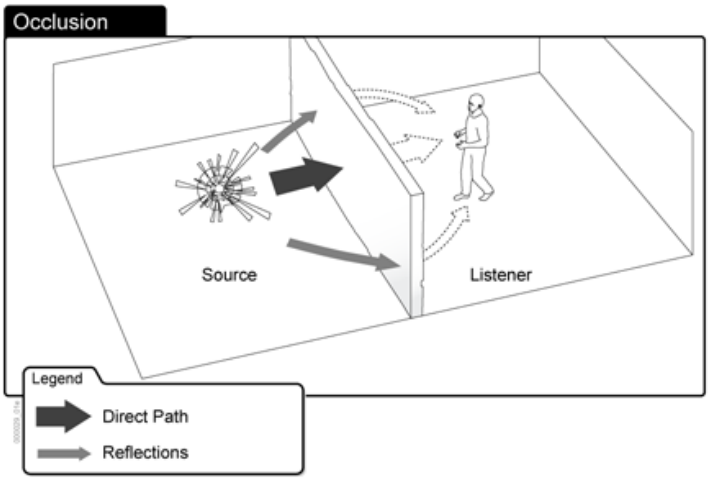
Hvis et objekt blir plassert mellom en lydkilde og en lytter, vil det opplevde fenomenet i de fleste tilfeller være en filtrering av høye frekvenser og demping av direktelyden. For å simulere den *lydlige* eller *hørbare effekten*, gjøres det ofte parameterendringer i form av frekvensfiltrering og volumdemping i lydmotoren, enten om det er en mellomvare, eller den innebygde lydmotoren. Hvordan dette *beregnes* varierer derimot i stor grad, og noen avanserte og gode løsninger blir blant annet presentert i Guy Somberg sin bok *Game Audio Programming 2 – Principles and Practices*. En av de vanligste metodene for å beregne obstruksjon på i spill, er ved bruk av såkalt *Raycasting.* Dette innebærer at man sender en usynlig virtuell stråle i direkte retning fra spilleren til lydkilden (eller omvendt). Det blir da registrert om noen spillobjekter er i veien for eller blokker denne strålen. Hvis noe blokkerer den, vil høyfrekvensene til direktelyden fra kilden bli filtrert, og volumet dempet. (Raybould & Stevens, 2015, s. 250).



Figur 19: Visuell representasjon av obstruksjon (Hentet april 5, 2022 fra https://www.audiokinetic.com/library/edge/?source=SDK&id=soundengine_obsocc.html).

3.3.2. Okklusjon

Før de vanligste beregningsmetodene for okklusjon blir presentert, vil den hørbare og oppfattede påvirkningen dette konseptet har på lyden bli forklart. Når okklusjon oppstår, vil man oppleve en filtrering av høyfrekvenser og demping av lyden, men i motsetning til obstruksjon vil dette påvirke både direktelyden og refleksjonene, da lydkilden og lytteren befinner seg i ulike miljø eller omgivelser.



Figur 20: Visuell representasjon av okklusjon (Hentet april 5, 2022 fra https://www.audiokinetic.com/library/2018.1.11_6987/?source=Help&id=understanding_okklusjon).

En av de vanligste måtene å beregne okklusjon på er (i likhet med obstruksjon) ved bruk av raycasting. Gjennom denne måten å gjøre det på vil man få enkle resultat på når en lydkilde blokkeres av en gjenstand (Ibid. s. 246). Et av problemene med enkel raycasting er at det er vanskelig å skille mellom *blokkering fra gjenstander som er i samme omgivelse som spilleren* og *blokkering fra gjenstander som befinner seg i andre omgivelser enn der hvor spilleren er*. Dette er fordi at strålen kun registrerer det første objektet som har en blokkerende egenskap. På grunn av dette kan ikke denne metoden brukes effektivt og realistisk nok for beregning av okklusjon. I et scenario hvor man står i et rom og et objekt i rommet står mellom lytteren og en lydkilde, kan enkel raycasting benyttes uten problem. Hvis vi derimot bruker samme metode på en lydkilde som er tre rom vekk fra der hvor lytteren befinner seg, vil ikke denne metoden registrere at lyden må gjennom tre ulike vegger av ulik størrelse og materiale, men kun registrere den første veggen. Dette vil føre til et urealistisk resultat. En mulig løsning for dette kan være å bruke en såkalt *Multi Raycast,* eller *Multi Line Trace* (UE4), som registrer alle treff langs strålen som blir sendt ut fra spilleren (Epic Games, u.å.c). Dette er en vanlig løsning for problematikken presentert i det sistnevnte eksempelet.

Ulike varianter av raycasting utgjør de vanligste måtene å beregne okklusjon på. Disse innehar likevel en felles problematikk når det gjelder bruken av dem. Denne problematikken går ut på at å *bruke raycasting for å beregne okklusjon er en krevende prosess for maskinen*. Dette gjelder spesielt metoden som klarer å registrere flere treff (Multi Raycast). Disse treffene må registreres for hver lydkilde som finnes i en bane eller innenfor et gitt område. Det må også skje ofte og som regel over store avstander. Dette kan føre til stor påkjenning på maskinen og påvirke ytelsen til spillet. I tillegg vil man i større spillselskap ha et CPU -og minnebudsjett, og for lyd er det som regel ikke veldig stort. Det er derfor viktig å finne løsninger som ikke er for ressurskrevende. På internett finnes det flere eksempler på hvordan man kan lage et okklusjonssystem, men flesteparten av dem bruker raycasting som løsning i eller annen form.

3.3.3. Romklang

En annen praksis i forbindelse med lydutbredelse i spill, er hvordan man simuler refleksjoner eller romklang i et gitt miljø eller sett med omgivelser. I kapittel 3.2 ble det nevnt at man ofte deler inn spillverdenen i ulike regioner eller omgivelser for lyd, ofte i form av bokser med forskjellige former for å passe det angitte området (Raybould & Stevens, 2015, s. 227). I denne prosessen blir det ofte tildelt en type romklang til disse områdene. Denne romklangen har en spesifikk etterklangstid, filterkurve, tidlige refleksjoner og flere ulike parametere som skal simulere hvordan refleksjonene oppfører seg i det aktuelle området (Boev, 2019). Disse skapes ofte gjennom delaybaserte eller konvolusjonsbaserte digitale romklangeffekter.

Verdiene til romklangen blir ofte lagt inn i mellomvaren, eller lydmotoren. Hvordan man håndterer oppførselen til disse refleksjonene varierer, og det finnes flere løsninger på å håndtere utbredelsen til refleksjonene i ulike rom. Noen lager for eksempel såkalte *dynamiske* romklang – system. Dette vil si at det kontinuerlig utføres beregninger i forhold til rommets størrelse, materialer og tidlige refleksjoner basert på spillgeometrien slik at romklangen endres hele tiden uten at man trenger å definere spesifikke områder for den. Selv om dette ofte kan føre til mer realistiske resultater, vil det kreve mye mer ressurser. For å forhindre dette, kan det derfor være mer hensiktsmessig å bruke regioner og bokser.

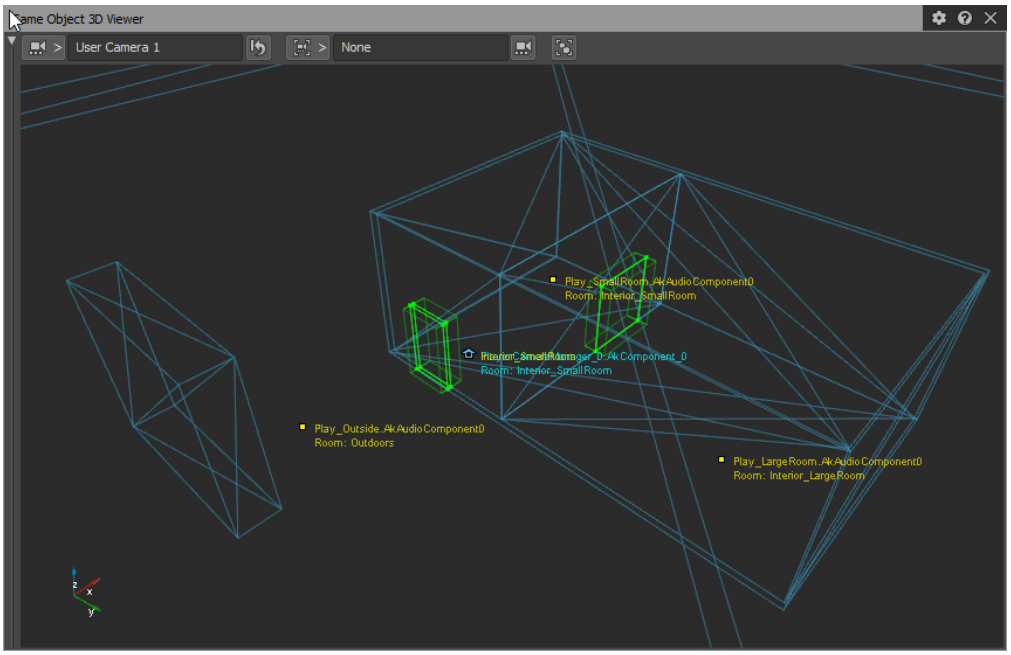
3.3.4. Utbredelsesveier

Et annet konsept ved lydutbredelse er å finne ut av hvor lyden faktisk skal bevege og spre seg. Dette er et større konsept som består av både okklusjon, obstruksjon og flere av de akustiske begrepene nevnt tidligere i denne oppgaven. Utbredelsesveier vil også være et resultat av spillgeometrien og hvordan spillverdenen er bygd opp. En spillmotor vet i utgangspunktet ikke hvordan lyd skal reagere på geometrien til et spillobjekt (Raybould & Stevens, 2015, s 250). Det er med andre ord ikke en innebygd funksjon for håndtering av *lydgeometri,* som det for eksempel er for lys. Hvis prosesseringskraft ikke er et problem, kunne man brukt samme geometri som for lys, men dette blir ofte for ressurskrevende (Filion, 2018, s. 283). På grunn av dette ender man som regel opp med å lage en form for enklere geometri kun for lyd. Dette er ofte spillobjekt med ulike

former som representerer formen til spesifikke omgivelser, hvor lyden har en spesiell egenskap i forhold til utbredelse (obstruksjon, okklusjon og romklang).

Det finnes allerede noen tilgjengelige verktøy som håndterer lydutbredelse veldig godt, og noen av dem har fungert som grunnlag for løsninger gjort i det omgivelsesbaserte lydsystemet. Løsninger for lydutbredelse finner man ofte i form av egne programvarer og separate plugins eller utvidelser for mellomvarer. Eksempler på slike programvarer og utvidelser kan være *Resonance Audio* (Resonance Audio, u.å.), *Steam Audio* (Steam, u.å.) og *Wwise Spatial Audio* (Audiokinetic, u.å.). Selv om disse programmene er kompatible med flere spillmotorer, kommer de ofte med begrensninger. De fleste av de nevnte programvarene og utvidelsene er for eksempel *kun kompatible med sine egne løsninger*, som vil si at man som regel ikke kan kombinere dem med hverandre. En annen begrensning er at noen av dem *koster ekstra å bruke*. Skal man for eksempel lansere et spill med *Wwise Reflect* (se kapittel 2.1.2.2), koster det (mer) penger. En tredje begrensning er at ettersom de er eller er en del av tredjeparts programvarer, er det ofte *vanskeligere å feilsøke, modifisere og tilrettelegge dem til egne prosjekter*. På grunn av dette velger mange å lage egne løsninger for lydutbredelse.

Et av de verktøyene som har fungert som inspirasjon for utviklingen av undersystemet for lydutbredelse i det omgivelsesbaserte lydsystemet er Wwise Spatial Audio. Det viktigste konseptet fra dette verktøyet er håndteringen av geometri for lydutbredelse, gjennom det de kaller *Rooms & Portals.* Rooms og Portals gir oss muligheten til å lage geometriske objekt som effektivt kan modellere lydutbredelse for ulike lydkilder (Audiokinetic, u.å.d). I denne sammenhengen er begrepet Room det samme som en region eller område (kan være både inne og ute). Rooms blir koblet sammen med hverandre gjennom bruken av Portals. Disse fungerer sammen som et nettverk av flere rom, hvor lyd kan bevege og spre seg. Rooms og Portals sørger da for at obstruksjon, okklusjon og refleksjoner (romklang) og generelle lydutbredelsesveier blir beregnet.



Figur 21: Figur som viser hvordan rom og portaler fra Wwise kan se ut. De store blå boksene representerer rom, mens de gule boksene representerer portalene som kobler sammen rommene (Hentet april 6, 2022 fra https://www.audiokinetic.com/en/library/2019.2.14_7616/?source=UE4&id=usingwwisespatialaudio.html).

Dette verktøyet har vært en viktig inspirasjon for hvordan lydsystemet presentert i denne oppgaven har blitt utviklet, da mye av tankegangen rundt ulike funksjoner og oppsett har blitt hentet herfra. Videre i oppgaven vil det derfor bli gitt mer nøyaktige beskrivelser av hvordan ulike konsept faktisk utføres i Wwise Spatial Audio, og hvordan jeg har valgt å gjøre det i det omgivelsesbaserte lydsystemet som et resultat av det.

4. Omgivelsesbasert lydsystem

I denne delen av oppgaven skal det redegjøres for hvordan det har blitt utviklet en prototype for det jeg kaller et omgivelsesbasert lydsystem eller *Environmental Sound System*. De ulike valgene som er gjort i utviklingen av dette systemet vil bli begrunnet og sammenlignet med kjente praksiser i bransjen som ble gjennomgått i kapittel 3. På bakgrunn av dette vil det også redegjøres for hvorfor det eventuelt har blitt valgt å gjøre ting annerledes. Mye av det som er gjort i utviklingen av dette systemet er inspirert fra andre anerkjente navn i bransjen, og deres gode og stabile løsninger på tilsvarende system. Eksempler på disse vil være ulike løsninger fra Guy Somberg sin bok *Game Audio Programming 2: Principles and Practices*, Stepan Boev sine løsninger for *Hitman* (IO Interactive, 2016), og ulike forslag presentert av Bjørn Jacobsen. I tillegg er noen av valgene gjort i forbindelse med utviklingen av dette systemet et resultat av direkte dialog og samtaler med ulike lyddesignere, som blant annet Bjørn Jacobsen (Ubisoft, CD Projekt Red) og Henriette Jenssen (IO Interactive).

For å få et bedre innblikk i hvordan det omgivelsesbaserte lydsystemet brukes i praksis, og mer detaljert oversikt over hvordan de ulike undersystemene er knyttet sammen, er det vedlagt en videofil som viser dette i sin helhet. Dette vedlegget heter *Vedlegg/Filmer/Env_Sound_System_Implementation_Example_W_Audio.MP4.* Det er også lagt med en videofil som demonstrerer hvordan dette systemet kan oppleves som en del av et komplett lyddesign. Denne videofilen heter *Vedlegg/Filmer/Env_Sound_System_Walkthrough_Showcase_Audio_Edit.MP4.* Dersom man ønsker å teste systemet selv, vil vedlagte prosjektfil være mer aktuell: *Vedlegg/UE4 – Wwise Prosjekt/Env_SoundSystem/Env_SoundSystem.uproject.* Ønsker man å få innblikk i hvordan Wwise prosjektet ser ut, er det også vedlagt med navn *Vedlegg/UE4 – Wwise Prosjekt/Env_SoundSystem/Env_SoundSystem_WwiseProject/ Env_SoundSystem_WwiseProject.wproj*. For dette kreves det som nevnt i innledningen at man har Unreal Engine 4 og Audiokinetic Wwise installert. Anbefalte versjoner for dette prosjektet er Unreal Engine 4.27.2, og Wwise 2021.1.6.7774. Det kreves også at man bruker *Windows 10* som operativsystem.

Det er også viktig å nevne at dette spesifikke systemet vil fungere for *tredimensjonale spill som hovedsakelig operer ut ifra et førstepersons, eller tredjepersons perspektiv*. Det er også slik at *ettersom systemet prøver å simulere fysikkens lover og regler i forhold til lyd og lydutbredelse*, vil det *ikke nødvendigvis være egnet for spill som ikke baserer seg på dette*. Eksempler på spilltyper eller genrer hvor dette systemet kan ansees som uegnet vil være todimensjonale spill, eller tredimensjonale spill som utspiller seg i omgivelser hvor de fysiske lovene og reglene vi forholder oss til i den virkelige verden ikke gjelder. Dette kan være spill som for eksempel finner sted i det åpne verdensrommet.

4.1. Undersystem for statisk tredimensjonale audio emittere

I denne delen av oppgaven skal vi se på hvordan vi i dette undersystemet lager en statisk tredimensjonal audio emitter, som beskrevet i kapittel 3.1. Det vil derimot være noen forskjeller på hvordan vi løser det her i forhold til hva som er presentert i det kapittelet. Metoden som blir presentert her vil være enkel, men effektiv når det gjelder å kontrollere lydkilder og unngå unødvendig ressursbruk.

4.1.1. Culling

I kapittel 3.1 ble det nevnt at en vanlig praksis i forbindelse med bruken av statiske audio emittere er å plassere mange ulike varianter av dem ut i en spillverden og aktivere dem når banen starter. Det ble også nevnt at overdrevent bruk av denne praksisen kan føre til negativ påvirkning på spillets ytelse, som ofte resulterer i uønsket oppførsel for lydene (for eksempel voice starvation). For å kunne unngå dette, ble culling nevnt som en anvendbar metode. Dette kunne gjøres ved å begrense antall instanser av lyder som kunne spilles av samtidig, eller endre oppløsningen til lyden. Wwise og UE4 har sine egne metoder for å håndtere culling av ulike objekter, men i et utdrag fra en epost - utveksling med Henriette Jenssen (Lyddesigner ved IO Interactive) angående relevansen ved å lage en egen culling - funksjon for audio emittere, skriver hun:

> Du nevner at det kan være lurt å stoppe 3D lyder når man er utenfor en viss radius. Wwise har sitt eget culling system som kan ta seg av det. Men, selv om Wwise har dette, ender likevel alle med å lage sine egne culling systemer, fordi Wwise sitt system ofte fjerner feil lyd, eller er ikke til å stole på, så jeg synes egentlig det er relevant nok (H. Jenssen, personlig kommunikasjon, 15. september. 2021).

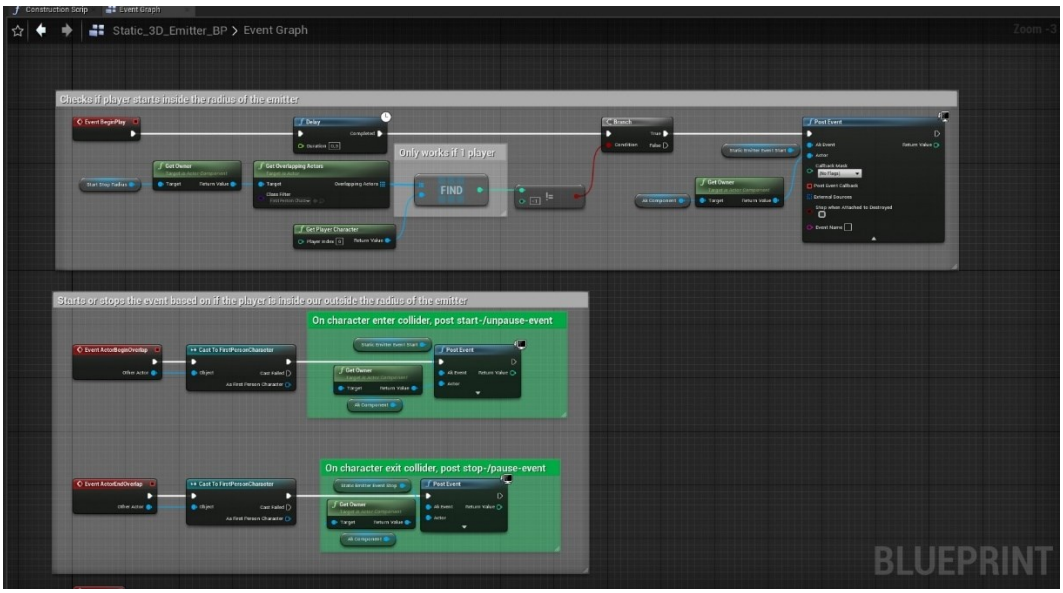
For å forhindre unødvendig prosesseringsbruk i det omgivelsesbaserte lydsystemet er det derfor laget en egen måte å culle lydene for en audio emitter på, som er mer pålitelig enn den som er innebygd i Wwise (og UE4). En variant av denne metoden er også beskrevet av Bjørn Jacobsen i en undervisningsserie på YouTube ved navn *Wwise & Unity*. Måten vi gjør dette på er først og fremst ved å være bevisste over hva funksjonen til en *attenuasjonsradius* faktisk gjør; *den definerer i all hovedsak en distanse for hvor hørbar lyden er*. Innenfor radiusen vil man kunne høre lyden, men det vil man ikke dersom lytteren står utenfor radiusen. Det vil derfor være unødvendig bruk av ressurser å la lydene spilles av dersom lytteren ikke er innenfor en hørbar distanse. Dette danner grunnlaget for hvordan vi kan culle en audio emitter i dette undersystemet.

I Wwise kan vi lage to ulike events for samme lyd. Én som spiller den av, og én som stopper den. I det omgivelsesbaserte lydsystemet kan vi da definere tre ulike sett med regler for statiske audio emittere:

1. Hvis spilleren entrer radiusen så begynner lyden å spille
2. Hvis banen starter med spilleren innenfor radiusen så begynner lyden å spille
3. Hvis spilleren er utenfor radiusen så stopper lyden (så lenge lyden spilles av)

Dette sørger for at lydene som blir spilt av i en audio emitter kontrolleres i undersystemet og forhindrer unødvendig prosesseringsbruk når lydene ikke er hørbare. I tillegg forhindrer dette upålitelig oppførsel fra Wwise og UE4 sine innebygde metoder for håndtering av culling. En konkret visualisering av dette kan sees i vedlegg

Vedlegg/Filmer/Env_Sound_System_Concept_Walkthrough_W_Music.MP4, med tidsstempel 01:06 – 02:22.



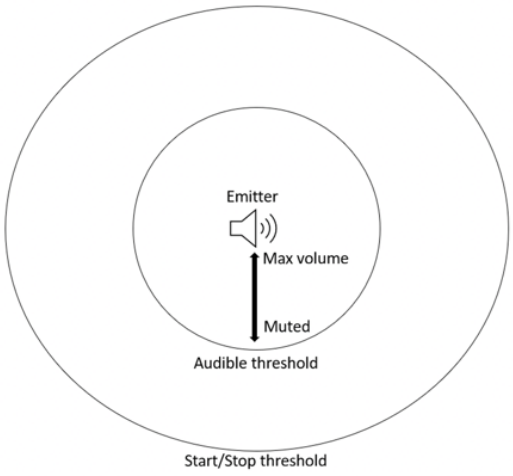
Figur 22: Eget utdrag fra koden som definerer reglene for kontrollering av en audio emitter. Øverste kodeblokk sjekker om spilleren starter innenfor radiusen til audio emitteren, mens kodeblokkene under aktiverer en start eller stopp event ut ifra om spilleren går inn eller ut av radiusen.

4.1.2. Forhindre like avspillingspunkt

Som nevnt i kapittel 3.1 kan flere av disse audio emitterene inneholde lyder som gjentar seg eller «loopes» (source loops). Dette presenter et nytt problem for måten undersystemet er satt opp på. Slik det er nå vil lyden spilles av fra start, hver gang man går innenfor radiusen til audio emitteren. I epost – utvekslingen med Henriette forklarte hun problematikken slik:

> Hvis lyden din er litt mere kompleks enn bare en statisk lyd, vil du få problemer hvis du gjenstarter den fra begynnelsen hva gang du går inn i dets radius igjen. Ta en radio for eksempel. Hvis du går inn og ut av radiusen, hvordan kan du unngå at sangen som spiller fra radioen ikke starter fra begynnelsen hver eneste gang (H. Jenssen, personlig kommunikasjon, 15. september. 2021)?

Det er flere måter man kan løse dette på. En løsning kan være å lage en Wwise event som demper lyden istedenfor å stoppe den. Dette vil da føre til at lyden fortsatt spilles av, og bruker ressurser selv om den ikke er hørbar. For å gjøre dette mindre ressurskrevende kan man lage en ekstra radius for audio emitteren utenfor den andre, som definerer start – og stopp punkt. Man bruker da to radiuser; *en ytre radius som beskriver når denne kan starte og stoppe*, men ikke hvor den er hørbar og ikke hørbar. Denne bør være stor nok til at lytteren bruker litt tid på å komme seg innenfor det hørbare området til audio emitteren, slik at punktet hvor lytteren forlater det hørbare området og kommer tilbake er avhengig av hvor raskt spilleren beveger seg. *Den andre (indre) radiusen definerer et område hvor lyden er hørbar eller ikke*, i form av å kun dempe lyden. Dette kan skape en del variasjoner for avspillingspunktet, men hvis spilleren konsekvent når det hørbare området like raskt, vil dette resultere i at man opplever at lyden starter fra samme punkt.

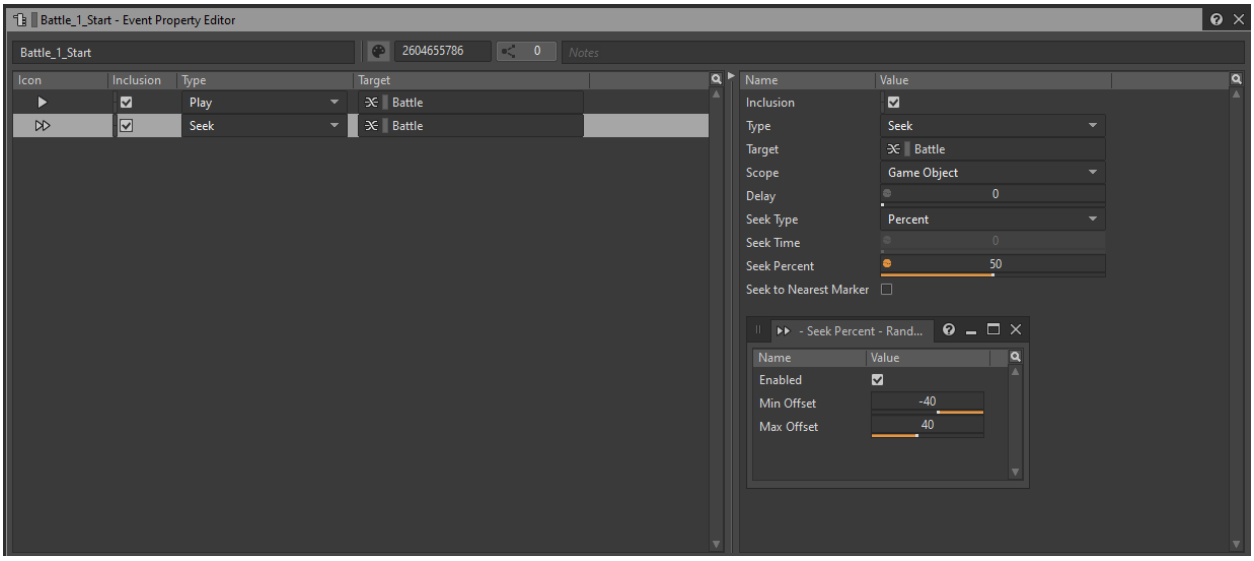


Figur 23: Egen representasjon av hvordan en dobbeltradius kan hindre opplevelsen av likt startpunkt hver gang. Ytre radius viser hvor lyden starter eller stopper, mens den indre radiusen viser hvor lyden er hørbar eller ikke (attenuasjon).

En annen og kanskje mer sikrere måte å løse denne problematikken på er ved å lage en regel som sier at hver gang man går innenfor attenuasjonsradiusen til audio emitteren, så skal *lyden spilles av fra en tilfeldig plass i lydforløpet*. Man burde også da kunne styre hvor tilfeldig dette skal være,

og i hvor store spenn lyden skal kunne endre startpunkt. Dette kan gjøres på flere måter; en av dem er ved å programmere det direkte inn i Unreal Engine 4 og styre det der, mens en annen måte er å bruke en innebygd funksjon i Wwise som heter *seek.* Denne brukes for å fortelle kilden om å søke til en vilkårlig posisjon i lydforløpet og starte derfra. Hvor vilkårlig dette skal være kan endres i funksjonens parameter i Wwise. På grunn av at de fleste mellomvarer har en eller annen form for å gjøre dette, har denne oppgaven brukt seek – funksjonen i Wwise, istedenfor å hardkode det inn i undersystemet. På den måten slipper man mer unødvendig og muligens komplisert kode i spillmotoren.

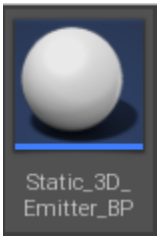
Et eksempel på hvordan dette oppleves finnes i vedlegg *Vedlegg/Filmer/Env_Sound_System_Walkthrough_Showcase_Audio_Edit.MP4,* med tidsstempel 03:59 – 04:08. I dette eksempelet går spilleren forbi flere fakler med audio emittere koblet til seg. Hver audio emitter spiller av en fakkel - lyd. Seek – funksjonen fører til at alle lydklippene for faklene starter på ulike tidspunkt hver gang, for hver audio emitter. I dette eksempelet resulterer det i at vi kan gjenbruke én lydfil uten at det virker som om samme lydklipp brukes for alle faklene. Dette fører til mindre ressursbruk uten å ødelegge den immersive opplevelsen.



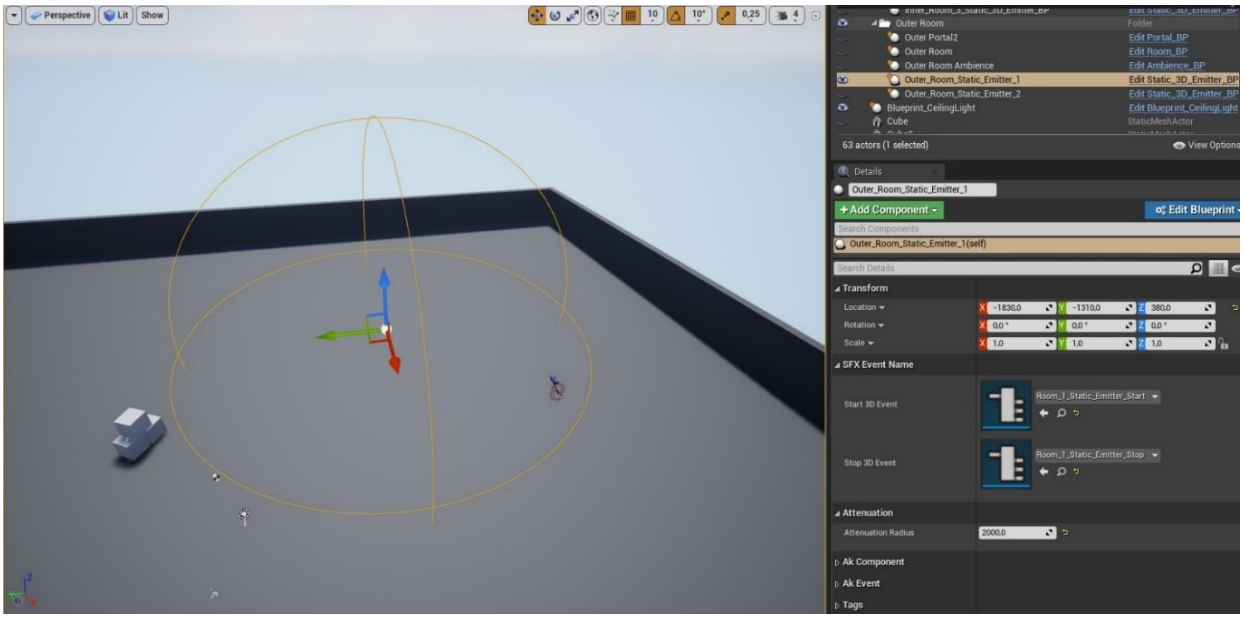
Figur 24: Eget utsnitt fra Wwise. Eksempel på bruken av seek funksjonen. I dette tilfellet skal den i utgangspunktet starte halvveis i lydklippet (seek percent), men ettersom det er lagt inn en vilkårlig verdi som kan være -40 eller +40 relativt til seek percent verdien som er 50, vil lyden hver gang den spilles av starte på en vilkårlig plass i lydfilen som er mellom 10 og 90%.

4.1.3. Oppsett i det omgivelsesbaserte lydsystemet

Hvordan ser en slik audio emitter ut i selve systemet? Dette skal være et gjenbrukbart verktøy for lyddesignere. For å gjøre det intuitivt og enkelt å håndtere, er audio emitteren derfor et spillobjekt som skal kunne dras ut i spillverdenen og plasseres der den ønskes, med den attenuasjonsradiusen som ønskes (denne attenuasjonsradiusen må samsvare med den man bestemmer i Wwise). *Figur 25* viser objektet til audio emitteren som blir plassert ut i spillverdenen, mens *figur 26* viser hvordan selve undersystemet ser ut.



Figur 25: Dette objektet plasserer man ut i spillverdenen.



Figur 26: Etter at den er plassert, skriver man inn lengen på attenuasjonsradiusen, og navnet på start og stopp eventen fra Wwise.

Selv om dette delkapittelet viser hvordan en statisk tredimensjonal audio emitter ser ut, vil *hovedprinsippene for hvordan den er bygd opp være felles for alle former for audio emittere* i dette prosjektet. Det vil si at de består av en start - og stopp event (med noen unntak) og med en attenuasjonsradius. Dette gjelder selv om de kanskje ikke kan *plasseres* ut i spillverdenen, eller er en del av et annet undersystem.

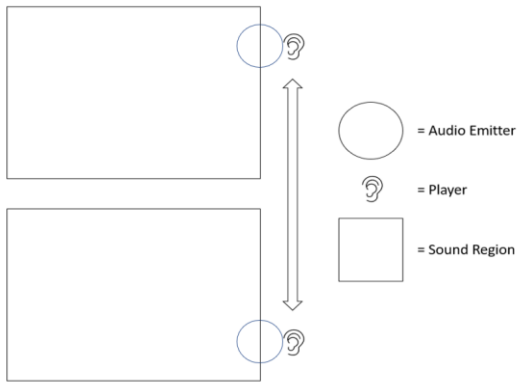
4.2. Undersystem for atmosfærelyder

I kapittel 3.2 ble det vist eksempler på metoder for hvordan et atmosfæresystem ofte lages i spill. I dette kapittelet skal vi se på ulike variasjoner og kombinasjoner av disse, som resulterer i et komplett undersystem for atmosfærelyder. Undersystemet vil da bestå av et aspekt som omhandler todimensjonale atmosfærelyder, og et annet som består av tredimensjonale atmosfærelyder. Vi vil også se hvordan metodene presentert i dette undersystemet skiller seg fra de som ble nevnt i kapittel 3.2.

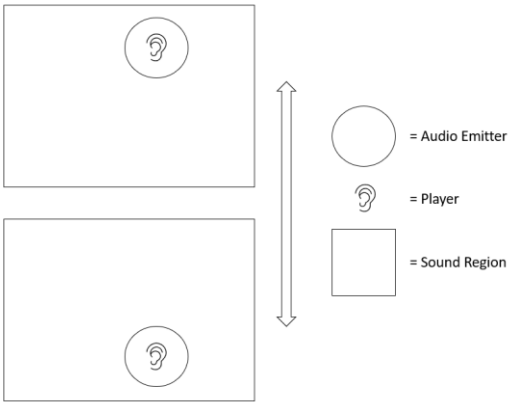
4.2.1. Todimensjonale atmosfærelyder

I motsetning til metodene for å lage todimensjonale atmosfærelyder som ble presentert i kapittel 3.2.1, vil det her brukes en form for Sound Shape Tool (boks som definerer et område for en spesifikk atmosfærelyd), *kun* for å definere selve området hvor lyden kan spilles av. Dette betyr at boksen *ikke genererer eller spiller av en lyd* med en attenuasjonsradius i seg selv. Et slikt området kan ofte defineres som en *Sound Region* eller *lydregion.* Det er flere grunner til at det er gjort slik; en av grunnene er at boksen hovedsakelig skal bidra til å kontrollere lydene og hvordan de oppfører seg (se kapittel 3.3.4). For å gjøre dette utføres flere beregninger i en Sound Region, som skal forklares nærmere i kapittel 4.3. Den skal også fungere som en slags sammenkobling for hele det omgivelsesbaserte lydsystemet. For å unngå ekstra ressursbruk på noe som allerede bruker prosesseringskraft på beregninger og kontroll av ulike konsept i systemet, vil ikke denne boksen spille av noen form for lyd.

I dette aspektet ved undersystemet brukes det en egen dynamisk audio emitter for å spille av den todimensjonale atmosfærelyden. Denne audio emitteren er ikke statisk, men beveger seg med spilleren og kan *kun bevege seg med sitt senterpunkt innenfor grensene til den aktuelle lydregionen*. Den vil hele tiden forsøke å få samme posisjon som spilleren, men ettersom den kun kan bevege seg innenfor denne boksen, vil dette kun være mulig når spilleren befinner seg i den. Dette resulterer i at når spilleren er utenfor lydregionen, vil audio emitteren følge spilleren langs kantene eller grensene, og i det spilleren entrer regionen, vil audio emitteren ha samme posisjon som spilleren.



Figur 27: Egen illustrasjon av hvordan audio emitteren hele tiden prøver å få samme posisjon som spilleren, men på grunn av regionens grenser ender den istedenfor opp med å følge spilleren innenfor boksen.



Figur 28: Egen illustrasjon av hvordan audio emitteren hele tiden prøver å få samme posisjon som spilleren, og når spilleren er innenfor boksen, vil nettopp dette skje.

For å kunne sjekke hvor audio emitterens plassering er i forhold til spilleren, blir det gjort en sjekk mellom hvert 0.01 til 0.1 sekund for å oppdatere denne posisjonen. I store mengder kan slike hyppige sjekker være ressurskrevende. Dette skjer derfor kun når spilleren er innenfor en viss avstand til lydregionen, eller befinner seg i den. Denne avstanden kan bestemmes av lyddesigneren, slik at den kan endres etter prosjektets behov. På den måten har man større kontroll over prosessene i dette aspektet ved undersystemet, slik at man kan unngå unødvendig ressursbruk.

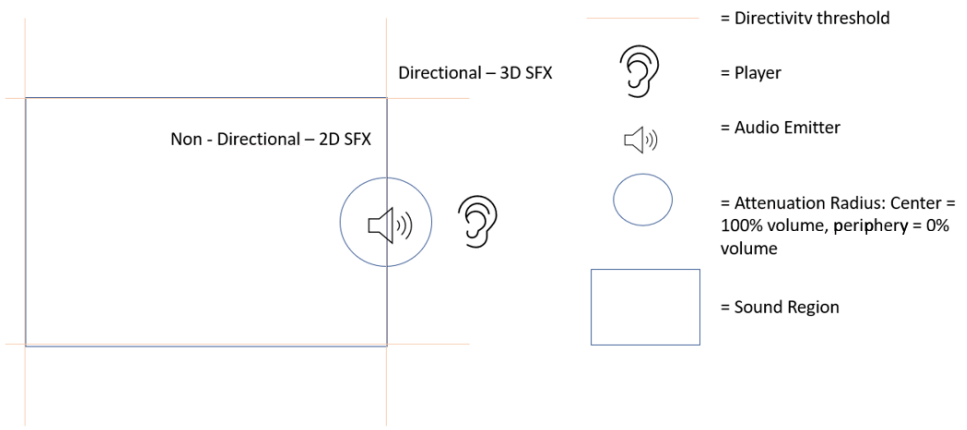
Ettersom den dynamiske audio emitteren er bygd opp av de samme prinsippene som den statiske audio emitteren, vil prosesseringskraften som brukes på avspillingen av selve lyden også kontrolleres med en start og stopp event. I tillegg til dette vil attenuasjonsradiusen til den dynamiske audio emitteren sørge for sømløse overganger mellom å starte og stoppe lyden, gjennom avstandsbaserte inn -og uttoninger. Hvordan dette oppsettet ser ut er vist i vedlegget *Vedlegg/Bilder/Ambience_CloseUp_Folder/Ambience_CloseUp.jpg.*

En av hovedgrunnene til at det todimensjonale aspektet ved undersystemet for atmosfærelyder er bygd opp slik de tidligere avsnittene beskriver, er også for å kunne *håndtere direksjonaliteten til lyden* på en mer korrekt måte. Som nevnt i kapittel 3.2.1 kan en todimensjonal atmosfærelyd beskrives som ikke – direksjonell og droneaktig i sin auditive karakteristikk. Det vil også være en lyd som oppfattes som direkte tilhørende det miljøet eller de omgivelsene det befinner seg i. Med unntak av at lyden blir mer diffus jo lenger unna man er, blir det ikke i større detalj beskrevet hvordan man opplever den når man er utenfor omgivelsene for atmosfærelyden (lydregionen). Selv om en atmosfærelyd *ofte ikke oppleves å ha en spesifikk lydkilde når man er inne* i den aktuelle regionen, vil den derimot ha en *opplevd direksjonalitet når man er utenfor* det; man oppfatter ikke nødvendigvis lydkildens originale opphav i lydregionen, men man registrerer at lyden kommer fra den spesifikke regionen.

For å kunne håndtere direksjonalitet i dette aspektet ved undersystemet er det i sammenheng med den allerede nevnt fremgangsmåten blitt brukt en RTPC (Real – Time Parameter Control). Hvis spilleren befinner seg utenfor en lydregion, vil den tilhørende atmosfærelyden være 100%

direksjonell. Da vil spillkoden sende en beskjed til Wwise om å gjøre lyden tredimensjonal (mono) og direktiv, ved å endre parameter i mellomvaren (eller UE4 sin egen lydmotor). I realiteten vil lyden spilles av fra punktet til audio emitteren, og ettersom den hele tiden vil følge spilleren innenfor den aktuelle lydregionen, vil lyden oppfattes som å komme derfra uansett hvor spilleren befinner seg. Når spilleren derimot bestemmer seg for å gå inn i området, vil det bli sendt en ny beskjed til Wwise om å gjøre lyden todimensjonal, slik at den virker som en del av omgivelsene og ikke som om den har en spesifikk kilde inne i regionen. Den blir da ikke – direksjonell.

På grunn av attenuasjonsradiusen, vil dette skje på det punktet hvor volumet er på sitt mest hørbare. Hvor fort denne overgangen skal skje kan bestemmes av lyddesigneren. Andre måter å løse dette på er ved å bruke en ren mono lydkilde uten å endre direksjonaliteten når man går inn og ut av lydregionen. Lyden vil da være direksjonell utenfra, men inne i området vil den opplevde plasseringen være i senter av lydbildet, og ikke omringende eller tilstedeværende overalt rundt spilleren. Fordelen med metoden presentert i de tidligere avsnittene, er at man da har muligheten til å bruke surround eller lignende format for å livliggjøre atmosfærelydene enda mer. I denne oppgaven styres direksjonalitetsendringen som sagt fra parameterendringer i Wwise, men dette konseptet kan overføres til de fleste mellomvarer og lydmotorer. Et eksempel på hvordan dette aspektet ved undersystemet fungerer kan sees i vedlegg *Vedlegg/Filmer/Env_Sound_System_Concept_Walkthrough_W_Music.MP4,* med tidsstempel 03:51 – 05:31.



Figur 29: Egen illustrasjon av direksjonalitetskonseptet for de todimensjonale atmosfærelydene. De gule linjene viser hvor lyden er direksjonell, og hvor den ikke er det. Utenfor linjene til en Sound Region vil lyden være direksjonell, men ikke innenfor. Høyttalersymbolet med sirkelen representerer en audio emitter med tilhørende attenuasjonsradius.

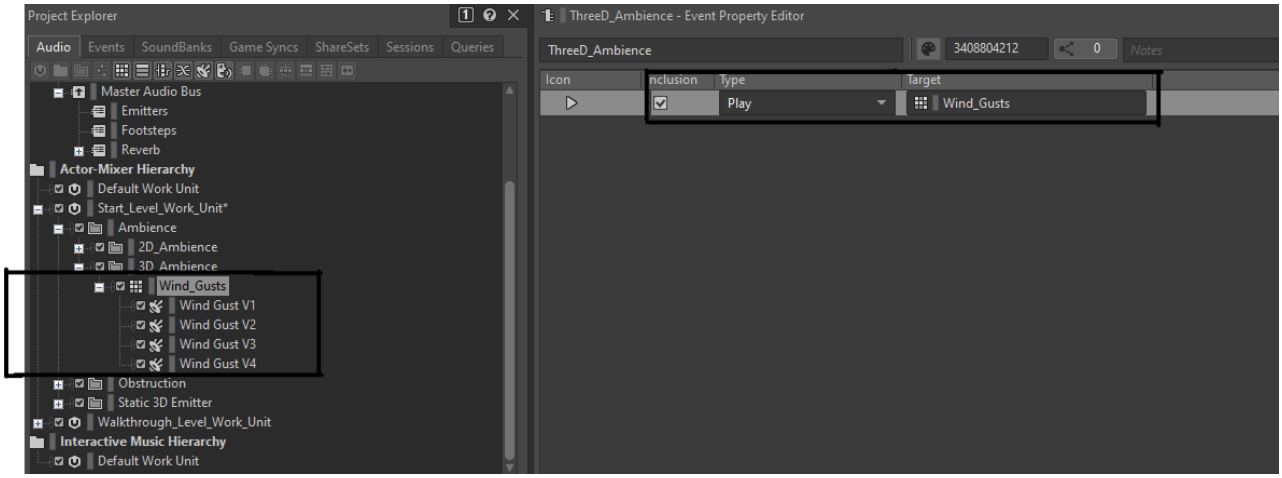
Et annet konsept som er med på å gjøre det omgivelsesbaserte lydsystemet anvendbart i større skala er at det tar hensyn til *nestete miljøer* (Omgivelser/regioner inne i andre omgivelser/regioner), gjennom en form for prioritering. For å visualisere dette kan vi se for oss en lydregion med atmosfærelyder for et stort utendørsområde. I dette området vil det også være plassert et hus. *Står man inne i huset, vil man ikke høre atmosfærelydene utenfra på samme måte som hvis man står utenfor*. Det samme gjelder motsatt vei; *man vil ikke høre atmosfærelydene fra huset like godt utenfra, som innendørs*. Det omgivelsesbaserte lydsystemet tar hensyn til dette og behandler atmosfærelydene ut ifra hvor spilleren befinner seg. Dette kan også tilpasses dersom man ønsker det. Hvis man har ulike lydregioner og områder som man ønsker skal dele samme atmosfærelyder, kan man velge å ikke behandle lydene ulikt. Hvis man for eksempel har en skog med ulike stier, kan man gi stiene egne lydregioner for lydutbredelse (forklares i kapittel 4.3), og plassere dem inne i en større lydregion med skogsatmosfære – lyder, og velge at lydregionene for stiene ikke skal gjøre noe med disse lydene. Åpne stier kan da ha egne lydutbredelsesveier, som er ulike utbredelsesveiene til tett skog, men ha like atmosfærelyder. Et eksempel på nestete miljøer finnes i vedlegget *Vedlegg/Filmer/Env_Sound_System_Advanced_Occlusion_Demo.MP4,* med tidsstempel 01:58 – 02:50. Et annet eksempel finnes i vedlegget *Vedlegg/Filmer/Env_Sound_System_Concept_Walkthrough_W_Music.MP4,* med tidsstempel 05:37 – 06:30.

Figur 30: Valg for om atmosfærelydene skal behandles ulikt eller ikke i det omgivelsesbaserte systemet.

4.2.2. Tredimensjonale atmosfærelyder

Denne oppgaven tar i bruk en variant av *Random FX* konseptet, som ble nevnt i kapittel 3.2.2. Grunnen til at dette er en god og anvendbar metode, er fordi i at i tillegg til å forbedre det immersive aspektet, kan det også brukes til å redusere antall vanlige statiske audio emittere i en spillverden. Ved å minke bruken av slike emittere til fordel for tilfeldig generering av tredimensjonale lyder, kan man redusere prosesseringskraften betraktelig. Dette er også mer egnet for lyder som ikke gjentar seg eller looper, såkalte *one – shots* (Soundnation, 2019). Hvordan det tredimensjonale aspektet ved undersystemet for atmosfærelyder er bygd opp skal forklares i neste avsnitt.

Som lyddesigner kan man først og fremst bestemme hvilken event som skal aktiveres i Wwise. Denne eventen vil velge en spesifikk variant av en lyd, blant et visst antall mulige varianter, og spille av denne. Hver gang Wwise mottar beskjed om å aktivere en slik event vil dette valget være tilfeldig. I Wwise har man også muligheten til å endre flere parameter for hver instans av lydavspillingen. Disse kan være vektleggelse av enkelte lydvarianter, pitch – endringer (tonehøyde), volumendring og lignende. Hver gang en event aktiveres, vil Wwise med andre ord velge en variant av lyden og spille den av med eksempelvis lavere volum eller økt tonehøyde. Ved å gjøre dette kan man skape enda mer variasjon i lydene. Slike endringer kan gjøres i de fleste mellomvarer, og i den innebygde lydmotoren til Unreal Engine 4.



Figur 31: Eget utsnitt fra Wwise. Boksen til venstre viser hvilke alternative lyder som kan velges mellom i en såkalt «Random Container», mens boksen til høyre viser funksjonen til eventen, som her er å spille av en lyd i denne containeren.

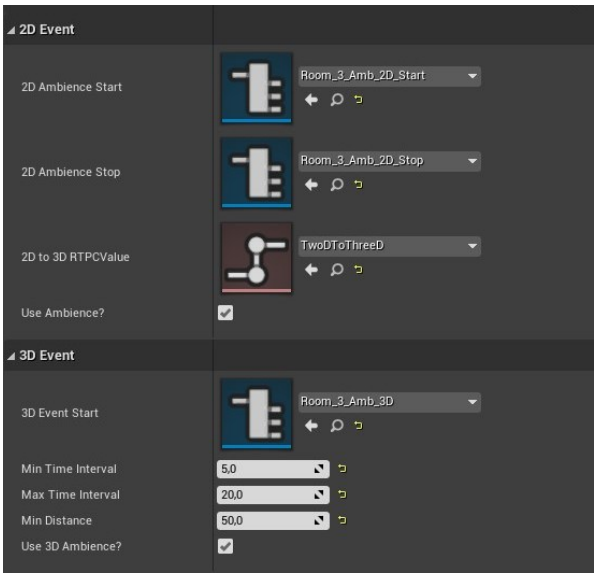
I spillmotoren, vil det hver gang en av disse lydene spilles av, skapes en midlertidig audio emitter som lyden kommer fra. Når den er ferdig spilt, vil audio emitteren fjernes. Dette gjøres for å unngå et stort antall inaktive audio emittere. Foreløpig er det bare vist hvordan lydene spilles av i Wwise, og skapes i spillmotoren. Hvordan dette aspektet ved undersystemet fungerer i praksis og oppfører seg som et resultat av ulike forhåndsbestemte regelsett skal forklares i de neste avsnittene.

Som lyddesigner får man muligheten til å skrive inn verdier for et tidsintervall, som er med på å bestemme hvor ofte de tredimensjonale atmosfærelydene skal genereres. Dette gis i form av et minimum og maksimum flyttall. Undersystemet vil så registrere dette tidsintervallet, generere en audio emitter og spille av lyden. Dette skjer på et *tilfeldig tidspunkt innenfor de angitte tidsrammene*. Slik kan man skape uforutsigbarhet for lydene og gjøre dem mer ekte. Hvis man derimot ønsker et mer konstant tidsintervall for når lydene skal spilles av, kan man også gjøre dette.

I tillegg til å skape tilfeldighet for når lydene skal avspilles, vil dette aspektet ved undersystemet også gi mulighet til å skape *tilfeldighet i hvor og hvor langt unna spilleren lyden skal spilles av*. Slik det er nå, vil audio emitterene være hardkodet til å genereres i en tilfeldig plassering 360

grader rundt spilleren. Dette kan man derimot gå inn i koden og endre selv, hvis man ønsker. Dersom man vil at lydene for eksempel kun skal spilles av bak spilleren, kan man endre det der.

Man har også muligheten til å definere et distanseintervall for hvor langt unna spilleren audio emitterene kan genereres. I dette aspektet ved undersystemet er maksimum avstand alltid lik det som er maksimum punkt for lydens attenuasjonsradius, men dette kan endres i koden dersom det er ønskelig. Denne maksimum avstanden er valgt for å unngå å generere audio emittere i distanser hvor lydene ikke er hørbare for spilleren. I tillegg til dette kan man, hvis man ikke ønsker at lydene skal spilles av for nært spilleren, skrive inn en ønsket minimums - distanse. Da vil audio emitterene genereres i en tilfeldig avstand fra spilleren mellom den angitte minimum og maksimum - avstanden. Dersom man foretrekker at distansen for generering av audio emittere skal være mer konstant og forutsigbar, kan dette endres ved å gi samme verdi til både minimum og maksimum avstanden. Gjennom metodene som er presentert i dette underkapittelet, vises det hvordan dette aspektet ved det omgivelsesbaserte lydsystemet kan skape stor variasjon, livlighet og dynamikk for tredimensjonale atmosfærelyder.



Figur 32: Slik ser det interaktive aspektet ved denne oppgavens undersystem for atmosfærelyder ut. Her bestemmer man eventnavn på alle atmosfærelydene som skal spilles av og stoppes, navn på RTPC som styrer 3D - 2D endring, og mengde vilkårlighet for de tredimensjonale atmosfærelydene.

I en epost – utveksling med lyddesigner Henriette Jenssen, angående tilfeldig generering av tredimensjonale atmosfærelyder, sa hun følgende:

> En ting du kan tenke på er dette: Du nevner du vil ha varierte one – shots som spiller rundt spilleren i ambiens - rommet. Men hva skjer hvis du står rett ved siden av en stor tykk vegg til høyre for deg? Vil du fortsatt kunne høre tilfeldige lyder spille på høyre side? Eller det mulig å kunne kansellere lyder til høyre for deg hvis det er en stor gjenstand plassert der (H. Jenssen, personlig kommunikasjon, 15. september. 2021)?

Innholdet i epost – utdraget belyser noen utfordringer i forbindelse med de tredimensjonale atmosfærelydene som bør håndteres; *det vil ikke virke naturlig om lydene spilles av midt i ulike objekt*, eller fra *objekt som ikke har noen form for assosiasjon med den avspilte lyden*. Dette kan bryte med den realistiske opplevelsen og ødelegge immersjonen. Det vil heller *ikke være ressursvennlig å la lyder genereres i andre omgivelser enn der spilleren befinner seg*. Dette er fordi man da må bruke ekstra prosesseringskraft på å beregne og tilpasse lydene til andre omgivelser i tillegg til den lydregionen hvor spilleren er. På grunn av dette vil det ikke genereres audio emittere utenfor det området eller den regionen spilleren befinner seg i, uansett om attenuasjonsradiusen eller maksimum avstandsverdi strekker seg forbi dette. For å forhindre at en audio emitter plasseres midt i et eksisterende spillobjekt, vil det i forkant av hver generering gjennomføres en sjekk som registrerer om den planlagte posisjonen til audio emitteren kolliderer med et objekt eller ikke. Hvis den kolliderer med noe, vil ikke audio emitteren skapes, og koden tilbakestilles og klargjøres for neste sjekk.

For de tredimensjonale atmosfærelydene vil nestete omgivelser også ha noe å si. Befinner spilleren seg i en lydregion med mindre regioner eller områder i seg, vil ikke atmosfærelydene kunne genereres innenfor disse (mindre områdene), da de vil bli registrert som kolliderende objekt. Et eksempel på hvordan det tredimensjonale aspektet ved undersystemet for atmosfærelyder fungerer finnes i vedlegget *Vedlegg/Filmer/*

Vedlegg/Filmer/Env_Sound_System_Concept_Walkthrough_W_Music.MP4, med tidsstempel 06:31 – 07:22.

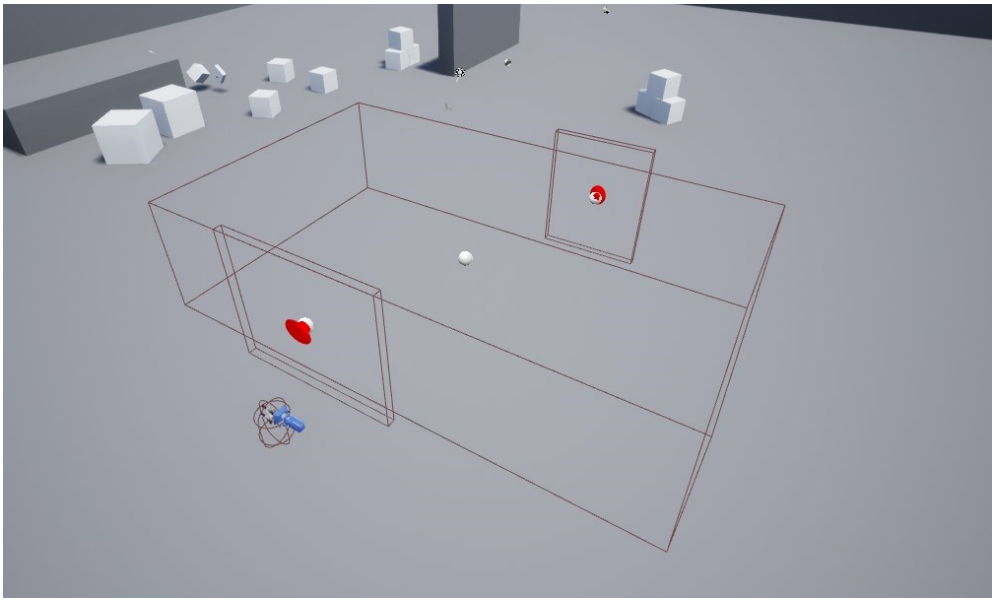
4.3. Undersystem for lydutbredelse

I denne delen av oppgaven skal vi se nærmere på hvordan undersystemet for lydutbredelse er laget i dette prosjektet. Som nevnt i kapittel 3.3.4 har ikke spillmotorer en innebygd forståelse om hvordan lyd beveger eller utbrer seg i ulike omgivelser. Dette må derfor lages selv, og kan ofte bli ressurskrevende, noe som gjør at det er ugunstig å bruke samme geometriske beregninger som for eksempel lys. Man ender derfor ofte opp med å lage egne geometriske løsninger for lyd. Dette er viktig å nevne fordi at det undersystemet som skal presenteres vil på mange måter fungere som hjørnestenen eller hjertet i det omgivelsesbaserte lydsystemet. Dette er fordi det binder sammen alle de ulike konseptene (og undersystemene) som har blitt gjort rede for, og sørger for at lydene får riktige egenskaper i forhold til hvordan de påvirkes av de omgivelsene de befinner seg i. Løsningen for lydgeometrien vil derfor bli presentert først. Deretter vil vi gå dypere inn i hvordan de ulike aspektene ved undersystemet for lydutbredelsen har blitt utviklet.

4.3.1. Rom & Portaler

I kapittel 3.3.4 ble det nevnt et eksisterende verktøy for lydutbredelse som heter Wwise Spatial Audio. Dette verktøyet består av et system som kalles for *Rooms* og *Portals* (Rom & Portaler). Det ble også nevnt at det systemet brukes til å lage geometri som effektivt kan simulere lydutbredelse for lydkilder i andre rom. Begrepet Rom beskriver en region eller område, både inne og ute. Fra kapittel 4.2.1 vet vi at et slikt område kalles for en Sound Region, eller lydregion. Videre ble det nevnt at Rom blir koblet sammen med hverandre gjennom Portaler, som sammen utgjør et nettverk av flere rom hvor lyd kan spre seg. Rom og Portaler sørger da for at lydutbredelsesveier blir beregnet. Begrepene Rom og Portaler er ganske normale begrep som blir brukt i denne sammenhengen (Boev, 2015). Andre begrep som også blir brukt her er *Environments* og *Gates* (Jacobsen, 2018, 1:32). I denne oppgaven har det blitt brukt en lignende løsning for håndtering av lydutbredelse, og på grunn av den generelle begrepsbruken, har disse også blitt omtalt som Rom og Portaler. *Omgivelser eller Sound Regions vil fra nå av bli omtalt som Rom*.

Dette (Rom) er objektet lyddesigneren drar ut i verden og definerer som et lydområde med spesifikke egenskaper for lyden. I dette objektet skriver lyddesigneren inn mesteparten av all nødvendig informasjon for at systemet skal fungere.



Figur 33: Rom og Portaler i det omgivelsesbaserte lydsystemet. Et rom er den store kvadratiske boksen, mens portaler er de mindre boksene, med røde piler som peker i retningen lyden beveger seg.

Som nevnt i kapittel 4.2 kan disse rommene også bli plassert i hverandre og fungere som nestete rom, slik at de blir behandlet som ulike omgivelser. Portaler vil hovedsakelig representere åpninger i et rom hvor lyden kan komme ut ifra. Disse kan for eksempel være døråpninger eller vinduer. Hvordan disse ser ut er vist i vedlegget *Vedlegg/Bilder/Rooms_Portals_Folder/Portal_CloseUp.jpg*. For å gi et overordnet blikk på hvilken informasjon disse rommene kan få matet inn, er det her presentert en liste med tilhørende vedleggnavn for bilder. Lyddesigneren kan velge å gi informasjon om disse tingene:

1. 2D Atmosfærelyder

   (*Vedlegg/Bilder/System_CloseUp_Folder/System_Layout_1.jpg)*

   a. Start og stopp – eventnavn
   b. RTPC navn for overgang mellom 2D og 3D format
   c. Mulighet for å aktivere/deaktivere 2D atmosfærelyder

2. 3D Atmosfærelyder

   (*Vedlegg/Bilder/System_CloseUp_Folder/System_Layout_1.jpg)*

   a. Start – eventnavn
   b. Minimum og maksimum tidsintervall
   c. Minimum avspillingsdistanse
   d. Mulighet for å aktivere/deaktivere 3D atmosfærelyder

3. Okklusjon

   (*Vedlegg/Bilder/System_CloseUp_Folder/System_Layout_1.jpg)*

   a. RTPC navn for Okklusjon
   b. Standard okklusjon - verdi for rommet
   c. Mulighet for å aktivere/deaktivere okklusjon

4. Obstruksjon

   (*Vedlegg/Bilder/System_CloseUp_Folder/System_Layout_2.jpg)*

   a. RTPC navn for Obstruksjon
   b. Obstruksjon verdi som skal påvirke lyden dersom aktuelt
   c. Maksimum distanse for å beregne obstruksjon
   d. Mulighet for å deaktivere/aktivere obstruksjon

5. Sammenkoblede rom og portaler

   (*Vedlegg/Bilder/System_CloseUp_Folder/System_Layout_2.jpg)*

   a. Antall tilkoblede portaler og hva disse objektene heter
   b. Tilkoblede rom og informasjon om de er «åpne» eller «lukket»

6. Nestete rom

   (*Vedlegg/Bilder/System_CloseUp_Folder/System_Layout_2.jpg)*

   a. Valg for om rommet befinner seg i et annet et
   b. Navn på det ytre rommet

    c. Verdi for hvor langt inne i det indre rommet de ytre atmosfærelydene skal være hørbare
7. Ytre rom
   (*Vedlegg/Bilder/System_CloseUp_Folder/System_Layout_3.jpg)*
    a. Valg for om det ytre rommet har et rom i seg
    b. Valg for om atmosfærelyden skal behandles som en del av begge rommene (både ytre og indre) eller for seg selv
    c. Navn på det indre rommet
8. Audio emitter for atmosfærelydene
   (*Vedlegg/Bilder/System_CloseUp_Folder/System_Layout_3.jpg)*
    a. Navn på audio emitteren for atmosfærelydene

Når rom omtales som hjertet i det omgivelsesbaserte lydsystemet betyr det blant annet at mesteparten av all kode som utføres i systemet skjer i dette spillobjektet. I tillegg vil all informasjon lagt inn av lyddesigneren bli brukt her, eller sendt videre til andre objekt som er aktuelle (for eksempel audio emittere og portaler). Ved å ha mesteparten av den nødvendige informasjonen i ett objekt, økes brukervennligheten. I rom og portal – objektene vil alt av audio emittere registreres, og lydutbredelsesveier (og verdier) beregnes. Disse vil for eksempel beregnes basert på informasjonen som er lagt inn av lyddesigneren og spillerens plassering i lydgeometrien. Mye av koden som omhandler rom (og forsåvidt portaler) går derfor ut på å ha kontroll over de ulike audio emitterene og spillerens posisjon.

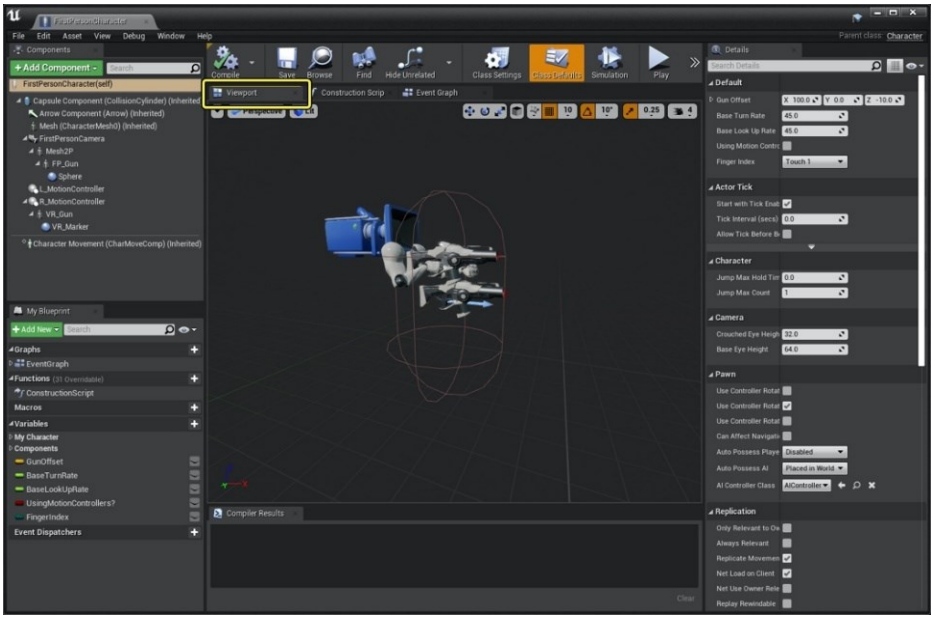
Ettersom mesteparten av beregningene for lydutbredelse og de konseptene som ligger derunder (obstruksjon, okklusjon og refleksjoner/romklang) gjøres i Rom og Portal objektene, vil disse bli grundigere forklart videre i dette kapittelet. For en mer detaljert representasjon av hvordan Rom og Portaler kobler sammen alle undersystemene, se utsnittene fra koden i vedleggene som finnes i mappen *Vedlegg/Bilder/Blueprints_Folder* og implementasjonseksempelet i videofilen *Vedlegg/Filmer/Env_Sound_System_Implementation_Example_W_Audio.MP4*.

4.3.2. Obstruksjon

Som nevnt i kapittel 3.3, er et viktig konsept innen lydutbredelse å kunne simulere hvordan lydbølger bøyer seg rundt et objekt eller utbrer seg gjennom åpninger (diffraksjon). Diffraksjon i denne oppgaven brukes både i obstruksjon og okklusjon, men på litt ulike måter ut ifra hva de representerer. I obstruksjon, vil definisjonen som beskriver diffraksjon som *fenomenet for hvordan lyd bøyer seg rundt et objekt* være mest aktuelt. Dette ser vi blant annet i definisjonen til Harris, og ut ifra hvordan Audiokinetic tolker dette fenomenet (som vist ut ifra figur 19). Derfor vil obstruksjon i denne oppgaven definere samme aspekt ved det akustiske fenomenet diffraksjon som nevnt i eksemplene ovenfor.

Det ble nevnt at en av de vanligste metodene for å lage et obstruksjon – system i spillutvikling er å sende en enkel usynlig stråle (raycasting) fra spilleren til lydkilden (eller omvendt), og dersom noe blokkerer den, blir direktelyden påvirket (ofte gjennom filtrering av høyfrekvenser og demping av volum). Refleksjonene til lyden vil derimot forbli uendret ettersom lydkilden og spilleren befinner seg i samme omgivelser.

Denne metoden er enkel og effektiv, men presenterer noen problemer som bør overveies før man designer et slikt system. Først og fremst må man definere hvor denne usynlige strålen faktisk kommer fra. I første persons 3D – spill, er spilleren ofte definert som en kapsel, med et kamera koblet til seg. Det kan også være en fysisk animert karakter, men i alle tilfeller vil det brukes et tilkoblet kamera for å gi første – persons perspektivet.



Figur 34: Eget eksempel på en generisk første – persons karakter i et tredimensjonalt spill.

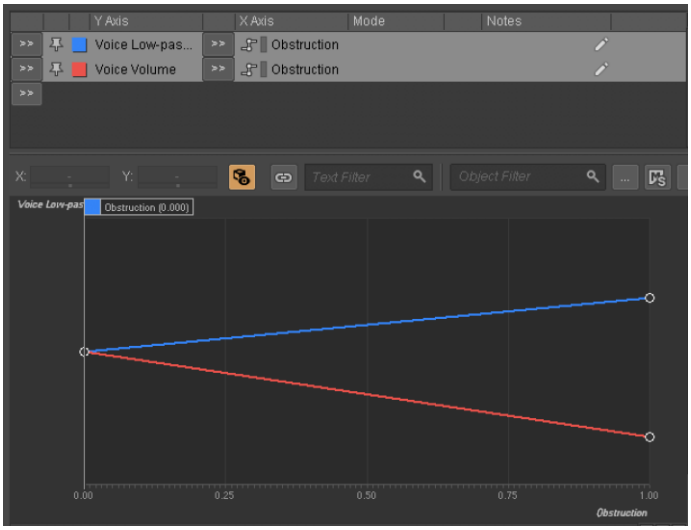
Det mange gjør i utviklingen av slike system er å definere spilleren som enten sender eller mottaker for strålen som sendes. For enkelthets skyld tar vi utgangspunktet i at spilleren er definert som sender. Dette innebærer ofte at senter for kapselen eller karakteren (spilleren) er startpunkt for strålen. Dette kan fungere i enkle sammenhenger, men et problem oppstår hvis spillerens *senterpunkt blokkeres for en lydkilde, og hodet (ørenes høydeplassering) ikke blir det*. Dersom et objekt med halvparten av spillerens høyde står mellom lydkilden og spilleren, vil direktelyden påvirkes selv om ingenting blokkerer veien mellom lydkilden og spillerens imaginære ører. Dette gir en urealistisk representasjon av diffraksjonsfenomenet, da lyden vil endres selv om ingenting blokkerer det som i virkeligheten faktisk mottar lyden.

For å unngå dette, vil derfor ikke strålen bli sendt fra spilleren, men fra kameraet som følger spilleren. Dette gjøres for å sikre at obstruksjon følger «line of sight», og at referansepunktet skal være så nært ørene som mulig. På denne måten sikrer man mest mulig realisme og økt immersjon. I *Vedlegg/Filmer/Env_Sound_System_Concept_Walkthrough_W_Music.MP4* vil det se ut som om strålen kommer fra senterpunktet til spilleren. Dette er kun gjort for å enklere demonstrere strålens retning.

Dersom strålen blir blokkert, vil spillkoden sende en beskjed til Wwise om å påvirke direktelyden med en verdi som følger en bestemt kurve. Denne kurven er definert for hver audio emitter gjennom en RTPC. Dette skjer også med en tidsbasert inn og uttoning av selve verdiendringen (interpolering), for å unngå brå endringer av direktelyden. Kurven lages i Wwise, og kan bestemme hvordan for eksempel filtrering av høyfrekvenser og demping av volum skal endres ut ifra ulike verdier langs kurven. Dette er definerbart for lyddesigneren, og kurvene kan være ulike for alle lydobjekt. Dette gir muligheter for detaljert simulering av det opplevde diffraksjonsfenomenet.

Lyddesigneren kan i dette undersystemet velge at dersom en lyd blir blokkert i et rom, så skal den aktuelle direktelyden påvirkes med en verdi mellom 0 og 1. Dette er verdiene som tilsvarer spesifikke endringer ut ifra kurven i Wwise, som nevnt i forrige avsnitt. Selv om obstruksjonsverdien er lik for alle lydobjekt i rommet, vil *lydpåvirkningen* være ulik ettersom *hvert lydobjekt har individuelle kurver*. Lyddesigneren kan også velge en spesifikk distanse for hvor langt unna spilleren obstruksjon kan beregnes for det aktuelle rommet. Dette gjøres for å unngå unødvendig bruk av prosesseringskraft på lydobjekt som er langt unna.

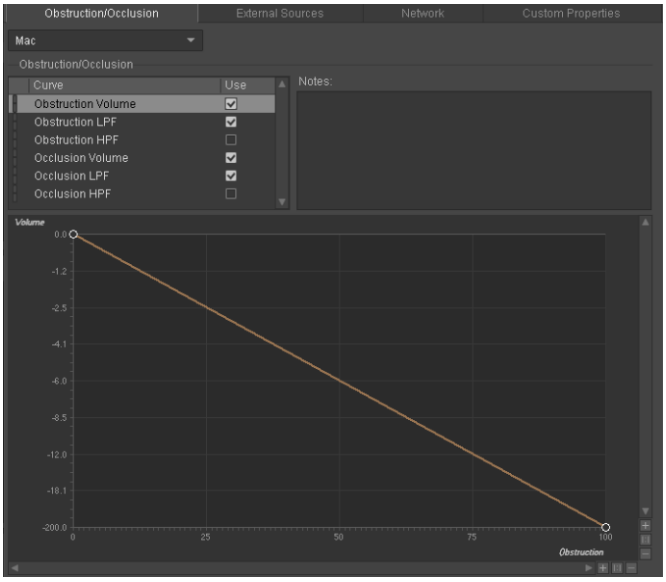
Figur 35: Verdiene man kan legge inn for obstruksjon i et rom i det omgivelsesbaserte lydsystemet.



Figur 36: Eget utsnitt av eksempel på hvordan lyddemping og filtrering av høyfrekvenser endres som et resultat av verdier mellom 0 - 1 for ett lydobjekt. Blå strek er filtrering av høyfrekvenser, mens rød strek er demping av volum.

Wwise har gjennom sitt Spatial Audio system (Rooms & Portals) også muligheten til å beregne diffraksjon, og i dette tilfellet obstruksjon. Hvorfor er det da ikke hensiktsmessig å bruke dette ferdigutviklede verktøyet? For det første kan Wwise Spatial Audio kun brukes i forbindelse med Wwise, noe som gjør det ugunstig dersom man vil bruke andre mellomvarer. I tillegg til dette kommer det med noen andre begrensninger som gjør det mer aktuelt å lage sin egen metode for beregninger av diffraksjon, og i dette tilfellet obstruksjon.

Som nevnt tidligere i kapittelet, kan vi i dette aspektet ved undersystemet for lydutbredelse selv definere ulike kurver for obstruksjon. Det ble også nevnt at disse kurvene kan være ulike for alle lydobjekt i et rom. Dette gir oss god kontroll over hvordan forskjellige lyder skal påvirkes. I Wwise sitt innebygde system, vil man kunne lage ulike kurver for parameter som lavpassfilter og demping av volum. I motsetning til metoden presentert i det omgivelsesbaserte lydsystemet, vil disse kurvene være *universale for alle lydobjekt i spillet og alle omgivelsene de befinner seg i*. Dette fører til færre muligheter og mindre detaljert simulering av det opplevde diffraksjonsfenomenet.



Figur 37: Eget utsnitt av kurvene for obstruksjon og okklusjon i Wwise sitt innebygde system.

Som nevnt i kapittel 3.3.2 er raycasting en funksjon som kan kreve mye ressurser, og vil derfor være ugunstig å bruke i enkelte tilfeller. For å beregne obstruksjon, er det for eksempel nødvendig å kontinuerlig sende stråler (mellom hver bildeoppdatering og 15ms) til alle audio emittere som finnes på en bane. Dette kan bli veldig ressurskrevende og ha en negativ påvirkning på spillets ytelse. På grunn av dette er det lurt å se på alternative løsninger, eller metoder for å kunne begrense ressursbruken.

I dette aspektet ved undersystemet for lydutbredelse vil beregninger for obstruksjon skje for audio emittere som *befinner seg i samme rom som spilleren*. Disse beregningene starter i det øyeblikket spilleren entrer det aktuelle rommet, eller hvis spilleren allerede befinner seg i det ved spillets oppstart. Vanligvis vil ikke et rom inneholde mer enn cirka 10 – 20 audio emittere, som i utgangspunktet er uproblematisk i forhold til ressursbruk. Hvis spilleren derimot befinner seg i et digert rom som inneholder et stort antall audio emittere, kan dette bli mer problematisk. For å forhindre overdrevent ressursbruk i slike situasjoner, er det derfor lagt inn en mulighet for å velge maksimum avstand for beregningen av obstruksjon. Dette vil si at alle audio emittere som er utenfor denne avstanden ikke vil bli inkludert i beregningene.

I tillegg til at obstruksjon beregnes for audio emittere som er i det samme rommet som spilleren, vil det også beregnes for audio emittere *som er i sammenkoblede rom*. Dette gjelder likevel *kun* dersom de *befinner seg bak en portal som er definert som åpen*. Dette fører til økt realisme for lyder som ellers kunne blitt neglisjert.

I det omgivelsesbaserte lydsystemet kan man også definere hvilke objekt eller materialer som skal ha en blokkerende egenskap for lydene, i forbindelse med obstruksjon. I tilfeller hvor det er snakk om veldig små objekt, eller objekt som brukes til okklusjon (vegger og lignende), vil *vanligvis* ikke disse ansees som å ha en blokkerende egenskap i forbindelse med obstruksjon. Likevel har man muligheten til å selv bestemme hva som skal defineres som en blokkerende egenskap i forbindelse med obstruksjon. En visuell fremvisning av obstruksjonskonseptet og hvordan det fungerer finnes i vedlegget *Vedlegg/Filmer/Env_Sound_System_Concept_Walkthrough_W_Music.MP4,* med tidsstempel 02:33 – 03:37.

4.3.3. Okklusjon

Som nevnt i kapittel 3.3 beskrives okklusjon ofte gjennom fenomenet akustisk overføring (acoustic transmission). Det forklarer i korte trekk hvordan lyd går gjennom objekter og overføres til et annet materiale (se kapittel 2.3.3). Dette er ofte tilfellet når det gjelder lydutbredelse fra et rom til et annet, eller gjennom vegger. I denne sammenhengen vil okklusjon også da defineres som når *både direktelyden og de reflekterte lydene blir blokkert*. Dette fenomenet oppleves som en filtrering av høyfrekvenser og demping av direktelyden og de reflekterte lydene. I likhet med obstruksjonskonseptet, brukes det ofte en raycast – basert løsning i forbindelse med beregning av okklusjon. Her kreves det derimot en metode for å registrere flere treff langs raycast - strålen (Multi Raycast), slik at lydutbredelsesveier kan beregnes for flere vegger eller rom. Det bør også være mulig å kunne registre hvilke materialer strålen treffer, for å vite noe om de *lydblokkerende* egenskapene veggene eller rommene har. I utgangspunktet vil disse egenskapene omhandle hvor godt lyd absorberes, reflekteres, eller overføres. Den oppfattede opplevelsen vil være at lyden blir blokkert, derav beskrivelsen lydblokkerende egenskaper. I likhet med obstruksjon er dette et

konsept som krever alternative metoder enn de som ble presentert i kapittel 3.3.2, for å kunne skape realistiske og gode resultater som ikke er for ressurskrevende.

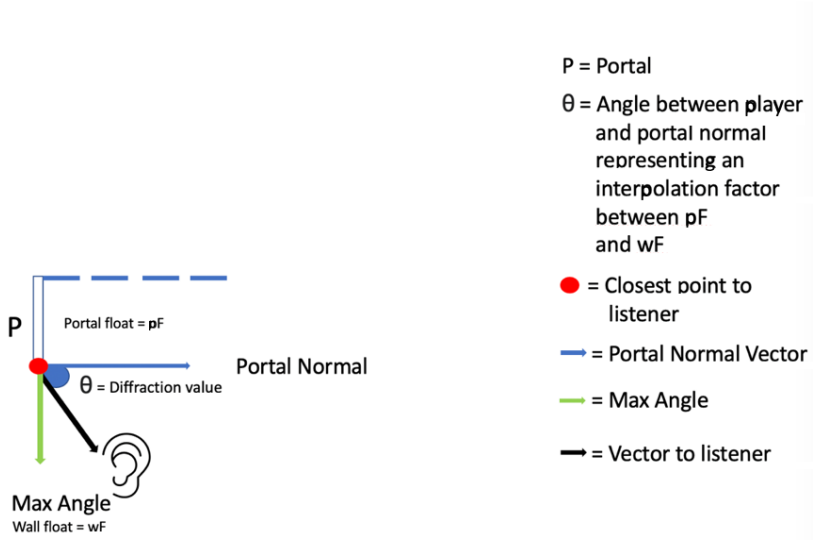
Før løsningen for dette aspektet ved undersystemet for lydutbredelse blir presentert, skal vi se på hvilke begrensninger som finnes i allerede eksisterende verktøy, som Wwise Spatial Audio. I likhet med obstruksjon, vil man i forbindelse med okklusjon også ha spesifikke kurver for parameter som demping av volum og filtrering av høyfrekvenser, *som er universale for alle lydobjekt i alle rom*. Dette vil være urealistisk ettersom rom, vegger eller materialer kan ha ulike egenskaper for lydabsorbsjon, refleksjon og generell isolasjon. I tillegg til dette vil Wwise sitt innebygde system *benytte seg av en egen form for raycasting*, som fortsatt kan være ressurskrevende i store mengder. Okklusjonsberegningene vil med andre ord være basert på «line of sight». I Wwise sitt system, kan man *ikke bestemme hvilke spillobjekt som har lydblokkerende egenskaper*. Dette kan føre til upålitelige resultater. I et online - forum ble det nevnt at det var tilfeller ved bruk av Wwise Spatial Audio, hvor okklusjon ble påført lydkilder som ble blokkert av ekstremt små objekter. Dette kunne være en hånd som plutselig stod i veien for spilleren og lydkilden (D. Crislip, personlig kommunikasjon, 16. oktober 2021). I andre tilfeller kunne det samme skje dersom for eksempel enorme skapninger var 90% synlige, men plasseringen til skapningens audio emitter tilsynelatende ikke var det (Ibid.).

I kapittel 2.3.3 (og 4.3.2) ble det nevnt at et aspekt ved diffraksjon var *spredningen av lydbølger i det de passerer en åpning* (i motsetning til hvordan de bøyer seg rundt objekter). I denne delen av oppgaven brukes dette aspektet ved diffraksjon som en del av okklusjon for å *simulere hvordan lyden spres fra åpninger i et rom*. Dette er blant annet med på å endre oppfattelsen av hvor en lyd kommer fra. I dette aspektet ved undersystemet for lydutbredelse bestemmer vi at hvert rom har en standard okklusjonsverdi, som igjen bestemmer hvor mye lyden blir absorbert av veggene i rommet. Dette gir oss muligheten til å tilegne ulike «absorpsjonsverdier» for rom ut ifra hvilke materialer de er bygd opp av. I tillegg til dette har hver portal en okklusjonsverdi for å bestemme hvordan lyden påvirkes hvis man står foran en åpning inn til rommet. Dette er illustrert i *figur 38*.

Det er flere grunner til at vi har definert en standard okklusjonsverdi for hvert og rom og ikke for hver vegg. Hvis okklusjonsverdien er definert for hver vegg, og spilleren beveger seg fra en vegg mot en annen en, vil dette føre til unaturlige hopp eller endringer i lyden. Dette kan negativt påvirke den immersive opplevelsen. En annen grunn er at ved å definere en standard okklusjonsverdi for hvert rom, får vi muligheten til å behandle alle lydkilder eller audio emittere i det aktuelle rommet med samme verdi. Dette vil ha en positiv effekt i forbindelse med ressursbruken.

Måten okklusjonsverdien beregnes på for et enkelt rom er først ved å finne nærmeste portal til spilleren, i det øyeblikket spilleren forlater rommet. Deretter vil nærmeste punkt til spilleren på denne portalen bli funnet. Det vil så bli gjort en sjekk for å finne spillerens vinkel til det nevnte punktet. Denne vinkelen tilsvarer en verdi som kalles en diffraksjonsverdi. Så lenge spillerens vinkel til punktet på portalen er null grader, vil den beregnede diffraksjonsverdien være 100% lik den bestemte *okklusjonsverdien til portalen*. Jo mer denne vinkelen øker, jo mer lik *okklusjonsverdien til rommet* vil den beregnede diffraksjonsverdien bli. Dette gjelder helt til man når en spesifikk vinkel hvor disse to (diffraksjonsverdien og rommets okklusjonsverdi) er helt like. Denne vinkelen kan selv bestemmes av lyddesigneren. På den måten kan ulike dører, vindu eller åpninger ha ulike punkt for maksimum absorpsjon. For ett enkelt rom, vil diffraksjonsverdien utgjøre den *totale okklusjonsverdien*. Vinkelen mellom spilleren og nærmeste punkt på portalen vil med andre ord bestemme en *total okklusjonsverdi som interpolerer mellom portal – okklusjon og rom – okklusjon*. Et eksempel på dette finnes i vedlegg *Vedlegg/Filmer/Env_Sound_System_Advanced_Occlusion_Demo.MP4,* med tidsstempel 00:00 – 01:00.

Disse verdiene vil bli sendt til Wwise og påvirke lyden i form av frekvensfiltrering (høyfrekvenser) og demping av volum, i henhold til ulike kurver for ulike lydobjekt i rommet som er definerte av lyddesigneren. Dette gjøres gjennom en RTPC. Okklusjonsverdier vil i det omgivelsesbaserte lydsystemet bestå av flyttall mellom 0 og 100.

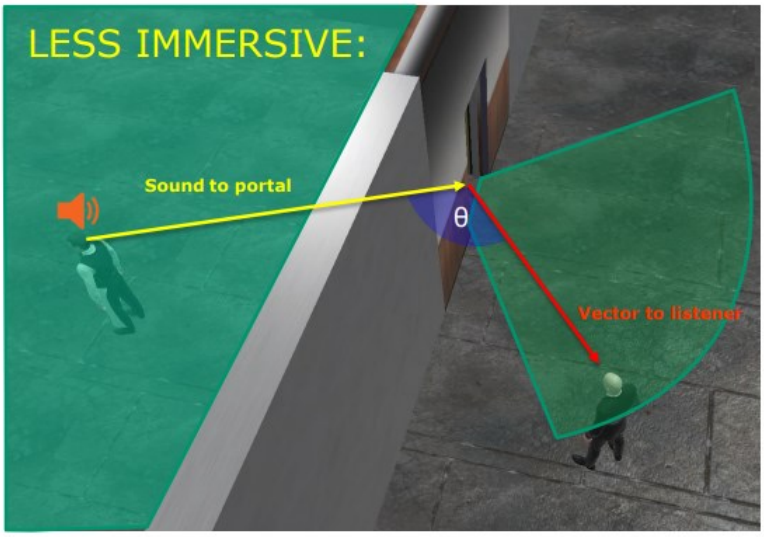


Figur 38: Egen illustrasjon av okklusjonsberegning for ett rom. wF vil være rom okklusjon, mens pF vil være portal okklusjon. Portal Normal beskriver punktet hvor vinkelen mellom spilleren og nærmest punkt på portalen er 0 grader. Når spilleren beveger seg mellom Portal Normal og Max Angle, vil θ interpolere mellom verdiene pF og wF. Interpoleringsfaktoren her vil utgjøre den totale beregnede okklusjonsverdien for ett enkelt rom.

Disse beregningene må skje kontinuerlig (mellom 10 – 15ms), ettersom okklusjon beskriver et dynamisk fenomen. Det er derfor lurt å finne en metode for å kunne begrense ressursbruken, da slike hyppige beregninger kan påvirke ytelsen til spillet. Måten det gjøres på er ved å definere at beregningene kun skal skje når spilleren er *utenfor rommet* og *innenfor en viss avstand til den nærmeste sammenkoblede portalen*. Denne avstanden kan selv defineres av lyddesigneren i portalens spillobjekt. Hvis det er snakk om flere sammenkoblede rom, vil beregningene *kun* skje dersom spilleren befinner seg i et av naborommene til det rommet hvor lyden kommer fra. Hvis spilleren befinner seg for eksempel to rom unna der hvor lydkilden er, vil ikke beregningsmetodene nevnt så langt brukes. Hvordan okklusjon for dette rommet beregnes vil bli forklart på side 68 i dette kapittelet.

Vanligvis (standard praksis) vil okklusjonsberegningene skje for hver audio emitter i det aktuelle rommet (gjøres blant annet i Wwise Spatial Audio). Det vil si at vinkelen og interpoleringsfaktoren beregnes mellom lydkilden og spilleren, istedenfor «portalåpningen» og lydkilden som gjøres i dette aspektet ved undersystemet for lydutbredelse. Da dette ble forsøkt under utviklingen av

Hitman, fant derimot lydteamet til IO Interactive ut at dette opplevdes mindre immersivt enn hvis beregningene ble gjort mellom portaløpningen og lydkilden (Boev, 2015).

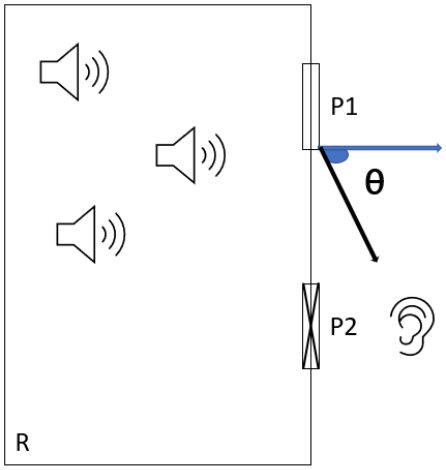


Figur 39: Illustrasjon av beregningsmetoden utprøvd under utviklingen av Hitman (Hentet april 14, 2022 fra presentasjon av Boev, 2015). Her vil okklusjonsberegningene skje ut ifra vinkelen mellom lydkilden og spilleren, istedenfor portalen og spilleren.

Det ble tidligere nevnt at vi beregner okklusjonsverdier for hele rom, og ikke enkle vegger. Hvis okklusjonsverdien er definert for hver vegg, og spilleren beveger seg fra en vegg mot en annen en, vil dette føre til unaturlige hopp eller endringer i lyden. Dette kan negativt påvirke den immersive opplevelsen. I undersystemet for lydutbredelse, har vi likevel muligheten til å gi vegger egne okklusjonsverdier dersom det ønskes. Måten dette kan gjøres på er ved å *plassere portaler på veggene* det er snakk om, og definere okklusjonsverdiene de skal ha. Beregningene som gjennomføres i henhold til metodene beskrevet i de tidligere avsnittene vil da sørge for at overgangene mellom veggene blir jevne og naturlige.

I det omgivelsesbaserte lydsystemet tas det også høyde for om en portal er definert som åpen eller lukket. Dette er relevant for situasjoner hvor spilleren for eksempel står nærme en portal som er definert som lukket, mens en annen portal litt lenger unna er definert som åpen. Da vil portalen som er lengst unna være den portalen som er mest gunstig å bruke, og lyden vil da oppfattes som

å komme fra den. Beregningene for okklusjon vil da skje ut ifra den portalen, selv om den andre kanskje er nærmere spilleren.



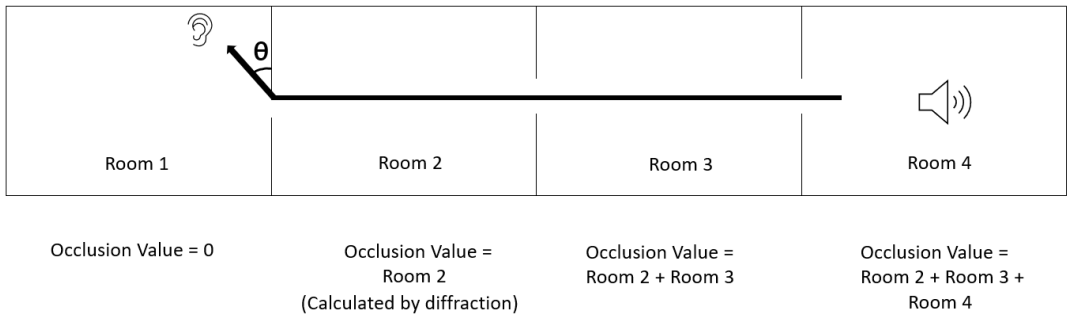
Figur 40: Egen illustrasjon av hvordan portalens tilstand endrer beregningsreglene. Her er P1 (Portal 1) åpen, mens P2 er lukket. Koden vil da beregne okklusjon fra P1 selv om P2 er nærmere.

Måten okklusjon beregnes på i det omgivelsesbaserte lydsystemet gir også gode resultater for den *oppfattede plasseringen av lyden*, spesielt sammen med den naturlige uttoningen av lydkildene som resultat av attenuasjonskurven til hver audio emitter. Måten disse samhandler på vil føre til at lyder for det meste oppfattes som å komme fra åpningen til rommet, selv om lydkilder ikke fysisk er plassert i døråpningen.

Beregningsmetoden for okklusjon i undersystemet for lydutbredelse tar så langt kun høyde for at spilleren befinner seg utenfor ett rom, og hvordan lyden endres fra åpninger til direkte blokader (veggene til rommet) gjennom diffraksjon. Det som enda ikke er redegjort for er hvordan lyden påvirkes av flere rom, eller hvordan okklusjon beregnes dersom lyden må gjennom flere lag med materialer. Dette vil bli forklart i større detalj i de neste avsnittene.

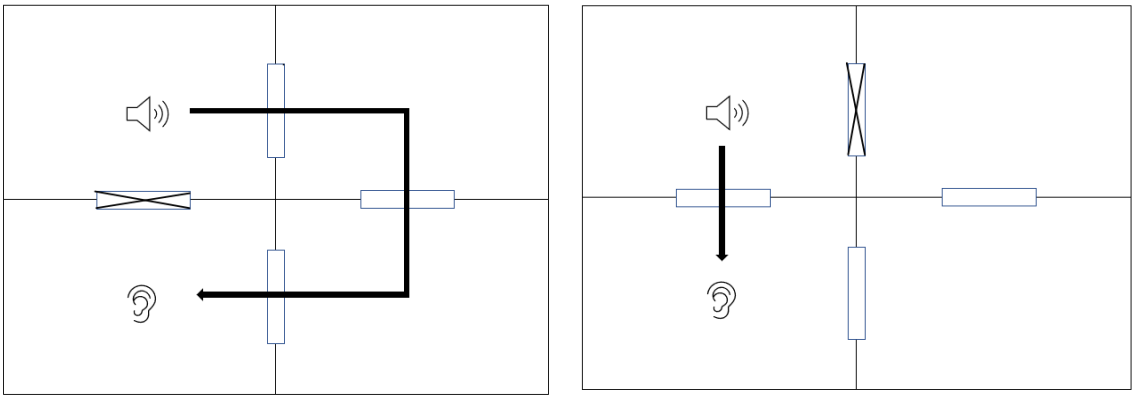
For å beregne større og mer komplekse lydutbredelsesveier er det flere metoder som kan brukes. Såkalt *pathfinding* – metoder brukes i spill, til for eksempel å finne mulige forflytningsveier for NPC'er (Ikke – spillbare karakterer) som er styrt av kunstig intelligens. Her kan det brukes ulike algoritmer som A* (Filion, 2018, s. 285), eller for eksempel Breadth - First Search (Boev, 2015). Funksjonaliteten til BFS (Breadth – First Search) algoritmen har blitt brukt som inspirasjon for beregningen av utbredelsesveier i det omgivelsesbaserte lydsystemet.

Prosessen for beregning av utbredelsesveier (i dette tilfellet okklusjon for flere rom) i denne oppgaven starter først og fremst med å *registrere spillerens plassering*. Dette skjer hver gang spilleren forlater et rom og *går inn i et nytt et*. Det vil da bli sendt en beskjed til tilhørende naborom om å *iverksette beregning av okklusjonsverdier* med metoden nevnt tidligere (spillerens vinkel i forhold til portalen). Dette kan ansees som hovedberegningene for okklusjon. Disse rommene vil da s*ende en beskjed til sine naborom om å addere sine rom – okklusjonsverdier* med verdien fra rommet som gjør hovedberegningene. Disse naborommene vil *sende dette resultatet videre til neste naborom, som igjen legger denne verdien sammen med sin egen rom - okklusjonsverdi*. Denne prosessen vil fortsette for alle naborom videre, *til man når et spesifikt antall rom*, som kan bestemmes av lyddesigneren. Dette gjøres for å unngå unødvendige beregninger på rom som er så langt unna at lydene derfra ikke kan høres. En slik prosess kalles for en *traversal pass.* En traversal pass vil skje kontinuerlig (hver 10 – 15 ms), og etter hver runde vil den beregnende okklusjonsverdien bli påført alle lydene i de aktuelle rommene. Eksempel på dette finnes i vedlegg *Vedlegg/Filmer/Env_Sound_System_Advanced_Occlusion_Demo.MP4,* med tidsstempel 01:04 – 01:57. Et annet eksempel finnes i vedlegg *Vedlegg/Filmer/Env_Sound_System_Concept_Walkthrough_W_Music.MP4,* med tidsstempel 07:45 – 10:50.



Figur 41: Egen illustrasjon av hvordan okklusjon beregnes for flere rom.

I prosessen for beregningen av okklusjon for flere rom er det også definert et sett med regler for hvordan beregningene skal foregå, og hva som må til for at et rom skal inkluderes i disse beregningene. Tidligere i oppgaven ble det nevnt at okklusjonsverdien er et flyttall mellom 0 – 100. Tallet 100 representerer derfor fullstendig okklusjon og *kan ikke overskrides*. Hvis et rom får okklusjonsverdi over 100 som et resultat av sammenlagte verdier for naborommene, *vil tallet overstyres og settes til 100*. I tillegg til dette beregner vi okklusjonsverdi på et rom *kun dersom okklusjonsverdien ikke allerede er 100*. Enda et kriterium er at rommene og portalene som skal beregnes *må være åpne*, og ikke definert som lukkede (dette gjøres av lyddesigneren i systemet). Dette betyr at utbredelsesveiene kan endre seg ut ifra om en portal er lukket eller åpen.



Figur 42: Egne illustrasjoner som viser hvordan lydutbredelsesveiene endres ut ifra hvilken tilstand portalene (og rommene) har. De lille blå boksene er portaler, mens de store svarte boksene er rom. Kryss over portal betyr at den er lukket.

4.3.4. Romklang

I kapittel 3.3.3 ble det nevnt at en annen praksis i forbindelse med lydutbredelse i spill er hvordan man simuler lydrefleksjoner (romklang) i et gitt miljø eller sett med omgivelser. Det ble også gitt eksempler på hvordan dette kunne gjøres. Romklang - aspektet ved undersystemet for lydutbredelse ble utelatt av det leverte praktiske arbeidet som et resultat av tidsbegrensning. Selv om det likevel ikke er inkludert i det *leverte praktiske arbeidet*, betyr det ikke at det fullstendig utelates fra oppgaven. Dette aspektet ved undersystemet for lydutbredelse har vært under utvikling lenge og er ferdig utviklet fra et teoretisk perspektiv og standpunkt. Implementasjonstanken og fremgangsmåten er illustrert i flytskjemaet som kan sees i *figur 45*. Det eneste som mangler er med andre ord å programmere det i spillmotoren. Ettersom dette aspektet ved lydutbredelse er ferdig planlagt og teoretisk utviklet, vil det derfor være naturlig å beskrive konseptet i denne delen av oppgaven.

Det må nevnes at alle romklang – effekter som er hørbare i de vedlagte filene (spill, prosjektfiler og filmklipp) er kun ment som eksempler, og lagt på i henhold til standard praksis som nevnt i kapittel 3.3.3. Disse er lagt på for å gi en ide om hvordan dette aspektet ved undersystemet kan *høres ut* og *oppleves* i en helhetlig sammenheng. De er med andre ord ikke resultater av metoden presentert i dette kapittelet.

I dette aspektet ved undersystemet for lydutbredelse vil det forsøkes å skape mer realistisk romklang, enn ved å kun definere én romklangtype med forhåndsbestemte statiske verdier for et enkelt område. Her vil verdiene til romklangen automatiseres basert på rommets størrelse og spillerens plassering i det. Konseptet vil likevel være regionsbasert, da det skal være tilknyttet romobjektene (se kapittel 4.3.1). I det omgivelsesbaserte lydsystemet prøver vi å kontrollere tre konsept innenfor romakustikk og generell romklang. Disse er etterklangstid (RT), Rom Moder og Pre – Delay.

I dette aspektet ved undersystemet for lydutbredelse trenger lyddesigneren kun å legge inn informasjon om hvilke materialer rommet består av. Dette vil si at det må legges inn

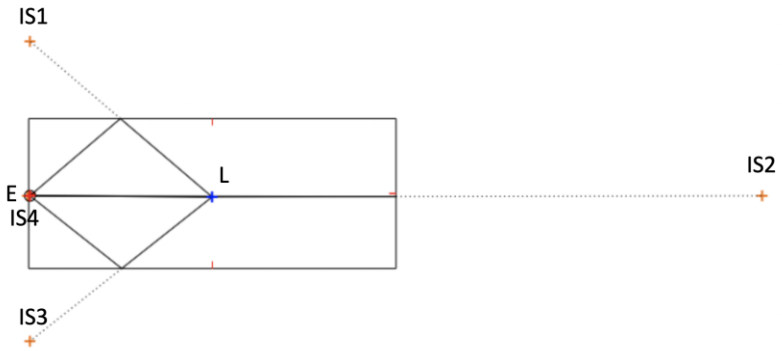
absorpsjonskoeffisienter for alle vegg, tak og gulv i rommet. Ettersom dette skal være knyttet til romobjektene, vil vi allerede ha informasjon om størrelsen og volumet til det aktuelle rommet. Ut ifra denne informasjonen *vil spillmotoren regne ut etterklangstiden for ulike frekvensbånd* (gjennom *Sabines* ligning fra kapittel 2.3.3.). Dette vil brukes til å *bestemme en gjennomsnittstid for RT – tider, for høyfrekvenser og lavfrekvenser*. Denne tiden vil gjennom en RTPC bli stilt inn i en romklangeffekt, i for eksempel en mellomvare. Målene for rommet vil deretter bli brukt til å *beregne et sett med axiale rom moder* (lyddesigner kan selv bestemme antall), mellom 100 og 200 Hz (ved bruk av formelen for å finne rom moder presentert i kapittel 2.3.3), da disse vil være mer hørbare. Man kan da bestemme om man vil bruke en spesifikk frekvens, eller flere (avhengig av hva romklangeffekten har mulighet for). I sanntid vil det da kontinuerlig bli *gjennomført prosesser som lokaliserer spillerens posisjon* i forhold til veggene i rommet. Ut ifra dette vil da skje en parameterendring som øker eller senker nivået på de beregnede lavfrekvensene ut ifra hvor nært veggen spilleren er; jo nærmere veggen, jo høyere nivå.

Et av de store problemene man ofte møter på i forbindelse med beregninger av tidlige refleksjoner er at dette ikke er en verdi som er lik for alle lydkilder i et rom. Dette er en dynamisk verdi som endres ut ifra faktorer som for eksempel romstørrelse, og lytterens posisjon i forhold til en spesifikk lydkilde. *Pre – delay tiden* (tiden det tar for de første refleksjonene å treffe lytteren etter direktelyden), er med andre ord *ulik for hver lydkilde i et rom*. Dette kan føre til utfordringer ettersom de fleste romklangeffekter kun har en mulighet for pre – delay innstilling per instans av effekten. På grunn av dette må man representere romklangen i et rom med en innstilling for pre – delay, som brukes for alle lydkildene. Det mest realistiske hadde selvfølgelig vært å kontinuerlig beregne pre – delay tid basert på rommets størrelse og avstand mellom spilleren og hver enkelt lydkilde i rommet. Dette hadde derimot vært en ressurskrevende prosess og derfor lite hensiktsmessig.

I dette aspektet ved undersystemet for lydutbredelse er tanken derfor å søke en mer optimaliserende løsning i forhold til ressursbruk, men som likevel gir et godt resultat. I denne løsningen vil *rommets senterpunkt og ytre grenser først registreres*. *Midten av rommet vil ansees som referansepunktet*

for hvor de tidligere refleksjonene skal ende opp. Vi kan se for oss at en fiktiv lytter er plassert her. Videre *tas det utgangspunkt i at lydkilder befinner seg på kanten av hver side i rommet*.

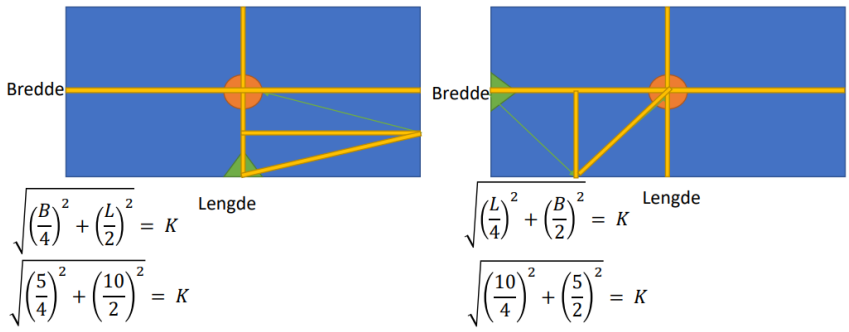
Ved å bruke en metode som kalles for *Image Source* (Savioja, 216), kan vi *finne veiene de tidlige refleksjonene må ta for å nå lytteren*. En *Image Source* er i korte trekk et speilbilde av en audio emitters plassering bak en reflekterende overflate (Keklikian, 2017). I denne oppgaven blir dette begrepet kun omtalt som *speilbilde.* Hvis man trekker en linje mellom en audio emitters speilbilde og lytteren, vil linjen krysse den reflekterende overflaten i det punktet hvor lyden reflekteres (Ibid.). Dette gjelder for alle speilbildene til audio emitteren. Dette er illustrert i *figur 43*. I denne oppgaven er vi kun interesserte i 1 – ordens refleksjoner. Dette innebærer kun de første refleksjonspunktene for lyden.



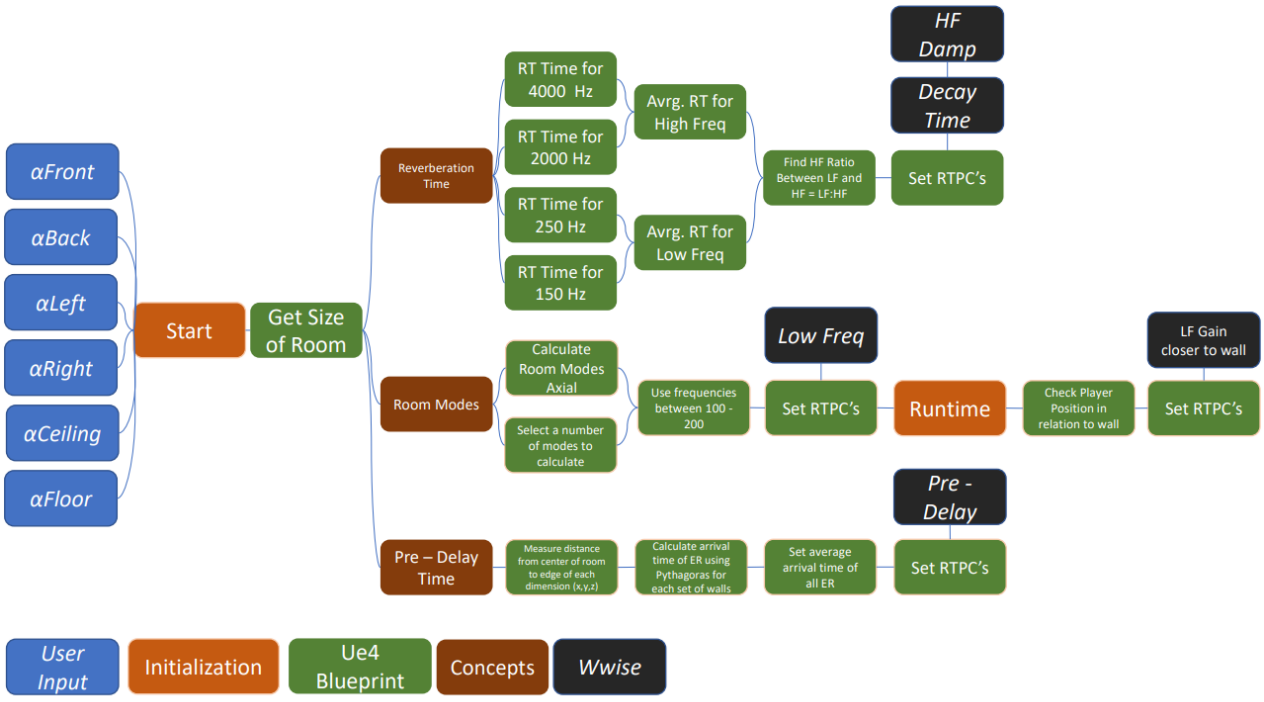
Figur 43: Egen illustrasjon av hvordan image source (speilbilde) metoden kan brukes til å finne veien til 1 – ordens lydrefleksjoner. L er lytterens plassering, E er plasseringen til audio emitteren. Plusstegnene med benevningene IS over seg er speilbildene til audio emitteren E.

Resultatene man får fra image source - metoden, kan brukes til å *finne lengdene til de ulike refleksjonsveiene* ved bruk av Pytagoras' læresetning. Når disse er funnet, kan man *regne ut tiden det tar for lyden å reise disse lengden*e, og bruke et gjennomsnitt for vegg og tak for å finne en estimert pre – delay tid for rommet (Se *figur 44* for forklaring). Disse beregningene vil bli gjort

for et rom før banen starter, og stiller inn pre - delay tiden til en romklangeffekt i for eksempel Wwise. Dette kan gjøres gjennom en RTPC.



*Figur 44: Egen representasjon av beregningene for å finne avstanden lyden må reise, som er K*2. Deretter deles avstanden (K*2) på lydens hastighet for å få tiden det tar for refleksjonene å treffe lytteren. Vi kan deretter bruke et gjennomsnitt av vegg og tak for å sette en Pre – Delay tid for rommet.*



Figur 45: Eget flytskjema for implementasjonen av romklang i undersystemet for lydutbredelse.

5. Diskusjon/Evaluering

5.1. Oppsummering

I denne oppgaven har jeg redegjort for hvordan det omgivelsesbaserte lydsystemet har blitt utviklet, med bakgrunn i presenterte teorier, praksiser og konvensjoner for lyd i spillbransjen. I det teoretiske aspektet som har blitt presentert, har det blitt redegjort for relevant *programvare* som er viktig i forhold til spillutvikling (*spillmotor*) og lydimplementasjon i spill (*mellomvare*). Teori og beskrivelse av begrepet *immersjon* har blitt gjennomgått for å forklare *følelsen av å være til stede i en fiksjonell verden gjennom stimuli av sansene, og oppfatte innholdet som ekte*. I tillegg til dette har det blitt forklart ulike begrep innen *spillutvikling*, *begrep direkte rettet mot lyd i spill* og *lydbegrep innenfor akustikk og fysikk*. Det teoretiske aspektet har blitt presentert for å gi en grunnforståelse av innholdet i denne oppgaven, slik at man bedre kunne forstå de generelle lydpraksisene og konvensjonene i tredimensjonale spill, og valg og begrunnelser som er tatt i forbindelse med utviklingen av det omgivelsesbaserte lydsystemet.

Det har blitt redegjort for *generelle lydpraksiser og lydkonvensjoner i tredimensjonale spill*. Her ble det presentert konsept som omhandler *tredimensjonale lydkilder*, *atmosfæresystem (Todimensjonale* og *tredimensjonale)* og *lydutbredelsessystem (obstruksjon, okklusjon, romklang og lydutbredelsesveier).* For hver av disse konseptene har det blitt presentert konvensjonelle løsninger, metoder og fremgangsmåter for å implementere dem i dataspill. Disse har blitt gjennomgått for å gi en bedre forståelse av valgene som er gjort i utviklingen av det omgivelsesbaserte lydsystemet.

I kapittel 4 har vi sett på hvordan det omgivelsesbaserte lydsystemet har blitt utviklet. Det har også blitt vist hvordan konvensjonene og allerede eksisterende verktøy har gitt grunnlag for nye håndteringsmetoder og implementasjonsløsninger av ulike konsept som *statiske tredimensjonale audio emittere, atmosfærelyder* og *lydutbredelse*.

I henhold til målsettingene nevnt i kapittel 1.3 vil lydsystemet presentert i denne oppgaven gjøre det mulig å *enklere integrere lyd i en programvare.* Den gir også større mulighet for å kunne *frigjøre seg fra det mer tekniske aspektet* ved lyddesign for tredimensjonale spill, ved å ta i bruk allerede programmerte regler og funksjoner, slik at det blir mer *tid til det kreative aspektet ved lyddesign*. Det omgivelsesbaserte lydsystemet kan også fungere som et *verktøy for allerede erfarne mennesker i bransjen*, ettersom det inneholder løsninger som kan bygges videre på, endres og utvikles ut ifra hva behovet er. Det er også et intuitivt system som er enkelt å bruke, slik at de med mindre erfaring innen spillutvikling eller de tekniske aspektene enklere kan ta det i bruk for å lage det innholdet de ønsker. Det kan derfor også brukes som et *læringsverktøy* for alle som ønsker å bli kjent med praktisk bruk av lyd i dataspill.

Det omgivelsesbaserte lydsystemet vil også kunne fungere i en helhetlig sammenheng for spillutvikling, da alle komponentene jobber sammen. I tillegg har det vært fokusert på å gjøre systemet prosesseringsvennlig og sparsomt i forhold til ressursbruken (der det er mulig). Dette gjør det mer aktuelt å bruke i forbindelse med større og mer profesjonelle prosjekt, hvor dette er viktig. Det er også kompatibelt med en rekke mellomvarer som kan brukes i sammenheng med Unreal Engine 4 dersom det ønskes, da koden i seg selv gjøres i spillmotoren, og endringene som blir gjort i mellomvarene består av parameterendringer og håndtering av events. På den måten er det omgivelsesbaserte lydsystemet *ikke begrenset til kun en programvare*.

5.2. Evaluering

I evalueringen av dette arbeidet er det noen momenter å trekke frem. I starten av 2021, begynte jeg å jobbe i spillstudioet Pineleaf Studio som lyddesigner for spillet *Dwarfheim* (Pineleaf Studio, 2021), som praktikant. Den opprinnelige planen var at arbeidet som ble gjort der skulle brukes i direkte tilknytning til masteroppgaven. Underveis i det seks - måneders forløpet, ble det klart at av ulike årsaker, lot ikke dette seg gjøre. Selv om det arbeidet som ble gjort der ikke kunne brukes i direkte tilknytning til oppgaven, har det likevel vært en stor ressurs for den tilegnede kunnskapen angående spill. Dette på alle måter; jeg lærte mye om lyd, programmering, praksiser og bransjen generelt. Det er en opplevelse jeg ikke ville være foruten. Likevel, førte dette til at jeg behøvde en

ny problemstilling. Problemstillingen som da ble utviklet er den som har blitt presentert i denne oppgaven. Dette var riktignok ikke et problem med tanke på tidsbegrensningen, spesielt ettersom tiden som praktikant ble brukt til å tilegne meg mye relevant kunnskap, som gjorde denne oppgaven mulig å gjennomføre.

Ettersom denne oppgaven krever mye kunnskap om programmering, ble det i starten planlagt å få hjelp av en bekjent som er utdannet spillutvikler og har god kjennskap til Unreal Engine 4. Dette fungerte godt i starten, men etter hvert ble det tydelig at ettersom begge hadde mye å jobbe med var ikke denne hjelpen tilgjengelig nok til å kunne ansees som stabil. Som et resultat av dette, måtte all form for videre programmeringskunnskaper tilegnes på egen hånd. I ettertid er dette noe som har vist seg å være mer positivt enn negativt, da det har ført til en forståelse rundt programmeringsfaget som gjør at jeg nå føler meg trygg nok til å kunne drive med komplisert utvikling i Unreal Engine 4.

5.2.1. Utbedringer og videre arbeid

Som et resultat av tidsbegrensning, er det noen ting i denne oppgaveløsningen som ikke har hatt mulighet til å bli utbedret eller utviklet. I denne delen av oppgaven, skal det derfor bli redegjort for hva som kan forbedres i forhold til hva som allerede er skapt, og hva som ønskes å legges til det omgivelsesbaserte lydsystemet. Noen av disse aspektene ble klart tidlig i oppgaveløsningen, mens andre har blitt til etter flere runder med testing.

En av de første og viktigste aspektene når det gjelder forbedring av systemet er i forhold til prosesseringsbruken og optimalisering. Selv om det er lagt inn mye tanke og arbeid i å unngå unødvendig ressursbruk, vil dette alltid være et punkt som kan forbedres. Optimalisering av systemet kan gjøres på flere nivå. Som nevnt tidligere i oppgaven er hele systemet laget gjennom visuell programmering i UE4, i noe som kalles for Blueprints. Fordelen med dette er at det ikke krever stor kunnskap om direkte programmering (selv om logikken er lik) for å kunne jobbe med spillutvikling. Problemet med det, er at det ikke er like optimalisert som ren kode og er mer ressurskrevende. For å kunne optimalisere dette systemet, vil det derfor være hensiktsmessig å

konvertere det til C++ kode. I tillegg til dette kan det være noen innebygde funksjoner og verktøy, eller programmeringskonvensjoner i UE4 som kan erstatte en del av koden som jeg har laget, for å utføre de samme prosessene. Med mer kunnskap om hvilke funksjoner og verktøy som krever mer ressurser enn andre, kan dette bidra til å optimalisere systemet enda mer. Det er også ønskelig å optimalisere den generelle kodestrukturen og oppsettet.

Etter større stresstester på undersystemet for lydutbredelse, og spesifikt okklusjon, ble det oppdaget at beregningskostnadene for okklusjon i forbindelse med sammenkoblede rom og lydutbredelsesveier var større enn antatt. Dette er en del av det omgivelsesbaserte lydsystemet som da burde utbedres. Som et resultat av dette viste det seg at å påføre verdiendringene på alle audio emitterene i sanntid, med den hyppige frekvensen som var satt, kunne føre til nedsatte målinger på antall bilder i sekundet. En løsning for dette ble teoretisert for videre arbeid; ettersom vi allerede behandler alle audio emittere for et rom likt med tanke på lydutbredelse, kan en løsning være å sende alle lyder som befinner seg i et rom til en form for «bus». Istedenfor å påføre verdiendringene for okklusjon på alle individuelle audio emittere, kan man da påføre disse endringene på rom – busen, som vil redusere antall prosesser som kjøres. Enda et fokuspunkt for videre arbeid i undersystemet for lydutbredelse vil være implementasjonen av romklang - konseptet, presentert i kapittel 4.3.4.

I kapittel 4.3.3 ble det nevnt at måten okklusjon beregnes på i undersystemet for lydutbredelse gir gode resultat for den oppfattede plasseringen av lydkilden. Det er likevel noen steg som kunne ha blitt tatt for å gjøre den oppfattede plasseringen til lyden enda bedre. Det første aspektet her er at lydutbredelsen i form av okklusjon egentlig også burde påvirke attenuasjonskurven til lydkildene. Ettersom lyden må kunne reise lenger for å nå lytteren, vil attenuasjonskurven som da er aktuell være den som går fra lydkilden til portalen, og deretter til lytteren, og ikke fra lydkilden direkte til lytteren. I tillegg kan det være en god ide å enten flytte audio emitteren eller plassere en slags virtuell emitter for en lydkilde i portalen eller i nærhet av den, slik at lyden virker som den faktisk kommer derfra. Disse løsningene blir gjort i Wwise Spatial Audio, og gir gode resultater. Dette kan enkelt gjøres i for eksempel Wwise ved å bruke en funksjon som heter «multiple positions», som gjør at en lydkilde kan representeres fra forskjellige posisjoner (Audiokinetic, u.å.b).

5.2.2. Tilgjengelighet og bruksområder

Det er ønskelig å kunne publisere det omgivelsesbaserte lydsystemet og gjøre det tilgjengelig for alle som ønsker å bruke det. Tanken er at det skal være helt gratis og alle deler av det skal tilgjengeliggjøres med kildekode og kommentarer på hva koden gjør. I tillegg til dette skal alle kodedelene som er spesifikke for Wwise kommenteres og tydeliggjøres slik at de kan bli erstattet med kode for andre mellomvarer eller for Unreal Engine sin egen lydmotor. Alle som tar i bruk dette systemet kan dermed endre det, modifisere det, lære av det og bruke det som de vil. Det er ikke planlagt å opprette en spesifikk vedlikeholdstjeneste for systemet, da det ikke er ressurser til det. Dette betyr at etter det er publisert og offentliggjort, vil det ikke bli gjort vedlikehold for å sørge for at det er kompatibelt med nye oppdateringer av programvarene. Før prosjektet blir delt vil det derimot gjøres kompatibelt med de nyeste versjonene av Unreal Engine (Unreal Engine 5), og Wwise.

Det ble også gjort en del funn i denne oppgaven i forhold til hvordan det omgivelsesbaserte lydsystemet kan være aktuelt i flere bruksområder enn ren spillutvikling. Et av de mest aktuelle for dette kan være VR (Virtual Reality). Dette er felt som stadig vokser, og spesielt programvarer som Unreal Engine har dette som store satsingsområder for fremtiden. I tillegg til dette har Unreal Engine også blitt brukt i sammenheng med film og tv. Et av de nyere eksemplene på dette kan være *The Mandalorian* (Favreau, 2019) og *The Book of Boba Fett* (Favreau, 2021). I slike sammenhenger kan det omgivelsesbaserte lydsystemet eventuelt være aktuelt.

Systemet presentert i denne oppgaven kan også brukes i forbindelse med animasjonsfilm. Unreal Engine 4 har blant annet blitt brukt til å produsere animasjonsfilmen *Allahyar and the Legend of Markhor* (Khan, 2018). I en slik produksjon kan dette systemet bidra til å gjøre lydverdenen mer dynamisk og erstatte en del av arbeidet som vanligvis gjøres i postproduksjonen. Om ikke den erstatter arbeidet, kan det for eksempel brukes til eventuelle preproduksjoner av scener i animasjonsfilmer.

6. Litteraturliste

Anderson, L. (2018). *Film Sound Design.* Oxford Bibliographies. https://www.oxfordbibliographies.com/view/document/obo-9780199791286/obo-9780199791286-0168.xml

Andrade, A. (2015). Game engines: a survey. *EAI Endorsed Transactions on Serous Games, 2*(6), 1 – 6. https://doi.org/10.4108/eai.5-11-2015.150615

Apple (u.å.). *Augmented Reality - More to Rxplore with ARKit 5.* Apple Developer. https://developer.apple.com/augmented-reality/arkit/

Audiokinetic (u.å.a). *Applying Distance – Based Attenuation.* Audiokinetic. https://www.audiokinetic.com/library/edge/?source=Help&id=applying_distance_based_attenuation

Audiokinetic (u.å.b). *Creating Multple Positions for a Single Game Object.* Audiokinetic. https://www.audiokinetic.com/library/2017.1.9_6501/?source=Help&id=creating_multiple_positions_for_single_game_object

Audiokinetic (u.å.c). *Real – Time Parameter Control (RTPC).* Audiokinetic. https://www.audiokinetic.com/library/2017.1.9_6501/?source=WwiseProjectAdventure&id=real_time_parameter_control_rtpc

Audiokinetic (u.å.d). *Rooms and Portals.* Audiokinetic. https://www.audiokinetic.com/library/edge/?source=SDK&id=using_rooms_and_portals.html

Audiokinetic (u.å.e). *Understanding Events*. Audiokinetic. https://www.audiokinetic.com/library/2018.1.11_6987/?source=WwiseFundamentalApproach&id=understanding_events_understanding_events

Audiokinetic (u.å.f). *Unreal Objects.* Audiokinetic. https://www.audiokinetic.com/library/edge/?source=UE4&id=features_objects.html

Audiokinetic (u.å.g). *Voice Starvation.* Audiokinetic. https://www.audiokinetic.com/library/edge/?source=Help&id=ErrorCode_VoiceStarving

Audiokinetic (u.å.h). *Wwise Spatial Audio.* Audiokinetic. https://www.audiokinetic.com/products/wwise-spatial-audio/

Balaguer – Ballester, E., He, X., Michailidis, L. (2018). Flow and Immersion in Video Games: The Aftermath of a Conceptual Challenge. *Frontiers in psychology, 18*(9), 1682. https://doi.org/10.3389/fpsyg.2018.01682

Boards To Bits Games. (2018, 03. august). *Systemic Game Design, Part 1: What Are Systems?* [Video]. YouTube. https://www.youtube.com/watch?v=NZc7yGdahkY

Boev, S. (2019, 15. juli). *Hitman 2*. Enhancing Reverb on Modern CPUs.* Intel. https://www.intel.com/content/www/us/en/developer/articles/case-study/hitman-2-enhancing-reverb-on-modern-cpus.html

Boev, S. (2015, 3 – 4. august). *Sound Propagation in Hitman.* GDC Europe 2015. Cologne, Germany. https://www.gdcvault.com/play/1022774/Sound-Propagation-in

Bridgett, R. (2013, 2. oktober). *Why ambient sound matters to your game.* Game Developer. https://www.gamedeveloper.com/design/why-ambient-sound-matters-to-your-game

Brooks, N. (2021, 28. januar). *Immersion is a KEY Element of Gaming – Here's Why.* CBR. https://www.cbr.com/immersion-gaming/

Buckley, D. (2021, 21. juli). *Unity vs. Unreal – Choosing a Game Engine.* GameDev Academy. https://gamedevacademy.org/unity-vs-unreal/

Buffoni, L.X. (2020, 6. august). *A Wwise Approach to Spatial Audio – Part 2 – Diffraction.* Audiokinetic. https://blog.audiokinetic.com/a-wwise-approach-to-spatial-audio-part-2-diffraction/

Buhler, J., Deemer, R. & Neumeyer, D. (2010). *Hearing the Movies: Music and Sound in Film History.* Oxford University Press.

Chattopadhyay, B. (2017). Reconstructing atmospheres: Ambient Sound in Film and Media Production. *Communication and the Public,* 2(4), 352 – 364. https://doi.org/10.1177/2057047317742171

Cheng, E.M., Chuah, H.G., Lam, C.K. & Tan, W.H. (2017). Sound Transmission Loss of Natural Fiber Panel. *International Journal of Mechanical & Mechatronics Engineering IJMME – IJENS.*16(06). 33 – 42.

Chion, M. (1994). *Audio – Vision: Sound on Screen.* Columbia University Press.

Christopoulou, E. & Xinogalos, S. (2017). Overview and Comparative Analysis of Game Engines for Desktop and Mobile Devices. *International Journal of Serous Games, 4*(4), 21 – 36. https://doi.org/10.17083/ijsg.v4i4.194

CryEngine (u.å.). *Culling Explained.* CryEngine. https://docs.cryengine.com/display/SDKDOC4/Culling+Explained

Dirrenberger, M. (2018). Designs for Ambiences in Open – World Games. I G. Somberg (Red.), *Game Audio Programming 2 – Principles and Practices* (s. 183 – 193). Taylor Francis Group.

Elnoshokaty, I. (u.å). *Basics of Sound and Hearing: Part 2 Sound Propagation*. Prof. Ibrahim Elnoshokaty (Egenutgitt bok).

Epic Games. (u.å.a). *Ambient Sound Actor User Guide*. Epic Games. https://docs.unrealengine.com/4.27/en-US/WorkingWithAudio/SoundActors/

Epic Games. (u.å.b). *Unreal Engine: The most powerful real-time 3D creation tool*. Epic Games. https://www.unrealengine.com/en-US/

Epic Games. (u.å.c). *Using a Multi Line Trace (Raycast) by Channel*. Epic Games. https://docs.unrealengine.com/4.27/en-US/InteractiveExperiences/Tracing/HowTo/MultiLineTraceByChannel/

Ermi, L. & Mäyrä, F. (2005, 16. – 20. juni). *Fundamental Components of the Gameplay Experience: Analysing Immersion*. [Paperpresentasjon] Digital Games Research Conference 2005, Changing Views: Worlds in Play. Vancouver.

Filion, M. (2018). Obstruction, Occlusion, and Propagation. I G. Somberg (Red.), *Game Audio Programming 2 – Principles and Practices* (s. 279 – 288). Taylor Francis Group.

Firelight Technologies. (u.å.). *FMOD Studio: The adaptive audio solution for game*. Fmod Studio. https://www.fmod.com/studio

GameMaker (u.å.). *The Ultimate 2D Game Development Environment.* GameMaker. https://gamemaker.io/en/gamemaker

Gjerdsjø, G.D. (2014). *Abstract Concrète – Transdiegetiske lydobjekter som estetiske og narrative virkemidler i filmmusikk.* [Masteroppgave]. Norges teknisk - naturvitenskapelige universitet.

Harris, N. (2018). Practical Approaches to Virtual Acoustics. I G. Somberg (Red.), *Game Audio Programming 2 – Principles and Practices* (s. 289 – 306). Taylor Francis Group.

Henderson, T. (u.å.). *Reflection, Refraction and Diffraction.* The Physics Classroom. https://www.physicsclassroom.com/class/sound/Lesson-3/Reflection,-Refraction,-and-Diffraction

Jacobsen, B. [Cujo Sound]. (2018). *Okklusjon and gates in AAA game sound design. By Cujo Sound.* Youtube. https://www.youtube.com/watch?v=mu2vDP74d5U

Keklikian, T. (2017, 14. november). *Image Source Approach to Dynamic Early Reflections.* Audiokinetic. https://blog.audiokinetic.com/image-source-approach-to-dynamic-early-reflections/

Lakka, E., Malamos, A.G., Pavlakis, K.G. & Ware, J.A. (2018). Spatial Sound Rendering – A Survey. *IJIMAI 2018 – Regular Issue – Vol 5 Issue 3 – 10th Anniversary*. 5(3). 33 – 45. http://doi.org/10.9781/ijimai.2018.06.001

Madigan, J. (2010, 27. juli). *The phsycology of Immersion in Video Games.* The psychology of video games – Examining the intersection of psychology and video games. https://www.psychologyofgames.com/2010/07/the-psychology-of-immersion-in-video-games/

Nygård, E. (2016). *Omgivelseslyd Som Ledemotiv.* [Masteroppgave]. Norges teknisk - naturvitenskapelige universitet.

Ohmori, H., Yamada, R. (2010). Frequency Dependent Specialization for Processing Binaural Auditory Cues in Avian Sound Localization Circuits. I P. Strumillo (Red.), *Advances in Sound Localization (s. 513 – 526).* IntechOpen.

Paul, L. J. (2015). *Game Audio Middleware.* Video Game Audio. https://videogameaudio.com/FullIndie-Apr2015/GameAudioMiddleware-FullIndie-SchoolOfVideoGameAudio-LPaul-Apr2015.pdf

Raybould, D. & Stevens, R. (2015). *Game Audio Implementation – A Practical Guide Using the Unreal Engine.* Focal Press.

Resonance Audio. (u. å.). *Fundamental Concept*. Resonance Audio. https://resonance-audio.github.io/resonance-audio/discover/concepts.html

Rossings, D.T. (2002). *The Science of Sound.* San Fransisco. Addison – Wesley Publishing Company.

Savioja, L. (2016). *Image – source Method in a Rectangular Room*. Interactive Acoustics. http://interactiveacoustics.info/html/GA_IS_isRectangular.html

Schmidt, B. (2020, 29. juli). *Game Audio Job Skills – How to Get Hired as a Game Sound Designer – Game Sound Design Skills: An Analysis of 100 Game Sound Job Postings.*

GameSoundCon. https://www.gamesoundcon.com/post/game-audio-job-skills-how-to-get-hired-as-a-game-sound-designer

Sinclair, J.L. (2020). *Principles of Game Audio and Sound Design – Sound Design and Audio Implementation for Interactive and Immersive Media.* New York. Taylor and Francis Group.

Soundnation. (2019, 10. september). *One – Shots: Small Sound, Big Impact*. Soundnation. https://soundation.com/station/2019/09/10/one-shots-small-sound-big-impact/

Steam Audio. (u. å.). A Benchmark in Immersive Audio Solutions for Games and VR. Valve Software. https://valvesoftware.github.io/steam-audio/#learn-more

Unity. (u.å.). *Unity.* Unity. https://unity.com/

7. Spill

Tittel	**(Utgiver, årstall)**
Dwarfheim	(Pineleaf Studio, 2021)
Hitman	(IO Interactive, 2016)
MediEvil	(Sony Interactive Entertainment, 1998)
The Witcher 3: Wild Hunt	(CD Projekt Red, 2015)

8. Filmer/Serier

Favreau, J. (Skaper). (2019). *The Book of Boba Fett* [TV Serie]. Lucasfilm.

Favreau, J. (Skaper). (2019). *The Mandalorian* [TV Serie]. Lucasfilm.

Khan, U.Z. (Regissør). (2018) *Allahyar and the Legend of Markhor.* [Animasjonsfilm]. 3rd World Studio.

9. Vedlegg

9.1. Bildefiler

Vedlegg/Bilder/Ambience_CloseUp_Folder/Ambience_CloseUp.jpg

Vedlegg/Bilder/Blueprints_Folder/Ambience_BP_1.jpg

Vedlegg/Bilder/Blueprints_Folder/Ambience_BP_2.jpg

Vedlegg/Bilder/Blueprints_Folder/Ambience_BP_3.jpg

Vedlegg/Bilder/Blueprints_Folder/Ambience_BP_4.jpg

Vedlegg/Bilder/Blueprints_Folder/Ambience_BP_Overall.jpg

Vedlegg/Bilder/Blueprints_Folder/Portal_BP_1.jpg

Vedlegg/Bilder/Blueprints_Folder/Portal_BP_2.jpg

Vedlegg/Bilder/Blueprints_Folder/Portal_BP_3.jpg

Vedlegg/Bilder/Blueprints_Folder/Portal_BP_4.jpg

Vedlegg/Bilder/Blueprints_Folder/Portal_BP_Func_1.jpg

Vedlegg/Bilder/Blueprints_Folder/Portal_BP_Func_2.jpg

Vedlegg/Bilder/Blueprints_Folder/Portal_BP_Overall.jpg

Vedlegg/Bilder/Blueprints_Folder/Room_Func_1_Find_Nearest_Portal.jpg

Vedlegg/Bilder/Blueprints_Folder/Room_Func_2_Ambience_Follow_Player.jpg

Vedlegg/Bilder/Blueprints_Folder/Room_Func_3_Add_to_Parent_Emitter_List.jpg

Vedlegg/Bilder/Blueprints_Folder/Room_Func_4_Add_to_Parent_Ignore_List.jpg

Vedlegg/Bilder/Blueprints_Folder/Room_Func_5_Remove_from_Parent_Ignore_List.jpg

Vedlegg/Bilder/Blueprints_Folder/Rooms_BP_1_Instantiate.jpg

Vedlegg/Bilder/Blueprints_Folder/Rooms_BP_2_Registering_Emitters.jpg

Vedlegg/Bilder/Blueprints_Folder/Rooms_BP_3_Registering_Emitters.jpg

Vedlegg/Bilder/Blueprints_Folder/Rooms_BP_4_Enter_Exit.jpg

Vedlegg/Bilder/Blueprints_Folder/Rooms_BP_5_Initiate_Room_Check.jpg

Vedlegg/Bilder/Blueprints_Folder/Rooms_BP_6_Ambience.jpg

Vedlegg/Bilder/Blueprints_Folder/Rooms_BP_7_Ambience.jpg

Vedlegg/Bilder/Blueprints_Folder/Rooms_BP_8_Occlusion.jpg

Vedlegg/Bilder/Blueprints_Folder/Rooms_BP_9_Occlusion.jpg

Vedlegg/Bilder/Blueprints_Folder/Rooms_BP_10_Occlusion.jpg

Vedlegg/Bilder/Blueprints_Folder/Rooms_BP_11_Occlusion.jpg

Vedlegg/Bilder/Blueprints_Folder/Rooms_BP_12_Occlusion.jpg

Vedlegg/Bilder/Blueprints_Folder/Rooms_BP_13_Obstruction.jpg

Vedlegg/Bilder/Blueprints_Folder/Rooms_BP_14_Obstruction.jpg

Vedlegg/Bilder/Blueprints_Folder/Rooms_BP_15_Obstruction.jpg

Vedlegg/Bilder/Blueprints_Folder/Rooms_BP_Overall.jpg

Vedlegg/Bilder/Blueprints_Folder/Static_3D_Emitter_BP.jpg

Vedlegg/Bilder/Rooms_Portals_Folder/Multiple_Rooms_Portals.jpg

Vedlegg/Bilder/Rooms_Portals_Folder/Nested_Rooms_Portals.jpg

Vedlegg/Bilder/Rooms_Portals_Folder/Portal_CloseUp.jpg

Vedlegg/Bilder/Rooms_Portals_Folder/Room_CloseUp_1.jpg

Vedlegg/Bilder/Rooms_Portals_Folder/Room_CloseUp_2.jpg

Vedlegg/Bilder/Rooms_Portals_Folder/Rooms_Portals.jpg

Vedlegg/Bilder/Rooms_Portals_Folder/Rooms_Portals_Level_Placement_Example.jpg

Vedlegg/Bilder/Static_3D_Emitter_Folder/Static_3D_Emitter.jpg

Vedlegg/Bilder/System_CloseUp_Folder/System_Layout_1.jpg

Vedlegg/Bilder/System_CloseUp_Folder/System_Layout _2.jpg

Vedlegg/Bilder/System_CloseUp_Folder/System_Layout _3.jpg

Vedlegg/Bilder/System_CloseUp_Folder/System_Layout _4.jpg

Vedlegg/Bilder/System_CloseUp_Folder/System_Layout _5.jpg

Vedlegg/Bilder/System_CloseUp_Folder/System_Layout _6.jpg

Vedlegg/Bilder/Walkthrough_Photos_Folder/Env_Sound_System_Walkthrough_All_Emitters.jpg

Vedlegg/Bilder/Walkthrough_Photos_Folder/Env_Sound_System_Walkthrough_Top_View.jpg

Vedlegg/Bilder/Walkthrough_Photos_Folder/Env_Sound_System_Walkthrough_Room.jpg

Vedlegg/Bilder/Walkthrough_Photos_Folder/Env_Sound_System_Walkthrough_Room_2.jpg

Vedlegg/Bilder/Walkthrough_Photos_Folder/Emitters_Placed_In_Room.jpg

9.2. Videofiler

Vedlegg/Filmer/Env_Sound_System_Concept_Walkthrough_W_Music.MP4

Vedlegg/Filmer/Env_Sound_System_Implementation_Example_W_Audio.MP4

Vedlegg/Filmer/Env_Sound_System_Walkthrough_Showcase_Audio_Edit.MP4

Vedlegg/Filmer/Env_Sound_System_Advanced_Occlusion_Demo.MP4

9.3. Spillfiler

Vedlegg/Environmental Sound System – Playable Version/WindowsNoEditor/Env_SoundSystem.exe

9.4. Prosjektfiler

Unreal Engine 4 Prosjekt:

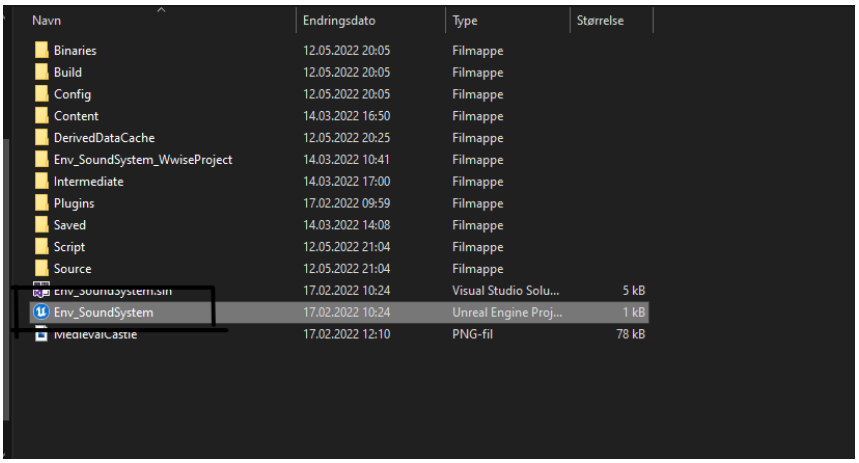
Vedlegg/UE4 – Wwise Prosjekt/Env_SoundSystem/Env_SoundSystem.uproject

Wwise Prosjekt:

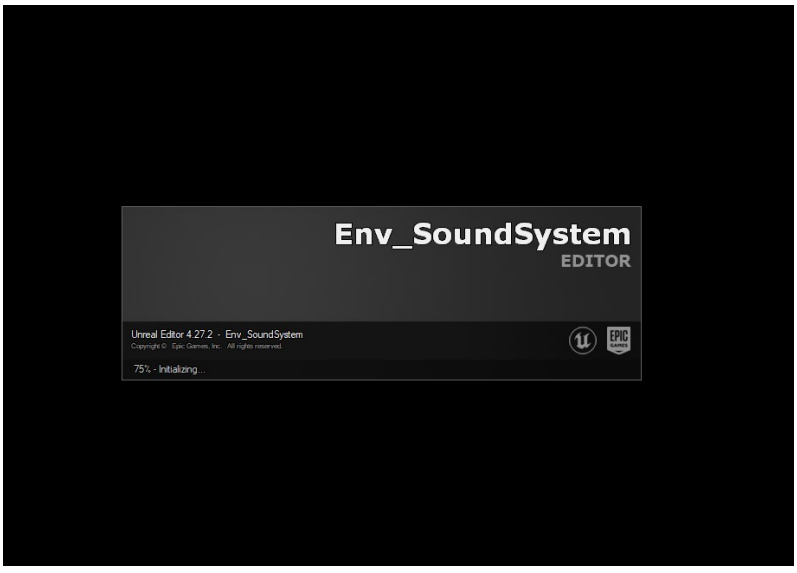
Vedlegg/UE4 – Wwise Prosjekt/Env_SoundSystem/Env_SoundSystem_WwiseProject/
Env_SoundSystem_WwiseProject.wproj

9.5. Navigasjonsveiledning for prosjektet

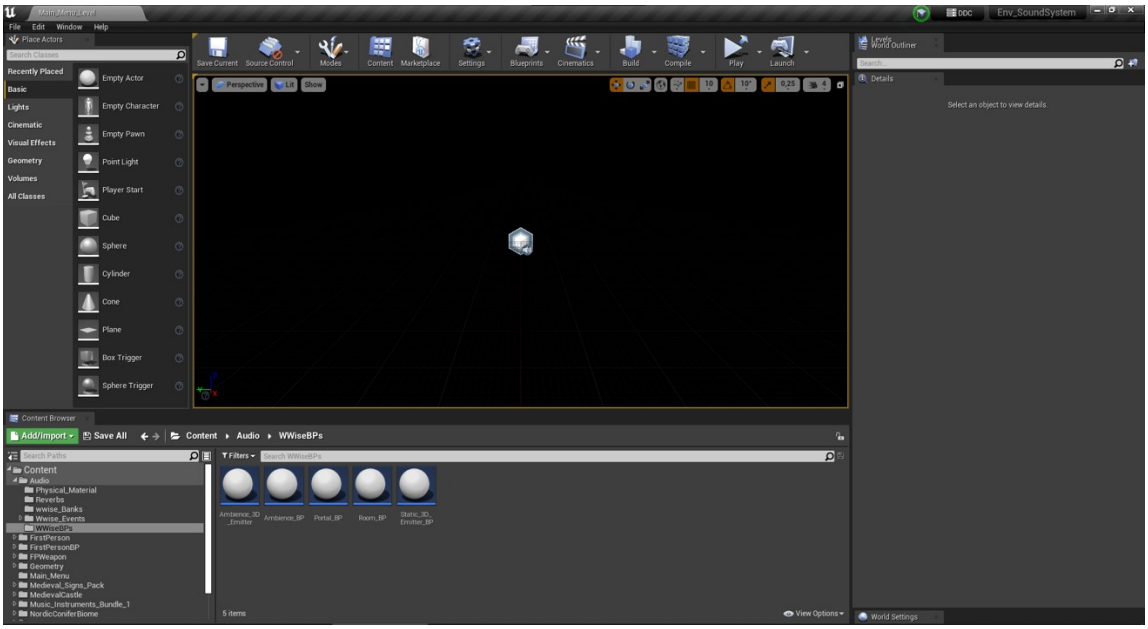
1. Etter at både Unreal Engine 4 og Audiokinetic Wwise er installert, åpne filen som heter *Env_SoundSystem.uproject.* Den finnes i mappen *Vedlegg/UE4 – Wwise Prosjekt/Env_SoundSystem/.*



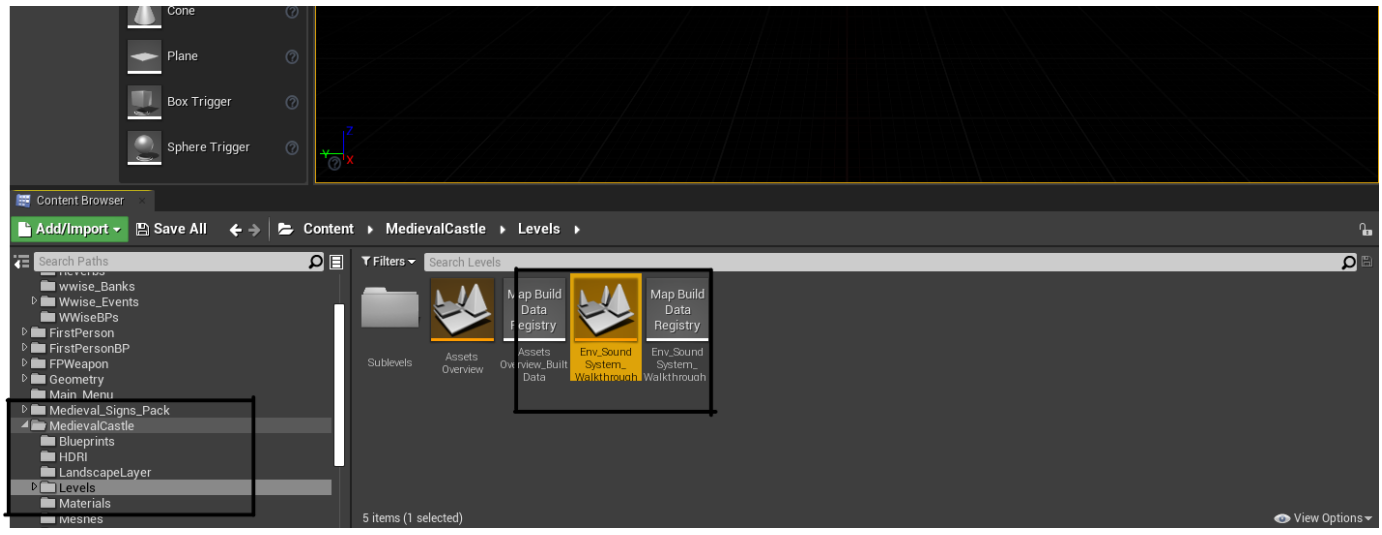
2. Hele spillprosjektet med tilhørende filer for det omgivelsesbaserte lydsystemet vil da lastes inn. *NB! Er dette første gangen man åpner prosjektet, vil dette ta en stund.*



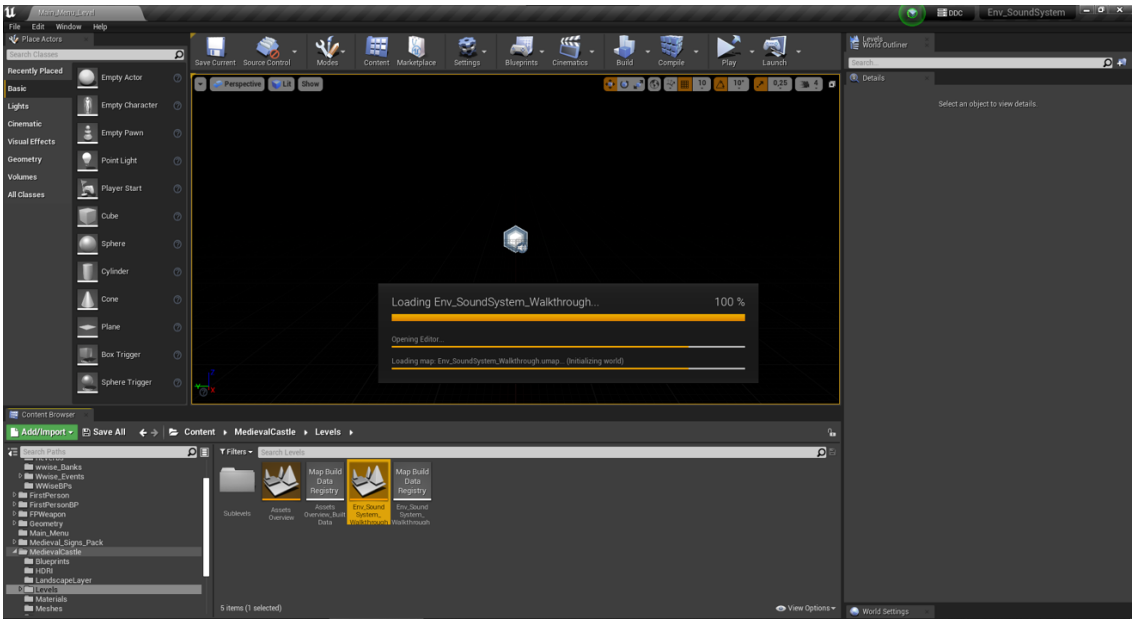
3. Prosjektet vil etterhvert åpnes og man vil se et vindu som ser tilnærmet likt ut som dette:



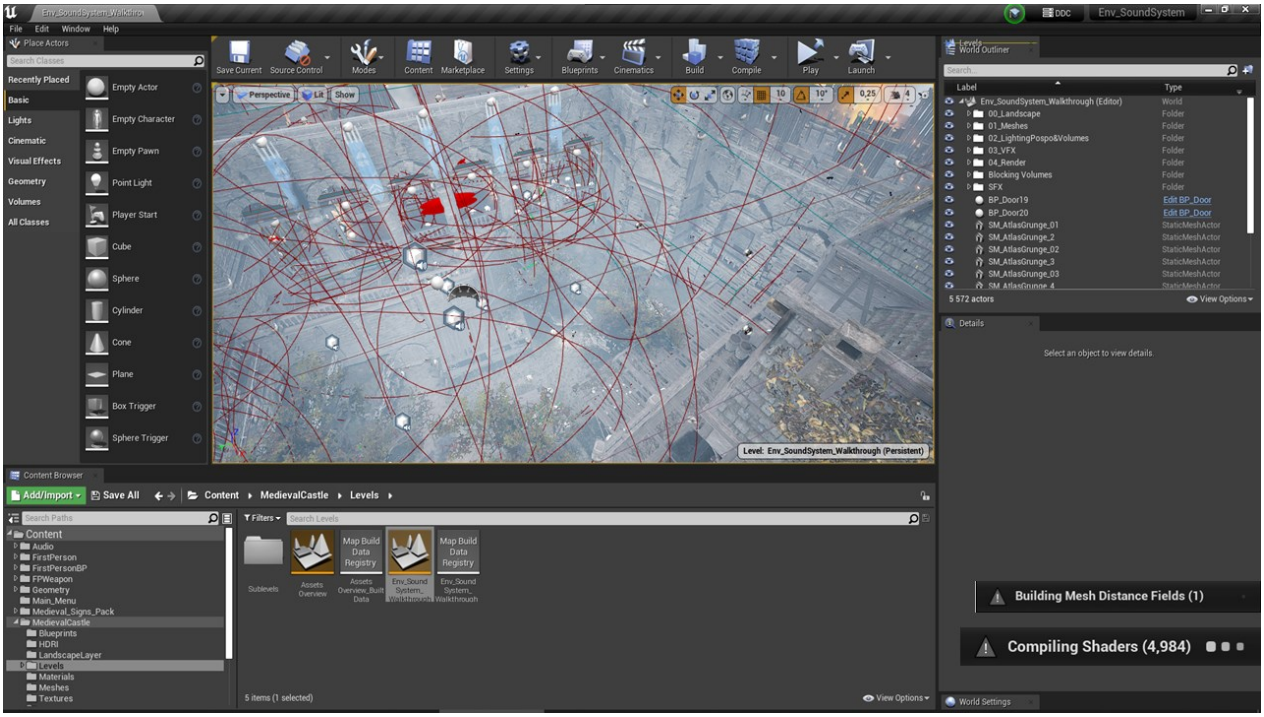
4. Nede til venstre, vil man se et mindre vindu som heter for en *Content Browser*. Denne gir en oversikt over alt innhold i prosjektet. Herfra kan man bla ned til man ser en mappe som heter *MedievalCastle*. Trykk på den, og åpne mappen som heter *Levels.* Der finner man banen som brukes for å demonstrere det omgivelsesbaserte lydsystemet i en mer helhetlig sammenheng. Dobbeltklikk deretter på ikonet som heter *Env_SoundSystem_Walkthrough* for å åpne banen.



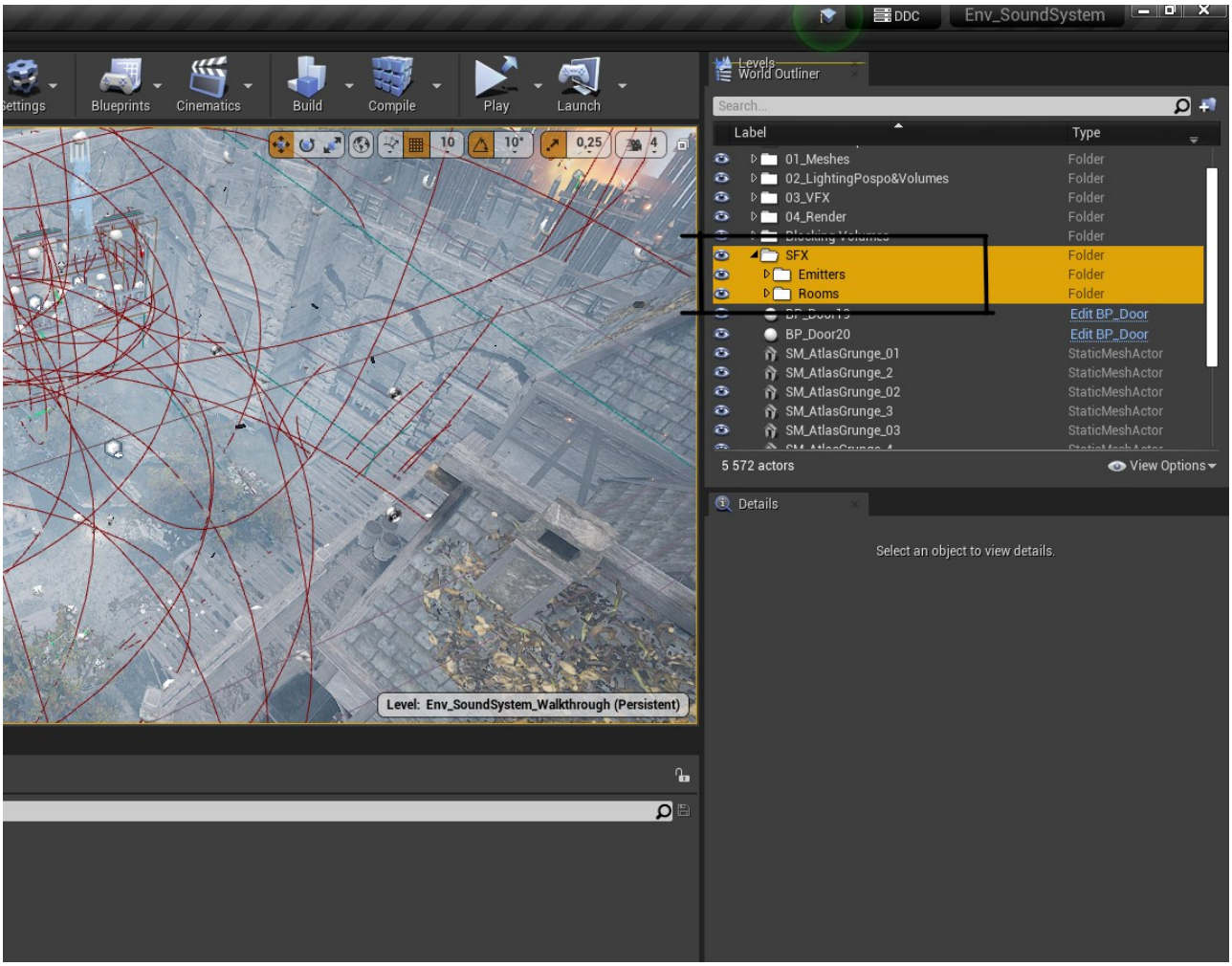
5. Etter man har dobbeltklikket på ikonet, vil banen begynne å lastes inn, som vist i bildet under.
   NB! Er dette første gang man åpner banen, vil dette ta en stund.



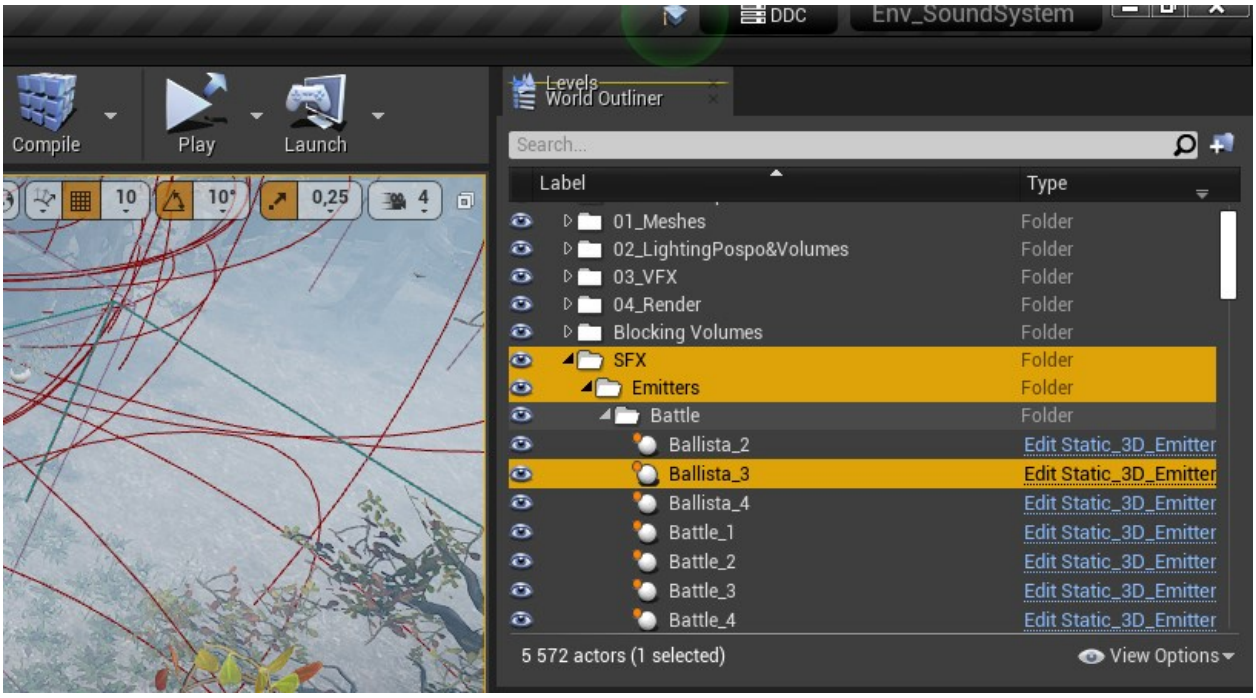
6. Dersom det er første gang man åpner banen kan man få noen beskjeder nederst til høyre på skjermen, som vist i bildet under. Hvis dette dukker opp bør man vente til prosessene som utføres er ferdige og beskjedene forsvinner. Dette kan ta litt tid.



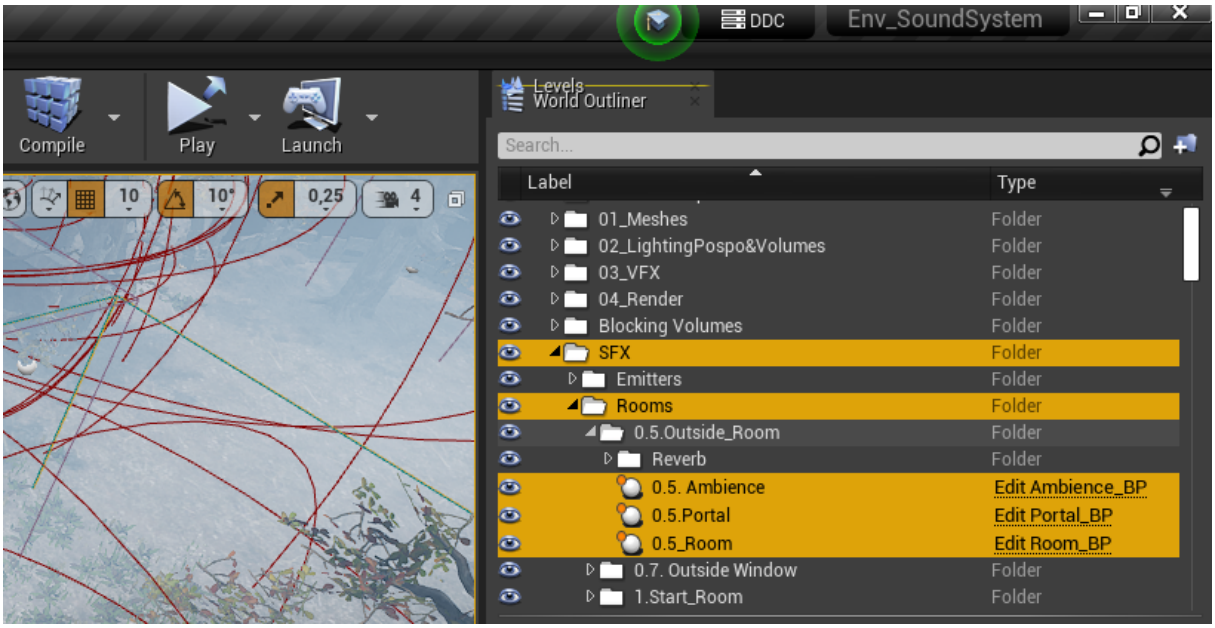
7. Øverst til høyre vil man ha et mindre vindu som heter en *World Outliner*. Her vil alle spillobjektene i den aktuelle banen være listet opp. Blar man ned til mappen som heter *SFX*, vil man finne to ekstra mapper som heter *Emitters* og *Rooms*. I disse mappene vil alle objektene tilknyttet det omgivelsesbaserte lydsystemet for denne banen være listet opp. Man kan dobbeltklikke på disse for å se deres plassering i banen.



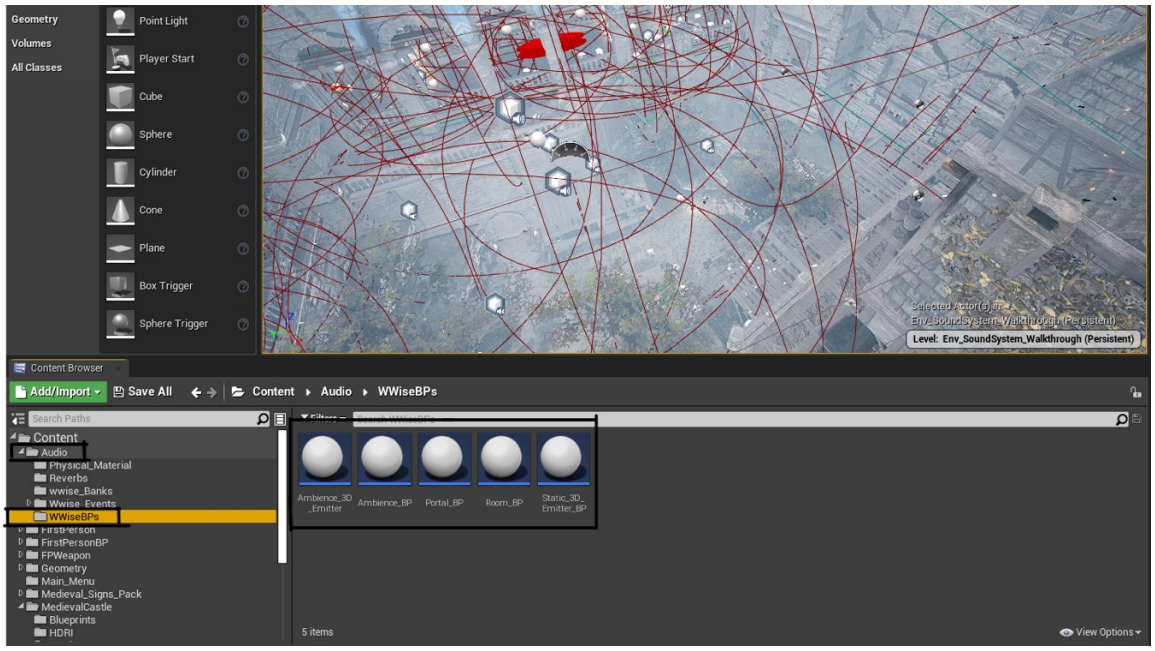
8. Under mappen *Emitters*, vil alle de statiske tredimensjonale audio emitterene for denne banen være listet opp.



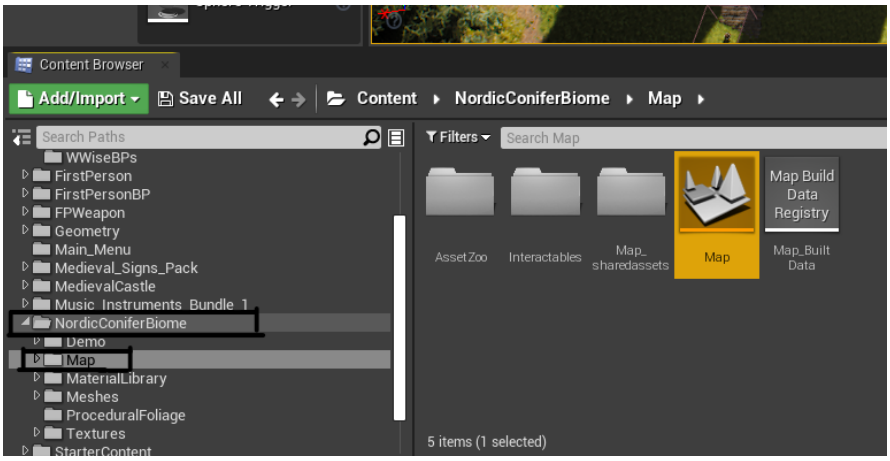
9. I mappen *Rooms,* vil alle objektene tilknyttet rom, portaler og atmosfærelyder for denne banen være listet opp.



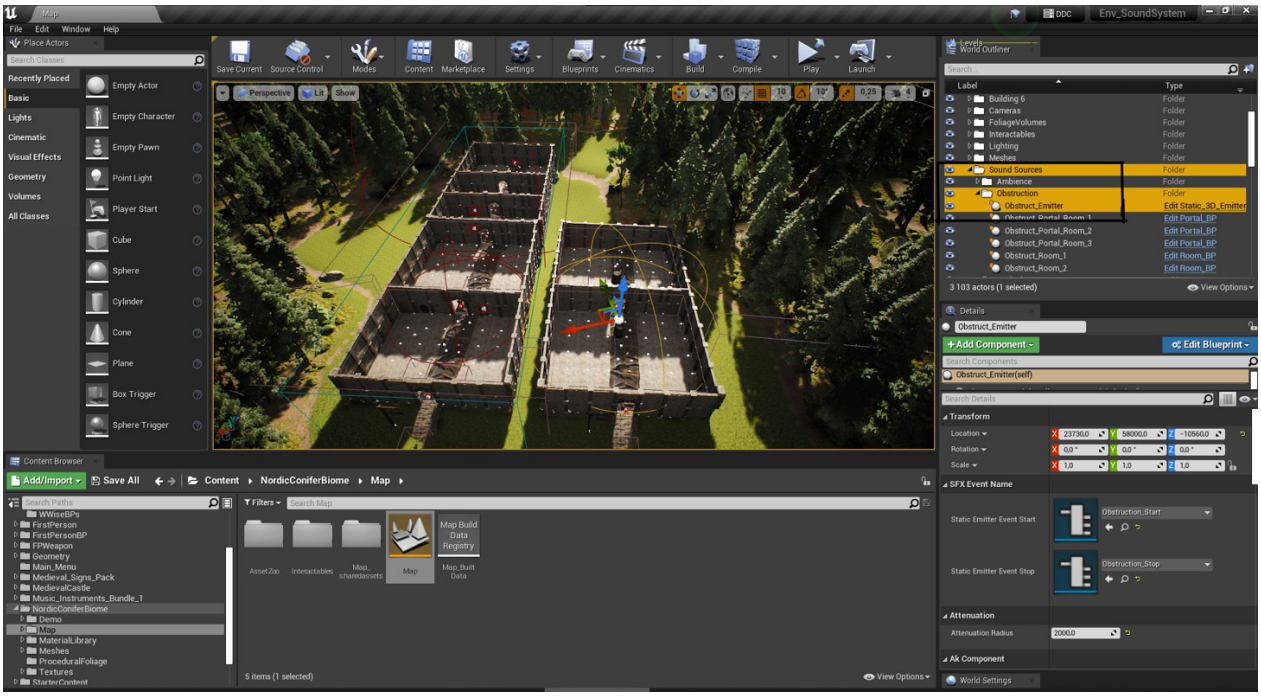
10. I *Content Browseren* nederst til venstre, kan man trykke på mappen som heter *Audio* og deretter på mappen som heter *WwiseBPs*. ***Her vil man finne selve spillobjektene til det omgivelsesbaserte lydsystemet.*** Det er disse objektene man drar ut i spillverdenen. Dersom man ønsker å se koden til disse, kan man dobbeltklikke på dem. Disse kan dras ut i spillverden og testes dersom det ønskes.



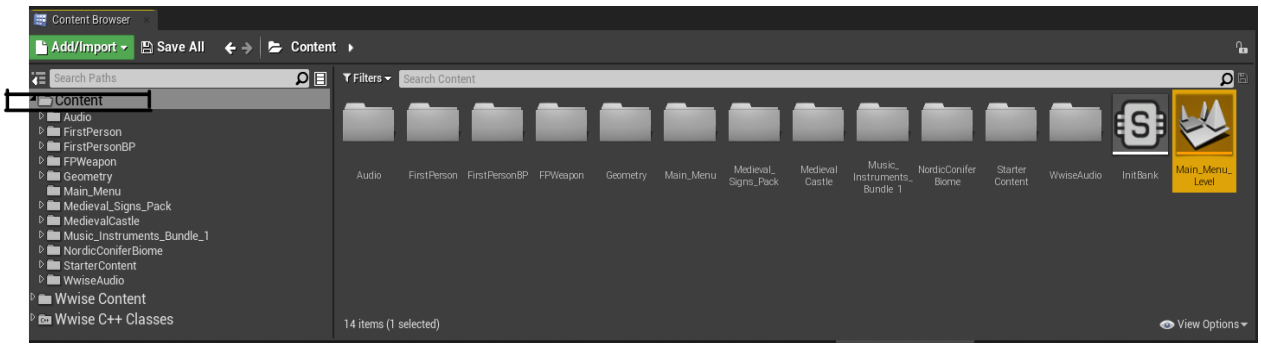
11. Hvis man ønsker å se hvordan banen som demonstrerer de ulike konseptene ser ut, kan man i *Content Browseren*, bla ned til mappen som heter *NordicConiferBiome*, og trykke på mappen som heter *Map.* Derfra kan man dobbeltklikke på ikonet som heter *Map. NB! Dersom dette er første gang man åpner banen, kan dette ta en god stund.*



12. Vinduet som åpnes, vil da se tilnærmet likt ut som bilde under. I *World Outlineren*, kan man bla ned til man finner mappen som heter *Sound Sources.* Her finner man mappene med de ulike objektene for det omgivelsesbaserte lydsystemet i denne banen. Et eksempel av dette er vist i bildet under.



13. Dersom man ønsker å prøve hele spillet i Unreal Engine 4 istedenfor gjennom vedlegget *Vedlegg/Environmental Sound System – Playable Version/WindowsNoEditor/Env_SoundSystem.exe,* kan man gjøre dette fra spillets hovedmeny. I *Content Browser,* trykk på mappen som heter *Content.* Deretter dobbeltklikk på ikonet som heter *Main_Menu_Level* for å åpne hovedmenyens bane.



14. For å starte spillet, kan man i verktøylinjen øverst finne ikonet som heter *Play,* til venstre for *World Outlineren*, og trykke på den. Dette kan også gjøres for alle banene dersom man kun ønsker å teste en spesifikk bane.

Alexander Polanco Olsen

Teknisk Lyddesign i Dataspill

NTNU
Kunnskap for en bedre verden