Universitetet i Oslo
Institutt for informatikk

Dynamiske soner i samvirke med dynamisk fokus og aura - en studie av relevansfiltrering for nettverksbaserte virtuelle omgivelser

Sigve Høghaug

Hovedfagsoppgave i informatikk

25. juli 2003

Sammendrag

Utviklingen innen nettverksbaserte virtuelle omgivelser har vært stor de siste 20 årene, hvor spillindustrien den senere tid har seilt opp som en viktig aktør. Tradisjonelt ble gjerne nettverksfasilitetene ved et spill implementert mer som et tillegg til et allerede ferdig utviklet spill, men etterhvert som de *rene* nettverksbaserte spillene begynte å dukke opp fra siste halvdel av 90-tallet ble som følge av dette nettverksdelen en essensiell del av design og utvikling.

Nettverksspill står overfor problemer og utfordringer man ikke finner hos spill som kun kjører på en maskin. Store flerbrukerspill, hvor tusentalls av deltagere skal kunne delta samtidig, må rette seg etter de ressurser Internett til enhver tid tilbyr. Brukere som interagerer med en nettverksbasert virtuell omgivelse vil typisk sende oppdateringspakker ut på nettverket. Siden båndbredde er en begrenset ressurs må det tas i bruk teknikker for å minimalisere nettverkstrafikken som genereres av deltakerne.

Denne oppgaven presenterer et forslag til hvordan det for en gitt klient kan defineres hva som er av relevans i en virtuell omgivelse - en *relevansfiltrering*. Hensikten er å begrense det sett av klienter en deltaker kommuniserer med. Dette skjer ved å definere et *interesseområde*. Vi vil her benytte prinsipper om en *dynamisk soneinndeling* og *dynamisk fokus og aura* som en løsning på den ovenfornevnte problemstilling. Disse to begrepene isolert sett, slik de i denne oppgaven presenteres, baserer seg på tidligere forslag og ideer. Det vi her skal se på er hvordan disse to begrepene i *samvirke* definerer en ikke tidligere måte å utføre relevansfiltreringen på.

Vi foreslår en todeling av relevansfiltreringen - *avstandsfiltrering* og *interessefiltrering*. Hovedvekten er lagt på avstandsfiltreringen; oppgaven presenterer og beskriver en implementasjon av denne og resultatene tilsier at ideene som presenteres i oppgaven har en nytteverdi.

Forord

Denne hovedfagsoppgaven innen studieretningen kommunikasjonssystemer er skrevet ved Universitetet i Oslo, Institutt for Informatikk, som en del av min cand. scient.-grad.

Jeg vil takke mine to veiledere Lars Aarhus og Olav Lysne for et flott samarbeid de to årene jeg har jobbet med oppgaven. Jeg vil også takke min bror, Leif Harald, for språklig konsulentarbeid. Tusen takk!

Blindern, 25. juli 2003

Sigve Høghaug

Innhold

KAPITTEL 1

Innledning

1.1 Motivasjon

Utviklingen innen nettverksbaserte virtuelle omgivelser har vært stor de siste 20 årene hvor spillindustrien har vært den viktigste aktøren siden midten av 90-tallet. Internett har bidratt til utviklingen av spill hvor tusentalls av brukere samtidig kan interagere i en felles distribuert virtuell verden.

Design av spill eller simuleringer[1] som kjører over et nettverk møter utfordringer knyttet til deres distribuerte natur:

- *Skalerbarhet* - En distribuert simulering må støtte et stort antall brukere og en spredning av brukere over et stort geografisk område. Kompleksiteten til en en nettverksbasert virtuell omgivelse (nett-VO) øker eksponensielt ($2^{(antall-entiteter)}$) siden alle entiteter i teorien skal kunne interagere med hverandre.

- *Forsinkelse* - Nettverksforsinkelse er tiden det tar å overføre et *bit* fra en endeterminal til en annen. Stor forsinkelse vil påvirke realismen i en nett-VO som opererer i sann-tid.

- *Pålitelighet* - En destinasjon mottar ikke alltid pakker den blir tilsendt. Re-transmisjon og metningskontroll (som benyttes i i TCP) er ikke gunstig i en sann-tids simulering; pakker som kommer for sent frem vil føre til at simuleringen må gå bakover i tid.

[1] Termen *simulering* brukes her om virtuelle omgivelser generelt - altså simulering av den virkelige verden eller noe som har visse likhetstrekk med denne hvor deltakere skal kunne interagere med hverandre i sann-tid. Spill blir dermed en simulering, men en simulering trenger ikke være et spill (f. eks. millitære simuleringer eller virtuelle omgivelser som kun er laget i forskningsøyemed).

- *Konsistent tilstandsoppdatering* - Tilstandsoppdateringer må prosesseres hos hver endetermainal slik at konsistens opprettholdes underveis i simuleringen for alle deltakere.

- *Båndbredde* - Størrelse og mangfold for en simulering over Internett bestemmes av den tilgjengelige båndbredden. For hver ny bruker som kommer til en simulering øker kravene til båndbredde. Hvis hver ny deltaker skal sende oppdateringspakker til alle de andre deltakere samt motta oppdateringspakker fra disse, vil bruk av båndbredde øke eksponensielt.

Det er problematikken relatert til en distribuert simulerings bruk av båndbredde denne oppgaven omhandler. Dette blir presentert nærmerer i seksjon 1.4, men først vil vi se mer på hva en nettverksbasert virtuell omgivelse er for noe, samt litt historikk rundt temaet.

1.2 Den virtuelle omgivelse

Helt generelt kan man forklare en *nettverksbasert virtuell omgivelse* med et softwaresystem hvor flere brukere er aktører og hvor disse interagerer med denne verdenen og de andre deltakerne, og at dette skjer i sann-tid ved at hver deltager er koblet opp til et nettverk (f.eks. Internett eller LAN[2]). I [1] betegnes en virtuell omgivelse som en verden inne i en datamaskin, mens en distribuert virtuell omgivelse kan betraktes som en verden inne i et nettverk av datamaskiner hvor brukere av forskjellige datasystemer kan interagere med hverandre.

En slik verden er gjerne representert med 3-dimensjonell grafikk - geometriske objekter som er bygget opp av polygoner[3]. En grafisk representasjon av en bruker (være seg en person, kjøretøy e.l.) i en virtuell omgivelse kalles gjerne en *avatar*[4].

Man ønsker å skape en felles oppfatning av tid og sted for deltagerne i en nettverksbasert virtuell omgivelse til tross for at disse kan være spredt over et stort geografisk område. Man vil at alle deltagere skal ha den samme oppfatningen av tid - at en persons interagering med omgivelsene skal oppfattes av de andre personene idet den skjer - såkalt *sann-tid*. Et av hovedmålene til designere av slike systemer blir da å skjule disse avstandene slik at man

[2]Local Area Network.

[3]Et polygon er en flate avgrenset av tre eller flere linjer (det enkleste polygon blir dermed et triangel). Overflaten til 3-dimensjonelle figurer i en virtuell-omgivelse er satt sammen av slike polygoner. Jo flere polygoner en figur er bygget opp av, jo større detaljrikdom (men det kreves mer prosesseringskraft jo flere polygoner som er med å bygge opp figurene i en VO, så det er mest gunstig å designe figurer med færrest mulig polygoner).

[4]Bruken av termen avatar i denne sammenhang ble popularisert av Neal Stephenson gjennom sci-fi romanen $Snow\ Crash$ (1992). På sanskrit brukes ordet avatar om "kroppene som gudene benytter når de skal vandre blant menneskene på jorden".

får den oppfatning at man befinner seg i samme rom som en person som i virkeligheten befinner seg på andre siden av jorden. Brukere skal kunne påvirke hverandre på en realistisk måte, men de skal også kunne påvirke tingene rundt seg i en virtuell omgivelse. En virtuell verden må da også være *rettferdig* i den forstand at hver enkelt aktør har de samme muligheter tatt i betraktning at nettverkskapasitet og maskinstyrke varierer fra bruker til bruker[5]. Det vil si at hver enkelt skal ha den samme visuelle oppfattelse og de samme muligheter til å interagere i den virtuelle omgivelsen.

Det er forskjellige måter man kan "delta" i en virtuell omgivelse på. Man kan bruke en standard PC hvor man kommuniserer og interagerer ved hjelp av tastatur, mus og skjerm. For å skape en større virkelighetsoppfatning er det utviklet briller med skjerm (*head-mounted display*) som hindrer at man blir distrahert av sanseinntrykk utenfor den virtuelle omgivelsen. Sensorer festet til disse detekterer hodebevegelser slik at man kan oppdatere skjermbildene i henhold til hodebevegelser. Det er også utviklet en femsidet kube ($the\ CAVE$ [3]) hvor man beveger seg i en virtuell omgivelse ved at skjermbilder oppdateres på veggene. For den alminnelige hjemmebruker er det pr. i dag helt vanlig PC-utstyr som benyttes for å interagere og delta i virtuelle omgivelser (og da i hovedsak spill), selv om flere på midten av 1990-tallet trodde spesialutstyr (slik som spesielle briller og hansker) ville bli vanlig å finne hos privatpersoner (se f.eks. [2]).

Videre i denne innledningen vil vi se på utviklingene så langt innen nettverksbaserte virtuelle omgivelser. Vi vil så se på problematikken relatert til nettverksressurser for en distribuert virtuell omgivelse. Det er da spesielt hvordan hver endebruker kan begrense transmisjonen av oppdateringspakker som er av interesse, og hvor hovedfokus vil være hva som kan gjøres for at den enkelte endebruker minimaliserer sin transmisjon av oppdateringspakker til de andre klientene i simuleringen.

1.3 Bakgrunn

Virtuelle omgivelser har utviklet seg innen tre forskjellige forskningsmiljøer:

- *Militært*
- *Akademisk*
- *Kommersielt*

1.3.1 SIMNET

$SIMNET$ ble begynt utviklet i 1983 med det mål å skape en virtuell nettomgivelse for å simulere krigssituasjoner. SIMNET, som ble utviklet av DAR-

[5]Dagens nettverksspill designes slik at brukere med et modem på 56.6K skal få den samme spillopplevelsen som en bruker som har f.eks. bredbåndtilkobling.

PA[6], trekkes frem som viktig i utviklingen av nettverksbaserte virtuelle omgivelser. Dette fordi $The\ US\ Department\ of\ Defense$ (DoD) var en av de første utviklerne innen nett-VOer og at de var først ute med nett-VOer i stor skala [4]. SIMNET arkitekturen består av tre hovedkomponenter:

- Den er *objekt-hendelses orientert* - en virtuell verden består av objekter som interagerer med hverandre, hvor hvert enkelt objekt (f.eks. et fly, eller en bil) vanligvis er styrt av én maskin.
- Den har en *autonom simulering* - individuelle spillere, fartøyer og våpensystemer i nettverket er selv ansvarlige for å sende ut pakker med informasjon om deres tilstand. Hver bruker er da ansvarlig for å selv motta slike pakker for korrekt å kunne oppdatere sin lokale modell av den virtuelle omgivelsen.
- Den tredje hovedkomponenten er bruken av *dead reckoning* algoritmer (se seksjon 1.4.3 på side 9) for å minske belastningen på nettverket og hver enkelt CPU.

1.3.2 NPSNET

Innenfor den akademiske verden trekkes gjerne $NPSNET$[7] $Research\ Group$ frem som den viktigste bidragsyter i utviklingen av nett-VOer [5]. NPSNET startet opp i 1986 ved $Naval\ Postgraduate\ School$ (USA) med fokus på utvikling av teknologi for DoD (m. a. o. millitære simuleringer). Et hovedmål for NPSNET fra begynnelsen av var å kunne benytte arbeidsstasjoner med begrenset grafisk ytelse og holde kostnader lave for visuell simulering. NPSNET I og II var begrenset til lokale nettverk og krevde Ethernet som LAN protokoll.

Så kom $NPS-Stealth$ som var en videreutvikling av NPSNET I. NPS-Stealth tok utgangspunkt i SIMNET protokollen for kommunikasjon over nettverk, som åpnet for WAN[8] kommunikasjon ved å knytte lokalnett sammen ved hjelp av broer.

$NPSNET-IV$ var den første virtuelle omgivelsen som tok i bruk både *IEEE 1278 Distributed Interactive Simulation* (DIS) [6] og IP Multicast nettverksprotokollen for multi-player simulering over Internett. DIS er en protokoll designet slik at forskjellige utviklere kan lage forskjellige simuleringer på forskjellige maskiner som i teorien skal kunne dele den samme virtuelle omgivelsen. DIS er implementert med en rekke standarder som f.eks. kommunikasjonsarkitektur, entitetsinformasjon og integrasjon, sikkerhet og databaseformat.

[6] $The\ Defence\ Advanced\ Research\ Projects\ Agency$ (DARPA) er den sentrale forsknings og utviklingsorganisasjonen ved $The\ US\ Department\ of\ Defense$ (DoD).

[7] $Naval\ Postgraduate\ School\ Networked\ Vehicle\ Simulator$

[8] Wide Area Network (WAN) er et nettverk som spenner over større geografiske områder, og som typisk binder sammen lokale nettverk (LAN).

$NPSNET - V$ er den siste i rekken av forskningsprosjekt hos Naval Postgraduate School. Den bygger på NPSNET IV, og forskningen er i hovedsak innen dynamisk utvidelse av store net-VOer. NPSNET V er ment som et rammeverk for å kunne skape virtuelle omgivelser hvor nye komponenter skal kunne legges til under kjøring. Deltakere skal da kunne laste ned oppdateringer via nettverk under kjøring fremfor å måtte rekompilere og re-starte simuleringen [10].

1.3.3 Interaktive, distribuerte flerbrukerspill

I 1961 ble det utviklet et spill ved $Massachusetts\ Institute\ of\ Technology$ (MIT). Spillet het $Spacewar$! [7], og regnes som det første flerbrukerspill. Det ble programmert på en PDP-1[9] (den første "minicomputer") av Stephen R. Russell. Spillet går ut på at to personer skal styre hvert sitt romskip hvor målet ganske enkelt er å skyte ned motstanderen. Interessen for spillet var stor blant studenter ved MIT og etter hvert andre forskningsinstitusjoner i USA. Flere personer ble engasjert i å modifisere og videreutvikle spillet, som f.eks. å la raketter bli påvirket av gravitasjon og effekter som elektrisk sjokk når man ble truffet.

$Spacewar$! begrenset seg til to spillere og hadde neppe den samme innflytelse på nettverksbaserte virtuelle omgivelser som $Flight$ og $Dogfight$. Flight ble utviklet av Gary Tarolli ved $Silicon\ Graphics,\ Inc.$ (SGI)[10] i 1983. Året etter ble nettverksversjonen presentert som benyttet seg av multicasting over Ethernet. I 1985 ble programmet oppgradert til Dogfight. Nå kunne spillerne interagere med hverandre i form av å skyte ned fly (i Flight kunne brukerne se andres fly, men ikke påvirke hverandre). Spillet la beslag på mye båndbredde, og begrenset seg til et titalls brukere. Flight og Dogfight ble distribuert med alle nye SGI-arbeidsstasjoner, og kildekoden var tilgjengelig for alle. Dette sees på som viktige ledd i utviklingen av net-virtuelle omgivelser [4]. Forskningsgruppen $PARADISE$[11] springer ut fra tidlige eksperimenteringer med å utvide Dogfight med dynamiske terreng (f.eks. at en spiller kan lage et krater i bakken ved å krasje et fly).

$MUDs$ (Multiple user dungeon) er en betegnelse på et program utviklet av Roy Trubshaw ved universitetet i Essex i 1979. Hensikten med dette programmet var å utvikle tekstbaserte flerbrukerspill. Den dag i dag finnes det forskjellige MUDs, både for underholdningsøyemed og innenfor undervisinig.

[9] PDP-1 ble introdusert i 1960 av $Didgital\ Equipment\ Corp.$. Den kostet 120.000 $ som var en lav pris sammenlignet med andre computere på den tiden. Den trengte ikke ventilasjon, kunne opereres av kun en person og fikk plass i hjørnet av et rom.

[10] $Silicon\ Graphics$ ble dannet i 1982 av James Clark, og har lenge vært ledende i produksjon av grafiske arbeidsstasjoner og super-computere.

[11] $PARADISE$ (Performance Architecture for Advanced Distributed Interactive Simulation Environments) ble startet ved Stanford University i 1993 for å jobbe med problemstillinger relatert til virtuelle omgivelser med tusenvis av brukere.

I 1993 kom spillet $Doom$ som regnes for å være trendsetteren for sjangeren første-person skytespill og skapte praktisk talt en ny seksjon innen spill industrien [8]. Spillet kunne i starten spilles over LAN med inntil fire spillere, men arbeidet med å videreutvikle spillet til å støtte TCP/IP for å kunne kjøre over Internett kom raskt i gang. Det er også i denne tidsperioden Internett virkelig åpner for bruk av grafisk grensesnitt (f.eks. så ble web-browseren $Mosaic$[12] lansert i 1993). Brukerne av Internett hadde inntil da hovedsakelig vært innen det offentlige, militæret, utdanningssektoren og forskning, mens den kommersielle delen var helt i startfasen.

I 1997 ble $Ultima\ Online$, som regnes som det første suksessfulle $MMORPG$[13], sluppet for spilling over Internett. Spillet fikk 50,000 betalende spillere innen tre måneder, noe som beviste at det var et stort marked for rollespill over Internett. Utviklingen av Internett har dessuten gjort det mer gunstig for brukere av denne type spill siden stadig flere får tilgang til bredbåndstjenester, samt at disse stadig blir billigere.

I dag finnes det mange rollespill med et stort antall brukere (f.eks. $Asherons\ Call$ og $EverQuest$). Disse spillene blir bare mer og mer komplekse. De er ikke bare spill lenger, men kan i mange tilfeller sees på som virtuelle "chatte-program" - det sosiale aspektet blir viktig. For noen brukere har disse spillene blitt "big-business"; på nettauksjonen $eBay$ blir f. eks. effekter (karakterer, hus, gull o.l.) til Ultima Online solgt til en ofte god penge.

Noe av det spesielle med disse spillene er at det representerer såkalte *persistente verdener*, det vil si at spillet kjører kontinuerlig selv om spillere logger av.

Av nyere Internett-basert flerbruker spill kan man nevne $Anarchy\ Online$ som er laget av norske $FunCom$[14]. Firmaet $Blizzard$, som står bak suksesser som $Warcraft$, $Starcraft$ og $Diablo$ har latt seg inspirere av Anarchy Online til å utvikle $World\ of\ Warcraft$ [9], og $Sony\ Online\ Entertainment$ slapp 26. juni 2003 spillet $Star\ Wars\ Galaxies$ for kommersiell spilling over Internett. Figur 1.1 på neste side viser et skjermbilde fra sistnevnte spill og for Ultima Online. Ved å betrakte disse to kan vi se hvordan det grafiske har utviklet seg for distribuerte flerbrukerspill. Bildet til venstre hvor avatarene er gjengitt i mer detaljert 3D grafikk er typisk for alle nyere $MMORPG$.

[12] $Mosaic$ ble utviklet av en gruppe ved $National\ Center\ for\ Supercomputing\ (University\ of\ Illinois)$. De startet $Mosaic\ Communications$ som senere ble $Netscape\ Communications\ Corporation$.

[13] $MMORPG$ (Massively-multiplayer-Online-Role-Playing-Game).

[14] $FunCom$ ble dannet i 1993 og etablerte etterhvert kontorer i USA så vel som i Europa. Siden 1996 har selskapet hovedsakelig fokusert på utvikling av online-spill.

Figur 1.1: Til venstre skjermbilde fra spillet *Star Wars Galaxies* (hentet fra http://starwarsgalaxies.station.sony.com) og til høyre *Ultima Online* (hentet fra http://www.uo.com).

1.4 Tilnærminger for å redusere bruk av båndbredde

Et nettverk er opplagt en essensiell del av en nettverksbasert virtuell omgivelse (i kontrast til en omgivelse som kjører på kun én maskin). Det er flere utfordringer man står overfor når man skal designe en nett-VO: forsinkelse i nettet, tap av pakker og belastning på nettet som følge av stor pakketrafikk mellom deltakerne i omgivelsene. Det er sistnevnte problemstilling som er motivasjonen for denne oppgaven - hvordan minske transmisjonen av oppdateringspakker fra hver enkelt bruker.

Hovedvekten i dette avsnittet vil dreie seg om hva som kan gjøres på applikasjonsnivå for å begrense pakketrafikken over et nettverk. Vi vil også se på prinsippene bak *multicast* og *MBone* som gjerne trekkes frem som vesentlige begreper når det kommer til å minske pakketrafikken. Dernest vil vi se på en metode som kalles *Dead Reckoning*. Til slutt tar dette kapitlet for seg en teknikk som kalles *Relevansfiltrering*, og det er denne som er tema for oppgaven.

1.4.1 Båndbredde

Båndbredde refererer til kapasiteten til en kommunikasjonslinje - hvor mye data som kan sendes pr. tidsenhet. For et WAN kan båndbredde variere fra titals kbps (kilo bit pr. sek.) som vi finner hos såkalte dial-up modemer (28, 56 kbps) opp til 1.5 Mbps (T1) og 44.7 Mbps (T3).

Bruk av båndbredde øker i samsvar med antall nye brukere som kobler seg til en nett-VO fordi:

- Hver nye bruker må motta den initielle tilstanden til omgivelsene.

- Hver nye bruker introduserer nye oppdateringer til den eksisterende felles tilstanden til omgivelsene.
- Hver nye bruker blir en del av omgivelsenes felles tilstand.

Ressurser som brukes for en nett-VO kan defineres som følger [4]:

$$Resurser = M * K * B * T * P$$

hvor M er antall meldinger (pakker) som distribueres, K er hvor mange klienter som i snitt mottar hver melding, B er hvor mye båndbredde en enkelt melding legger beslag på, T er krav mht. hvor lang tid man tillater at nettverket bruker på å levere en pakke til hver destinasjon (store verdier for T indikerer strenge krav til forsinkelse, mens lave verdier indikerer det motsatte) og P er antall prosessor sykluser det kreves for å motta og prosessere hver melding. Det er K, altså antall klienter som motar en gitt pakke, som er av relevans for problemstillingen denne oppgaven presenterer.

1.4.2 Multicast og MBone

Multicast

Kommunikasjon over et nettverk kan forgå som en-til-en, eller som mange-til-mange/en-til-mange (gruppekommunikasjon). I en virtuell omgivelse hvor flere deltakere interagerer med hverandre er det den sistnevnte kommunikasjonsformen som gjelder.

Det fundamentale i gruppekommunikasjon er IP-multicast som benytter et gruppebasert adresseskjema. En gruppe er definert av en IP-multicast adresse. Man kan velge å abonnere på en multicast adresse (dette skjer via *Internet Group Management Protocol* - IGMP [21]). Man vil da motta pakker som er adressert til denne uten at det taes hensyn til hvem senderen er. Grupper oppløses ved at alle medlemmer forlater den - IP-multicast adressen frigjøres og blir klar til gjenbruk.

IP-multicast benytter *User Datagram Protocol* (UDP) og tilbyr derfor ingen pålitelighet - det gis ingen garanti for at pakker som sendes ut blir mottatt av alle abonnentene[15]. En pålitelig multicast-protokoll vil per i dag ikke være praktisk for større grupper. Man må da benytte re-transmisjonsskjemaer for hvert medlem i en gruppe, noe som blir for tidkrevende under kjøring i en sann-tids simulering.

Det ser ikke ut til at det lar seg gjøre å utvikle en transportprotokoll som kan konfigureres til å yte optimal tjenestekvalitet for alle mulige applikasjoner som ønsker å benytte IP-multicast, men det finnes protokoller som kan gi

[15]Problemet med å tilby pålitelighet for multicast er langt mer komplekst enn for unicast. Ta f. eks. en en-til-mange-forbindelse. Hvis en av mottakerene skulle miste pakker fra sender (metning eller feil på linken) blir problemet hvorvidt sender skal fortsette å sende pakker eller vente til den ene motakeren som ble brutt igjen er mottakelig for data.

en optimalisert tjenestekvalitet for et begrenset antall applikasjoner [11]. En mulig løsning er å ta i bruk et protokoll-rammeverk hvor flere protokoller inngår for å tilby en fullgod tjenestekvalitet til alle applikasjoner. *The Reliable Multicast Framing Protocol* (RMFP) er et eksempel som baserer seg på et slikt rammeverk.

Et annet problem med multicast er at antallet adresser er begrenset. Adresserommet fra 224.0.0.0 til 239.255.255.255 (klasse D) er i dag reservert som multicast adresser. Det er ca. 300 millioner IP-multicast adresser. Dette er et stort tall, men det legges fort beslag på tusenvis av adresser i en nett-VO (når IPv6 tar over for IPv4 vil det bli et langt større adresserom for multicast adresser).

Adressene velges tilfeldig, og det er ingen garanti for at en adresse er tatt i bruk i f.eks. en annen nett-VO. For å ha en viss kontroll over hvor multicastpakkene sendes, kan man legge til et felt med en såkalt *time-to-live* (TTL) teller. Dette er et nummer som blir reduseres for hvert ruterhopp. Pakken blir så kastet når TTL er null. Hvis f.eks. en simulering er innenfor et LAN eller et WAN, kan man da unngå at pakkene spres utenfor disse ved å sette en fornuftig, og lav, TTL-verdi. To separate simuleringer som går over to forskjellige WAN kan dermed benytte de samme multicast-adressene selv om WANene ligger geografisk nærme hverandre.

MBone

Multicast Backbone (MBone) er et virtuelt multicast nettverk bygget på toppen av det fysiske Internett for å kunne støtte ruting av IP-multicast pakker. Dette virtuelle nettverket baserer seg på noder som er rutere med støtte for multicast. Disse ruterne er logisk koblet til hverandre via IP ruter - såkalte *tunneler*. MBone ruterne er igjen knyttet til lokale multicast rutere som igjen kan være knyttet til f.eks. et LAN.

Kritiske spørsmål ved simuleringer som går over flere MBones er: hvor mange ruterhopp er det fra endebruker til et MBone, hva er transmisjonstiden mellom forskjellige MBones (kan ligge på 100ms), og hvor mange hopp er det fra det siste MBone til destinasjonsmaskinen. Er linken mellom to MBone belastet ligger ofte raten på pakker som mistes rundt 20 prosent. Noen spillutviklere har sett seg nødt til å lage egne MBone nettverk hvor det kreves at spillere er direkte koblet til disse [4].

1.4.3 Dead Reckoning

I en nett-VO trenger hver endebruker informasjon om andre objekters bevegelser og tilstand - det være seg menneskestyrte eller datastyrte. En bruker kan motta slik informasjon konstant fra hver entitet - dvs. at det sendes ut en oppdateringspakke for hvert bilde som prosesseres (*frame-rate*). Hvis det prosesseres 30 bilder per sekund, så vil det sendes ut 30 pakker per sekund

på nettverket (dette var f.eks. tilfelle i spillet $Doom$). Men hvis en entitet ikke er i bevegelse eller beveger seg slik at man kan forutsi dens oppførsel, fører dette til at det sendes mye unødvendig informasjon over nettverket.

Hensikten med $Dead\ Reckoning$ er å redusere antall pakker som sendes ved å ta i bruk algoritmer som tillater hver endebruker å selv oppdatere tilstander til disse entitetene. Dead Reckoning kan deles inn i to deler: *predikering* og *konvergens/sammenløping*.

Predikering er estimering av fremtidige verdier basert på nåværende eller tidligere oppførsel. Hvis f.eks. et fartøy er i bevegelse så kan hver bruker i en nett-VO selv regne ut fartøyets posisjon ved hjelp av algoritmer som kildeklienten modellerer. Brukeren som styrer dette fartøyet må da holde rede på den faktiske posisjonen og den predikerte. Det settes en feilgrense mellom disse to, og hvis den predikerte posisjonen har et avvik som overskrider den gitte feilgrensen, sendes det ut en oppdateringspakke på nettverket.

Konvergens forteller oss hva vi skal gjøre når vi mottar oppdatert informasjon for å korrigere unøyaktig predikering. Den enkleste måten er at man flytter objektet direkte til dets korrekte posisjon, såkalt *snap-convergence*. Visuelt gir ikke dette et bra resultat, da det resulterer i en hakkete bevegelse. En bedre løsning er da å la objektet bevege seg lineært eller langs en kurve til den ønskede, korrigerte, posisjonen. Objektet fortsetter så med Dead Reckoning etter at posisjonen er korrigert.

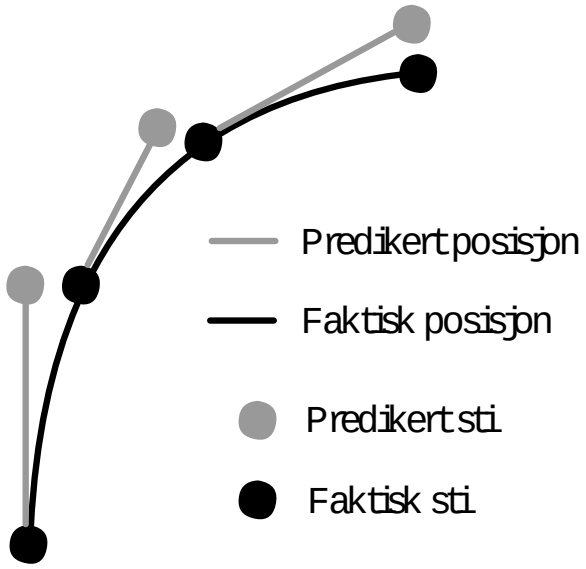
Hvis det det går for lang tid uten at det trengs å sendes ut en oppdatering, blir dette allikevel gjort for å si i fra til de andre brukerne at entiteten fortsatt er "i live" (*keep-alive pakker*). Nye brukere som slutter seg til omgivelsene trenger også informasjon om passive objekter for å kunne vite at de eksisterer. Dette gjøres f.eks. etter fem sekunder.

Dead Reckoning minker også problemet med pakker som går tapt, da et objekt vil bevege seg i henhold til forrige oppdatering. Figur 1.2 på neste side viser hvordan Dead Reckoning fungerer.

1.4.4 Relevansfiltrering

En virtuell omgivelse i stor skala kan inneholde tusenvis av entiteter. Hvis hver endebruker skal motta informasjon om alle disse så vil man etter hvert sprenge nettverkskapasiteten og hver enkelt brukers prosessorkraft.

Som nevnt over må man rette seg etter deltakere med lav nettverkskapasitet når man designer virtuelle omgivelser. Løsningen er at informasjon som ikke er av interesse for hver bruker filtreres bort - såkalt *relevansfiltrering* eller *area of interest management*. I en storskala simulering er det kanskje kun det som finnes innen et geografisk område av rimelig størrelse som er av interesse (et såkalt interesseområde - eng. *Area of Interest* - AoI). En slik relevansfiltrering gjøres ved hjelp av *Area of Interest software*. Denne filtreringen kan skje på mange måter, og man kan gruppere disse under tre kategorier [13]:



Figur 1.2: Eksempel på bruk av dead reckoning

- Serverbasert filtrering
- Senderbasert filtrering
- Regionbasert filtrering

Ved serverbasert filtrering er hver bruker knyttet til en eller flere servere som bestemmer hva som er relevant informasjon for brukeren basert på brukerens interesser. Bruken av sentrale servere til denne prosessen gjør det lettere å skape konsistens i en nett-VO siden disse sitter inne med all informasjonen i omgivelsen.

Ved senderbasert filtrering er det senderen som avgjør hva som er relevant informasjon for andre brukere. En svakhet ved denne tilnærmingen er at hver sender må ha kjennskap til andre brukeres interesseområder.

Den siste kategorien, regionbasert filtrering, baserer seg på at en nett-VO deles inn i geografiske regioner. Den virtuelle verdenen deles inn i *interessesoner* hvor en entitets interessesone bestemmes av hvor den befinner seg geografisk, og den ønsker å motta informasjon basert på dette. En løsning er å dele området inn i sekskantede, like store, felt. Dette har blant annet blitt beskrevet for NPSNET-IV, men aldri implementert. Nettverksspill benytter seg per i dag av geografiske soner som interesseområde hvor en server typisk administrerer en sone (region). Spillere kommuniserer så med hverandre via serveren (unicast). Alternativt så kan sonene representere en multicast gruppe; Brukeren abonnerer på multicast adressen som er dedikert til son-

en han eller hun befinner seg i, og sender samtidig ut oppdateringer om sin integrasjon til adressen (som da alle andre brukere innenfor sonen mottar).

1.5 Mål for denne oppgaven

Målet for denne oppgaven er å presentere en mulig løsning til hvordan relevansfiltreringen kan gjøres i en nettverksbasert virtuell omgivelse. Hvordan *interesseområdet* kan defineres for en gitt klient - det vi vil kalle *avstandsfiltreringen* - er hovedtema. Hensikten med denne avstandsfiltreringen er å holde antall *peer-to-peer*[16] forbindelser på et minimalt nivå. Dette behandles i neste kapittel. Vi vil se på eksisterende løsninger og ideer, og ut i fra dette presentere et nytt forslag til avstandsfiltreringen: bruken av *dynamiske soner* og *dynamisk fokus og aura*, noe som hovedsakelig bygger på, eller er modifiseringer/vidreutviklinger av, metoder som allerede er foreslått innen akademia. Det nye med dette forslaget er at det *kombinerer* de to ovennevnte begreper. Vi vil også se på simuleringsresultater av en implementasjon for å teste prinsippene for denne avstandsfiltreringen.

Etter at vi har definert en gitt klients interesseområde vil vi også se på ideer og mulige løsninger for å foreta ytterlige filtreringer av klienter som er innenfor en bestemt klients interesseområde - *interessefiltreringen*, hvor formålet er å rangere klientene man har et felles interesseområde med slik at man kan motta færre oppdateringspakker fra klienter som er av lav interesse.

Forslaget til relevansfiltrering som presenteres i denne oppgaven er ment til å kunne benyttes for dagens nettverksspill. Derfor må de begrensninger Internett har taes i betraktning - her vil da det begrense seg til den tilgjengelige båndbredde en simulering har. Innenfor forskning og akademia betraktes gjerne multicast som et viktig aspekt for relevansfiltrering, noe som denne oppgaven ikke vil forsvare, da Internetts støtte for multicast ikke er tilfredsstillende for kommersielle applikasjoner pr. i dag. Men det er derimot viktig å betrakte multicast som en fremtidig løsning.

[16] I mangel på en god oversettelse vil denne oppgaven benytte den engelske termen *peer-to-peer* for kommunikasjon direkte mellom to klienter over et nettverk.

KAPITTEL 2

Avstandsfiltrering

2.1 Innledning

Hva denne oppgaven angår vil en virtuell omgivelse være ensbetydende med en simulering av en 3-dimensjonal verden som spenner seg over et større geografisk område. Tusentalls av entiteter skal kunne bevege seg gjennom et stort virtuelt landskap og kunne interagere med hverandre. Siden vårt sanseapparat er begrenset, vil det på grunn av dette være et ganske lite område rundt oss hvor vi vil ha en fornemmelse for andre objekter - de fleste avatarer og gjenstander vil befinne seg utenfor sansbar rekkevidde. Det kommer selvsagt an på faktorer som f. eks. størrelse, men objekter av vår egen størrelse (som f. eks. andre mennesker) vil vi sjelden ha noe fornemmelse av hvis de befinner seg flere hundre meter borte fra oss. Unntak er selvfølgelig hvis man benytter noe for å forsterke sansene (en kikkert vil jo være til hjelp for bedre å se noe på avstand), eller hvis gjenstanden/personen skiller seg betraktelig ut fra omgivelsene (f. eks. et menneske kledd i svart på en snøvidde).

Men selv innenfor det felt hvor sansene mottar sterke nok signaler til at vi kan hente ut nøyaktig og meningsfull informasjon om et objekt, vil det ikke nødvendigvis være ønskelig å skulle bruke ressurser på å prosessere sanseinntrykkene fra det gitte objekt.

Ut i fra dette vil det da være hensiktsmessig å gjøre filtreringen av informasjon mellom klienter todelt:

- *Avstandsfiltrering*: Objekter man ikke kan sanse fordi deres avstand er for stor filtreres bort. Det naturlige her vil da være å geografisk dele inn omgivelsene i soner på fornuftige størrelser. Et subsett av disse sonene blir dermed grunnlag for hva som er av potensiell interesse. Dette kapitlet tar for seg bl. a. hvordan en slik soneinndeling kan foretas.

- *Interessefiltrering*: Filtrering av hva som er av interesse av det man faktisk kan sanse - m. a. o. en filtrering av det som befinner seg innenfor det fysisk sansbare felt (de klienter og objekter man sitter igjen med etter avstandsfiltreringen). Dette temaet behandles i neste kapittel.

2.1.1 Sentralisert og distribuert arkitektur

Nettverksarkitekturen til en nett-VO kan enten være en klient-server-arkitektur, en *ren* peer-to-peer-arkitektur eller en *hybrid* peer-to-peer-arkitektur som kan betraktes som en mellomting mellom de to førstnevnte. Den store forskjellen mellom en klient-server og en ren peer-to-peer-arkitektur er at hver spiller må holde rede på hele spillets tilstand (eng. *game state*) for peer-to-peer-arkitekturen. For en klient-server-arkitektur er det kun serveren som har ansvaret med å til en hver tid ha den korrekte tilstanden til hele spillet. Dette gjør at klient-server-arkitekturen blir *sentralisert*, mens peer-to-peer-arkitekturen får en *distribuert* topologi.

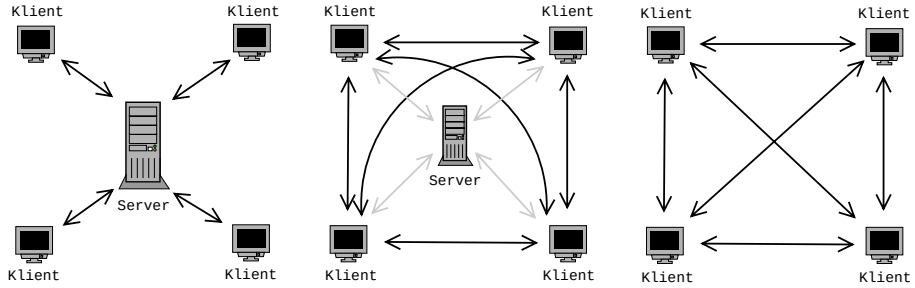
Problemet for løsningen med en sentralisert server er at serveren blir det svake punktet - prosessorkraften og kommunikasjonslinjene begrenser hvor mange klienter den kan støtte. Det viser seg at man når disse begrensningene fort selv om det investeres i servere i million-dollar-klassen [23].

Forskjellen på en ren og en hybrid peer-to-peer-arkitektur er at det eksisterer en sentral enhet for den hybride, noe som per definisjon ikke eksisterer for den rene. Denne sentrale enheten fungerer mer som en database eller katalog klientene kan foreta forespørsler overfor med den hensikt å lokalisere hvor tjenesten ligger[1].

Figur 2.1 på neste side viser forskjellen på en klient-server-arkitektur, en hybrid peer-to-peer-arkitektur og en ren peer-to-peer-arkitektur. For klient-server-løsningen sender alle klientene oppdateringspakker til serveren - hvor pakkene blir prosessert, validert og videre distribuert til de klientene oppdateringene er myntet på. For peer-to-peer-arkitekturen sender hver klient oppdateringene direkte til de andre klientene. Kommunikasjon foregår også direkte mellom klientene for den hybride løsningen, men det eksisterer også en sentral server hvis ansvarsområde er minimalt sammenlignet med serveren for en klient-server-arkitektur.

På grunn av begrensningene relatert til en sentralisert arkitektur vil den arkitekturen som presenteres i denne oppgaven være en *hybrid* peer-to-peer-arkitektur - dvs. at den forsøker å redusere belastningen på serveren ved å gjøre arkitekturen mest mulig distribuert. Dette kan oppnås ved å la visse typer kommunikasjon gå direkte mellom klientene. Vi vil adressere dette senere i kapitlet.

[1] Fildelingsapplikasjonen *Napster* benyttet en slik hybrid løsning hvor klienter via en sentralisert database kunne finne de klienter hvor filene man ønsket var lokalisert. *Gnutella* protokollen (som også er ment for fildeling mellom klienter) baserer seg derimot på en ren peer-to-peer-arkitektur.



Figur 2.1: Klient-server arkitektur til venstre, den hybride peer-to-peer-arkitekturen i midten og den rene peer-to-peer-arkitektur til høyre.

2.1.2 Problemer relatert til peer-to-peer-arkitekturen

Hovedproblemet med en peer-to-peer-arkitektur er at den ikke skalerer særlig bra. Belastningen pr. klient øker eksponensielt for hver klient som kommer til en simulering; hvis hver enkelt klient skal ha forbindelse til, og sende de samme oppdateringene til, alle andre klientene. Dermed blir hovedmålet for relevansfiltreringen i første omgang å holde antall klientforbindelser så lavt som mulig for hver klient. Før vi behandler dette temaet videre vil vi adressere to andre områder hvor peer-to-peer-arkitekturen står overfor større problemer sammenlignet med en klient-server arkitektur: *juksing* og *synkronisering*. Disse to områdene er ikke relevante for selve problemstillingen til denne oppgaven, men siden vi her vil forsvare bruk av en peer-to-peer-arkitektur vil de kort presenteres; mer som argumenter for at en peer-to-peer-arkitektur kan benyttes da de relaterte problemene (juksing og synkronisering) lar seg løse.

Juksing og online-spill

Det å jukse i, eller *hacke*, et data-spill er et fenomen som eksisterte lenge før online-spillenes tid, men det er først når spillere begynner å jukse i disse spillene at det kan kalles et problem siden det da vil gå utover andre spillere. Ved å benytte en sentralisert server er det lettere å forhindre juks ved at serveren kan validere alle oppdateringspakker. Dette er en viktig grunn til at spillutviklere i dag velger en sentralisert topologi.

Begge modellene for kommunikasjon (klient-server og peer-to-peer) er sårbare for hacking [22]. Et eksempel kan være at man automatiserer eller forsterker sine reflekser (som vil være fordelaktig i spill hvor hurtig reaksjon er avgjørende), eller at man går inn i koden og endrer verdier for sin karakters variable (slik at man blir sterkere, rikere, raskere o.l.)[2].

[2]Ta f. eks. spillet *Diablo II*. Her er en karakters data lagret i lesbare filer på hard-disken, og man kan endre verdier i disse med en hex editor. Slik kan man enkelt gjøre sin karakter uovervinnelig.

For en peer-to-peer-arkitektur finnes det ikke en server til å validere en klients oppdateringer eller interaksjoner med omgivelsene. Man får en situasjon der "klienten alltid har rett" - sender en klient ut informasjon til andre klienter angående sine bevegelser eller verdier (*hit-points*, *spell-points* e.l.), så må dette hos de andre klientene betraktes som rett informasjon. En løsning på dette problemet kan være å endre en klients egne kommandoer til *kommando-forespørsler*. I stedet for at hver enkelt klient sender ut oppdateringer som de andre klientene må prosessere uten å kunne validere, så kan man heller betrakte disse oppdateringene som forespørsler. Hver klient vil så ha en intern kommando-kø (også for sine egne kommandoer). Hver kommando-forespørsel som hentes ut fra køen må så valideres før den prosesseres. Dermed vil en klients kommando sendes ut til alle klienter den måtte berøre, og disse klientene (så vel som klienten som kom med kommandoforespørselen) må validere denne. Dette krever at hver klient kjører en full kopi av hele spillets simulering (eller den delen av omgivelsene klienten måtte befinne seg i) - hver oppdateringspakke kan dermed valideres opp mot denne.

Synkronisering

I spill hvor all interagering må valideres av serveren vil så denne ha full kontroll over synkroniseringen i spillet. Siden serveren har oversikt over alle klientene, vil den kunne bremse ned hastigheten til selve spillet for kortere perioder hvis en av klientene skulle begynne å "henge etter". Dette oppleves da som forsinkelse, eller hakking hos de andre klientene (eng. $lagging$).

Skal man så tillate kommunikasjon direkte mellom klienter trengs andre metoder for å ivareta spillets overordnede synkronisering. Et eksempelt på dette presenteres i [24] hvor det benyttes en teknikk som kalles *bucket synchronization*. Ved å benytte tidsluker på 100ms hver vil alle oppdateringer assosiert med den samme tidsperioden (samme *bucket*) prosesseres samtidig. Metoden benytter en global klokke for å evaluere forsinkelsen mellom klientene som deltar i simuleringen.

2.1.3 Mulige løsninger for avstandsfiltrering

Som nevnt i innledende kapittel (se seksjon 1.4.4) kan relevansfiltrering, eller *Area of Interest Management* (AoIM), deles inn i tre kategorier: *Serverbasert, senderbasert* og *regionbasert*. Det vanlige i dagens nettverksspill er at man deler inn den virtuelle omgivelsen i felt eller soner hvor hjørnene defineres som punkter i et kartesisk koordinatsystem[3]. Hver klient vil så opprette

[3]Et kartesisk koordinatsystem har for en 2D verden to akser (xy) og for en 3D verden tre akser (xyz). Dette passer hvis man tenker seg en virtuell omgivelse som en flate eller som en kube. Et alternativ er å benytte et kule-basert koordinatsystem hvis den virtuelle omgivelsen er definert som en kule.

unicast forbindelse til serveren som har ansvaret for sonen - slik at all kommunikasjon går via serveren (sentralisert). Alternativt så kan klientene som befinner seg innenfor sonen oppretter peer-to-peer-forbindelse seg imellom (blanding av regionbasert og senderbasert filtrering - det kan først gjøres en grovfiltrering basert på hvilken region man befinner seg i, dernest kan klientene seg imellom definere hvor interessante de er overfor hverandre). Et tredje alternativ kunne vært at hver sone blir assosiert med en multicast-adresse hvorpå klientene som befinner seg i sonen kommuniserer via denne. Mangelen på multicast-støtte for rutere i Internett gjør at denne ideen ikke lar seg realisere per dags dato.

For dagens nettverksspill er det arkitekturen med sentralisert server og bruk av unicast som er gjeldende (som vist til venstre i figur 2.1). Derfor er det begrenset hvor mange som til enhver tid kan være koblet til en spill-server; $EverQuest$ støtter ikke flere enn 1,900 brukere samtidig på sine servere [23].

Problemet med å dele inn omgivelsene i soner er at det fort vil bli en ubalanse i antall entiteter pr. sone. Det vil ofte forekomme konsentrasjoner av entiteter (klynger) - hvis f. eks. flere hundre befinner seg innenfor en sone er det lite sannsynlig at alle er av relevans for deg og man vil belaste nettverket unødvendig mye ved at man sender informasjon til og mottar informasjon fra flere klienter enn nødvendig. Målet er å kunne fordele entitetene jevnt på feltene og i første innstans foreta en grov-filtrering på nettverksnivå. Før vi ser på hvordan dette kan løses ser vi på hvordan avstandfiltrering, eller soneinndelingen, gjøres for noen akademiske og kommersielle applikasjoner.

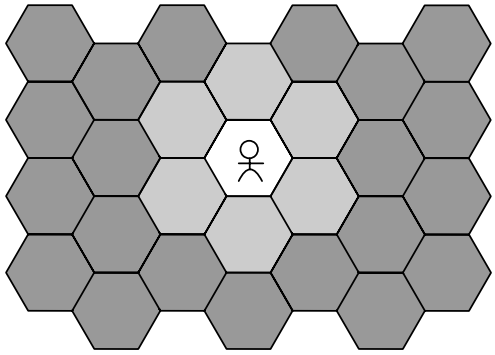
Hexagonale felter i NPSNET

I $NPSNET$ (se avsnitt 1.3.2 på side 4) er det foreslått at omgivelsene deles opp i like store hexagonale soner. En entitet er aktiv medlem av sonen den befinner seg i og passiv medlem av sonene som ligger inntil denne. Hver sone har en multicast-adresse knyttet til seg, og entiteter med felles soner kommuniserer så via disse. Medlemsskap til multicast-gruppene administreres av den klienten som er det eldste aktive medlem av sonen. Denne klienten vil så ha ansvaret for å legge til nye medlemmer som kommer til og slette medlemmer som forlater sonen (man slipper m. a. o. en sentralisert enhet for å håndtere dette). Figur 2.2 på neste side viser hvordan en klient forholder seg til en hexagonal soneinndeling.

Locales i SPLINE

Et alternativ til å dele en allerede implementert verden opp i soner, er å bygge opp en virtuell verden ved å sette sammen små verdener som er designet uavhengig av hverandre - man lar sonene definere omgivelsene og ikke omvendt. Denne tankegangen benyttes i systemet $SPLINE$[4] hvor den virtuelle

[4]Scalable Platform for Large Interactive Networked Environments



Figur 2.2: Hexagonal soneinndeling for NPSNET. Entiteten er aktiv medlem av hvit sone og passiv av de lyse grå sonene. De mørke sonene faller utenfor klientens interessefelt.

omgivelse er satt sammen av $locales$[5] hvor hver av disse representerer en sone [17]. Disse sonene prosesseres separat fra hverandre. En klient vil typisk befinne seg i kun en sone, men skal kunne interagere med flere samtidig, og fritt bevege seg over i andre soner. En sone assosieres med et sett av multicast-adresser for å kunne håndtere ulike typer av informasjonsutveksling (f. eks. lyd data og visuell data).

Det spesielle med denne løsningen er at sonene følger naturlige grenser og at disse har egne lokale koordinatsystem. En boligblokk vil f. eks. kunne representere en slik sone hvor ytterveggene definerer de naturlige grensene. Denne sonen vil så dynamisk kunne deles inn i sub-soner - etasjer (som igjen kan deles inn i rom osv.). Vi får dermed en naturlig hierarkisk soneinndeling.

Denne soneinndelingen vil effektivt filtrere bort de deler av omgivelsene som ikke er av interesse for en gitt klient (f. eks. etasjen under), men den løser ikke problemet med ubalanse mht. prosesseringsbelasting sonene seg imellom.

Metoder i dagens større nettverksspill

I spillene $Ultima\ Online$, $EverQuest$ og $Asheron's\ Call$ gjøres soneinndelingen ved at man har en klynge av servere [18], hvor hver server definerer grensene i spillet (en server har ansvar for en sone - m. a. o. en ganske grov oppdeling i hva sone størrelsen angår). En klient blir dermed overført til en annen server når han beveger seg mellom de forskjellige sonene.

Problemet med å unngå overbelasting hos enkelte servere løses i $EverQuest$ ved såkalt *portal storming*. Spillere blir automatisk overført (teleportet) til en annen, tilfeldig valgt, server som ikke er like tungt befolket av spille-

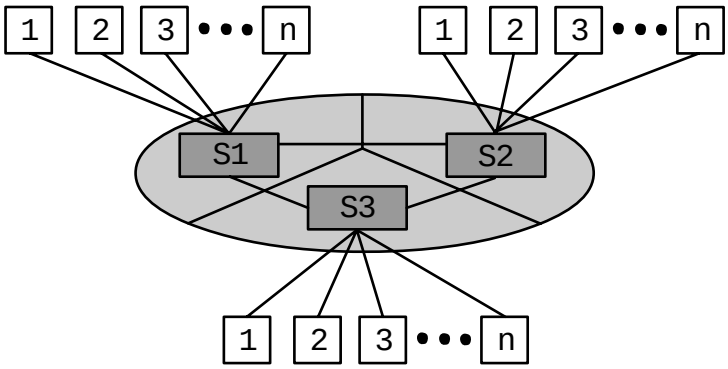
[5] Termen $locales$ kommer av at hva som kan observeres av en enkelt bruker til en hver tid er av en lokal natur.

re. Klientene plukkes ut etter FIFO prinsippet (*First-in-first-out*) - de som har vært lengst på serveren må ut først. Her har man da løst klyngeproblemet ved å *bake* det inn i selve handlingen i spillet.

Det å forflytte seg mellom to soner vil trigge en overflytting til en annen server. Hvis man i spillet faktisk kan bevege seg over en sone-grense i sann-tid, vil denne overflyttingen oppleves som en unaturlig pause for spilleren. I $Ultima\ Online$ og $Asheron's\ Call$ speiler man innholdet langs sonegrensene i nabo-serverne slik at overgangen ikke skal kunne merkes [20].

I $Asheron's\ Call$ er det muligheter for å dynamisk overføre en del av en sone, hvis server er tungt belastet, til en annen server som er mindre belastet for å jevne ut antall klienter pr. sone.

Figur 2.3 viser hvordan dagens nettverksspill deler inn omgivelsene ved å la en server representere et statisk sub-område for simuleringens omgivelser. En enkelt server kan til enhver tid støtte n klienter, hvorpå hele spillet samtidig kan håndtere <*n x ant. servere*> brukere.



Figur 2.3: Kluster av tre servere som administrerer hver sin sone i en virtuell omgivelse (oval sirkel). Hver server kan ha opptil *n* klientforbindelser.

2.2 Dynamisk soneinndeling

Som nevnt ovenfor er problemet med å dele inn en virtuell omgivelse i regioner at man ikke kan forutse hvor det kan forekomme klynger av entiteter. H. A. Abrams [13] har kommet med en løsning på dette problemet ved å ta i bruk soner som dynamisk forandrer størrelse for slik å kunne fordele klientene jevnt på sonene.

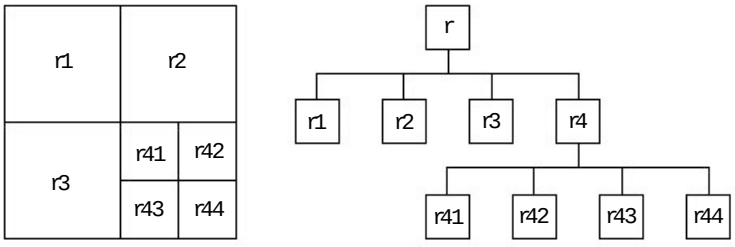
Først må det for en gitt klient defineres et *interesseområde* som vi kan betrakte som et sub-område av en virtuell omgivelse hvis hensikt er å skulle filtrere bort klienter og objekter som ikke er av interesse. Det ønskelige resultat vil da være at hver enkelt klient får et minimalt sett av klienter å sende

oppdateringspakker til slik at pakketrafikken sett under ett vil reduseres.

En brukers interesseområde blir først bestemt ved en kule hvor radius bestemmer den maksimale distanse av interesse. Hvis det er snakk om en 3D-verden vil så interesseområdet bli kuleformet. For en 2D-verden vil interesseområdet defineres ved en sirkel. Området deles så inn i like store, kubeformede felt (3D) eller kvadratiske felt (2D). De felt som helt eller delvis dekkes av denne interesse-kulen/sirkelen er gjenstand for hva som betraktes som brukerens interessefelt - de felt man ønsker å motta informasjon om og sende informasjon til. I henhold til Abrams er det m. a. o. de objekter/klienter som befinner seg i sonene som faller inn under en gitt klients interesseområde som definer klientene/objektene man vil utveksle informasjon med. Hvis et felt har en konsentrasjon av entiteter, kan det splittes opp i åtte eller fire felt (alt ettersom det er en 3D- eller en 2D-omgivelse), hvis mål er å få en balanse mellom entiteter pr. felt.

En virtuell omgivelse får dermed en trestruktur - den blir dynamisk i den forstand at den er rekursiv. Når et felt splittes opp blir dette ekvivalent med at en foreldrenode får åtte (3D) eller fire (2D) bladnoder (sub-soner). Det er disse bladnodene som til enhver tid er de *aktive* felt. Hvis antall entiteter i samtlige bladnoder til en foreldrenode kommer under en gitt terskel, vil man så kunne terminere alle bladnodene. På rekursivt vis vil så foreldrenoden igjen bli en bladnode - et aktivt felt. Vi kan m. a. o. både splitte opp felt og slå sammen felt. Vi starter med en rotnode, og etterhvert som entiteter kommer til eller flytter på seg bygger vi rekursivt opp et tre hvor det til enhver tid er bladnodene som definerer feltene og hvor entitetene er jevnt fordelt mellom disse. Oppsplitting i åtte sub-soner gjelder som sagt for en 3D verden, mens en 2D verden vil splittes opp i fire sub-soner (se figur 2.4).

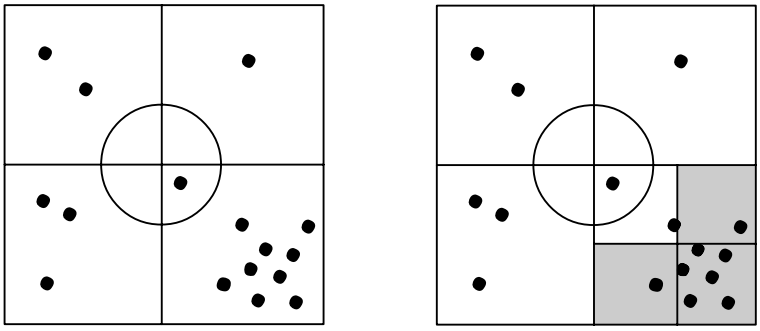
Ved å bestemme en maksimal dybde for treet gis det samtidig en minstestørrelse på sonene. Forlater alle klientene simuleringen, sitter man igjen med en sone - rotnoden.



Figur 2.4: Til venstre ser vi hvordan en verden dynamisk deles inn i felter. Disse feltene tilsvarer treet til høyre. I dette eksemplet splittes feltene i fire, noe som vil være typisk i en 2D verden. Treet blir da tilsvarende med fire bladnoder pr. foreldrenode.

Det er når interessekulen kun har delvis overlapp med et felt hvor kon-

sentrasjonen av entiteter er stor at den dynamiske soneinndelingen har sin styrke. Ved å splitte opp dette feltet vil flere av de nye feltene ikke lenger være innenfor interesseområdet, siden entitetene nå blir fordelt på de nye subsonene (som tar over foreldrenodens jobb som aktivt felt). Brukeren vil så ha færre entiteter å forholde seg til og bruken av båndbredde vil minske som følge av at det blir færre å kommunisere med. Figur 2.5 viser et eksempel på dette.



Figur 2.5: Til høyre vises situasjonen etter splitting. Entitetene i de grå sonene faller da utenfor interesseområdet som bestemmes av de sonene sirkelen helt eller delvis dekker.

2.3 Dynamisk fokus og aura

I avsnittet over introduserte vi begrepet interessekule eller interessesirkel som del av prosessen med å definere en klients interesseområde. Vi vil her ta to aspekt i betraktning for å kunne bestemme hva som gjør klienter interessante for hverandre - eller mer korrekt; hva som gjør klienter observerbare overfor hverandre, siden det her er avstandsfltreringen det er snakk om. De to aspekter er:

- *Hva er vi i stand til å oppfatte?* Dette går da på sanseapparatet, erfaring, kunnskap og forståelse. Disse faktorene tilsammen vil kunne bestemme hva brukeren skal motta av informasjon fra resten av en virtuell omgivelse.

- *Hvordan blir vi oppfattet av andre?* Kriterier her kan være i hvor stor grad man skiller seg ut fra resten av omgivelsene, hvor flink man er til å påkalle oppmerksomhet (hvis man f.eks. er en kjent person) og hvor stor karisma eller utstråling man har.

I [19] benytter man begrepet $aura$ som defineres som et område rundt et objekt - et område for potensiell interaksjon. Denne auraen kan være av forskjellig størrelse og fasong alt etter hva slags $medium$ den er en aura for;

hvis f. eks. mediet er lyd, vil to objekt kunne snakke sammen hvis deres auraer overlapper hverandre.

I [20] presenteres et lignende konsept som kalles $viewer\ scope$ og $object\ scope$. Viewer scope forteller hvor langt en klient kan se, mens object scope forteller hvor synlig et objekt[6] er, hvor utstrekningen er proporsjonal med størrelsen på objektet. Her vil en overlapp mellom en klients viewer scope og et objekts object scope avgjøre om klienten ser objektet. Det samme gjelder for det ovenfornevnte aura-begrepet, hvor det er overlapp mellom selve auraene for to klienter som trigger en interesserelasjon disse imellom.

I denne oppgaven vil vi presentere og ta i bruk termene $dynamisk\ fokus$ og $dynamisk\ aura$ for å bestemme hvilke klienter som er sansbare for hverandre - hvilke klienter som potensielt har en gjensidig interesse for hverandre. Dynamisk fokus og aura baserer seg på de to kriteriene ovenfor (hva vi er i stand til å oppfatte og hvordan vi blir oppfattet av andre).

Hver klient har én fokus og én aura. Disse er representert som kuler eller sirkler (alt ettersom vi snakker om en 3D- eller en 2D-verden), hvor radius definerer "utstrekningen" av fokus og aura.

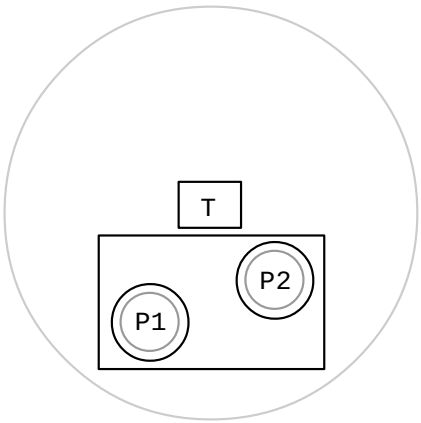
- *Fokus* bestemmer hvor godt utviklet våre evner er til å hente inn ekstern informasjon, og defineres da i henhold til kriteriene ovenfor om hva vi er istand til å oppfatte. En *dynamisk* fokus vil så kunne gjenspeile *hvor* fokusert en klient til enhver tid er.

- *Aura* bestemmer hvordan andre oppfatter en gitt klient. Denne vil da være stor hvis man skiller seg ut eller har en sterk utstråling. Auraens dynamiske natur vil dermed kunne reflektere hvor mye klienten til enhver tid skiller seg ut fra omgivelsene.

På dette nivået er det i hovedsak snakk om å filtrere bort det vi ikke kan *sanse* rent fysisk - vi ønsker å definere det geografiske området en entitet dekker med sin fokus. Det dreier seg om den energien (lys og lyd) vi kan oppfatte av objekter rundt oss som er den spesifikke delen av en persepsjonell prosess - signaler som er sterke nok til at de for oss er sansbare. Den abstrakte delen av en slik prosess - hvor vi filtrerer eller graderer sansbar input basert på graden av relevans - er neste trinn i å definere interesseområdet, og blir behandlet i kapittel 3.

Et eksempel på konseptet om en dynamisk fokus og aura er hvis en stor forsamling av deltakere er samlet på et lukket sted i en virtuell omgivelse (se figur 2.6 på neste side). Vi har en stor sal med mange tilskuere og en taler på et podium. Hver enkelt publikummers fokus vil da kanskje være liten, siden det er tett med folk og man kun greier å observere detaljer ved personer i umiddelbar omkrets. Det samme gjelder auraen - du observeres kun av per-

[6] Med objekt menes da her objekter som ikke styres av menneskelige brukere i en virtuell omgivelse.



Figur 2.6: Sirklene viser auraene til taler, T (den store sirkelen), og fokus og aura til to tilskuere, P1 og P2 (den store firkanten representerer området hvor tilskuerne befinner seg).

sonene rundt deg. Man registrerer alle individene heller som en del av en forsamling.

Taleren derimot har en stor aura, siden han får stor oppmerksomhet i egenskap av å være en taler og siden han høres godt gjennom et høyttaleranlegg. Hans aura kan da deles i to undergrupper: oppmerksomhet og lyd. Den store auraen gjør at han sender informasjon til alle i salen, og publikum oppfatter dette ved at det eksisterer et felles interesseområde mellom taleren og hver av de to tilskuerne (siden talers aura fullstendig dekker P1's og P2's fokus må det eksistere et felles interesseområde). Men hvis lydanlegget bryter sammen, så vil lyd-delen av auraen minske betraktelig - den dekker kun noen på første rad. En publikummer i salen reiser seg og begynner å rope. Da vil hans aura bli større, mens den til taleren kanskje minker enda mer siden oppmerksomheten blir tatt vekk fra ham.

2.4 Kommunikasjon basert på felles interessesoner

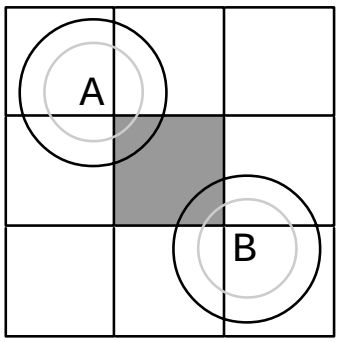
Etter å ha presentert prinsippene med dynamisk soneinndeling og dynamisk fokus og aura, vil vi så se på hvordan vi i denne oppgaven vil betrakte problemet med å definere et felles *interesseområde* mellom to klienter. Som tidligere nevnt vil et slikt felles interesseområde for to klienter tilsi at kommunikasjon vil være ønskelig mellom disse fordi de er sansbare for hverandre.

For denne oppgaven er det en *kombinasjon* av dynamisk soneinndeling og dynamisk fokus og aura som bestemmer et felles interesseområde for klienter. Både fokus og aura vil typisk helt eller delvis dekke en eller flere soner - hver klient vil ha en liste av fokus-soner og en liste av aurasoner.

Sonene som inngår i disse listene vil måtte oppdateres under en simulerings kjøretid ettersom:

- *Klienten beveger seg gjennom omgivelsene*. Når en klient flytter seg vil dette kunne resultere i at soner forlates av aura og/eller fokus, eller nye soner kommer i fokus og/eller innlemmes i aura.
- *Sonetreets topologi forandres*. Hvis f. eks. en sone som en klient har i fokus blir splittet opp vil da en eller fler av sub-sonene kunne falle ut av fokus (gitt at foreldre-sonen kun delvis overlappes av fokus).
- *Fokus eller aura forandrer størrelse*. Siden fokus og aura er dynamiske av natur vil dette kunne resultere i at soner faller utenfor eller må legges til fokus og/eller aura listene.

Dermed kan vi si at det eksisterer et felles interesseområde mellom to klienter dersom den ene klienten dekker en gitt sone med sin fokus og den andre klienten dekker den med sin aura. En slik felles *interessesone* avgjør også om informasjonsutvekslingen er toveis eller enveis - om begge er synlige overfor hverandre eller om det kun er den ene klienten som ser den andre klienten. Klienten som dekker sonen med sin fokus ser klienten som dekker den med sin aura. Hvis begge dekker sonen med både fokus og aura har vi en situasjon hvor kommunikasjonen er toveis. Figur 2.7 viser to klienter med en felles interessesone hvor begge ser hverandre.



Figur 2.7: Den grå sonen viser interesseområdet mellom klient A og klient B.

Det er da dette samvirke mellom den dynamiske soneinndelingen og en dynamisk fokus og en dynamisk aura som gjør at avstandsfiltreringen som presenteres i denne oppgaven kan betraktes som et nytt alternativ for relevansfiltrering. Fokus og aura slik her definert skiller seg fra [19] (som har et aura-begrep) og [20] (*viewer scope* og *object scope*) ved at:

- Aura-begrepet i [19] ikke er ment som del av en relevansfiltrering men mer som en mekanisme for å håndtere samtaler mellom klienter med

fokus på interagering mellom en gruppe av deltakere - f. eks. virtuelle konferanser og møter.

- $Viewer\ scope$ kun assosieres med klientene og at $object\ scope$ kun assosieres med objekter, samt at begge er av en statisk natur. Viewer scope og object scope er foreslått brukt for en ren klient-server arkitektur hvor målet er å minske belastningen på serveren (siden denne administrerer alle objektene). Begrepene settes ikke i relasjon til kommunikasjon og relevansfiltrering klientene seg i mellom.

2.4.1 Definere radius til fokus og aura

Figur 2.8 på neste side viser hvordan man kan definere fokus og aura til en klient. Aura vil kunne bestemmes av fysiske egenskaper, som typisk vil være størrelse, overflate (farge) og lyd. Dette må så settes i forhold til omgivelsene entiteten befinner seg i. En stor entitet omgitt av mindre objekter er mer synlig enn om den er omgitt av like store entiteter. Overflate, eller farge forteller også hvor synlig en entitet er - om man glir inn i omgivelsene eller ikke. Et eksempel på dette kan være en hare. Om haren hadde hatt hvit pels om sommeren og brun om vinteren ville auraen vært større en hva den faktisk er. Egenskaper ved et objekts omgivelser som svekker dets utstråling (m. a. o. aura) kalles gjerne *konkurrerende stimuli*[7].

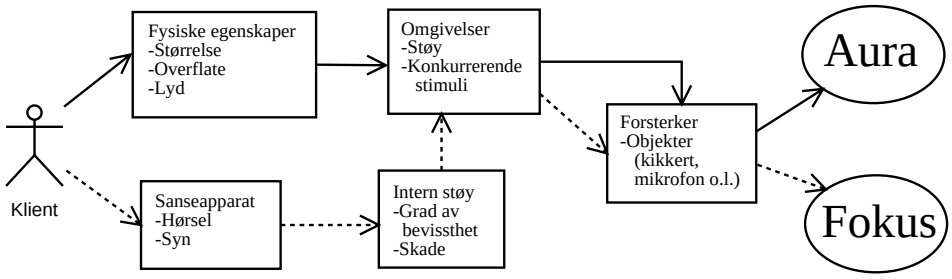
Lyden en entitet frembringer vil også vekke mindre oppmerksomhet om den omgis av entiteter som frembringer lyd som er overdøvende. Atmosfæriske omgivelser vil også dempe en aura - man er ikke like synlig i tåke, tett snøvær og i mørket. I tillegg til faktorer som demper en aura vil man også kunne benytte objekter for å forsterke den. Taleren i eksemplet ovenfor benytter en mikrofon for å forsterke sin aura. Samtidig kan også podiet taleren sto på fungere som en forsterker; generelt kan vi si at bestemte punkter en klient befinner seg på i den virtuelle omgivelse vil kunne ha innvirkning på (forsterke) aura (podiet kan betraktes som et objekt, men også omgivelser av mer statisk karakter som f. eks. en haug vil også kunne gjøre klienten mer synlig - altså forsterkende for aura).

Fokus defineres i første innstans av hvor godt utviklet sanseapparatet er[8]. Dernest må man ta i betraktning indre støy. Hvor fokusert, eller bevisst er man - hvor våken. Hvis man f.eks. er skadet vil sanseapparatet kunne sløves

[7] Dette refererer til hvor "rent" et signal er - hvor fri det er for konkurrerende stimuli eller støy (de signaler, interne eller eksterne, som ikke er gjenstand for det vi ønsker å vie vår oppmerksomhet til).

[8] Vi tar her utgangspunkt i *en* fokus som er et produkt av hele sanseapparatet, men man kunne like gjerne definert sub-kategorier hvor man delte fokus i en for hørsel og en for syn. Man kunne da også hatt en todeling av auraen - hvor godt man synes og hvor godt man høres. Dermed ville man kunne ha to interesseområder: et visuelt, og et basert på lyd. Hvis man f. eks. befinner seg bak en glassvegg vil dette da kun gå utover lyd-fokusen, mens en tett tåke i større grad vil minske den visuelle

som følge av dette. Til slutt vil også her de eksterne omgivelser virke inn - f. eks. så påvirker graden av lys hvor langt man kan se. På lik linje som med auraen vil man kunne forsterke fokus ved hjelp av eksterne objekter. En kikkert vil typisk øke synsrekkevidden.



Figur 2.8: En enkel figur som viser hvordan fokus (ved å følge de stiplete pilene) og aura (ved å følge de sorte pilene) kan defineres.

Initielt vil man definere radius til fokus og aura ved hjelp av personlige variable. Disse vil så justeres av omgivelsesvariable. Man må så finne fornuftige verdier for hvor mye man f. eks. skal minske fokus ved svekkede lysforhold, og hvor ofte man skal foreta justering av fokus og aura.

Her må man begynne å skille mellom den virkelige verden og en virtuell verden. De fleste spill og simuleringer som forsøker å gjenskape en virkelighet er gjenstand for klare fysiske begrensninger. Ta f. eks. et online-flerbrukerspill; her vil den største begrensningen ligge i at brukeren interagerer med den virtuelle omgivelsen via en monitor som typisk vil være mellom 15" og 21". Den grafiske oppløsningen til en slik skjerm vil selvfølgelig ikke kunne måle seg med hvordan øyet ser den virkelige verden. Ser man forskjell på to objekter på en gitt avstand i virkeligheten, er det ikke sikkert at man kan oppfatte dette på en skjerm, da detaljer ikke kan gjengis mer "finkornet" enn på pixel-nivå[9].

Man må så ta i betraktning den spesifikke simuleringen som er gjenstand for relevansfiltrering. F. eks. så vil forskjellige grafikk-motorer[10] gi ganske så forskjellige visuelle opplevelser. Mindre detaljrikdom og simplere grafikk vil f. eks. gjøre det lettere å skille objekter fra hverandre (kontrastene vil bli større). Hvordan atmosfæriske effekter som lys, tåke, regn o.l. er implementert vil bety mye for hvor godt man kan "se" i en virtuell omgivelse.

[9]En teknikk som kan benyttes for å redusere ressursene som kreves for å rendre et objekt på lang avstand er å la objektet bli representert med et redusert antall polygoner. Detaljrikdommen blir da mindre (eng. *level-of-detail*). Dette kan så føre til at det blir vanskeligere å kunne se hva et objekt er.

[10]Kalles 3D-motor (eng. *3D engine*) for en 3D-omgivelse, hvor den kan defineres som et sett av strukturer, algoritmer og funksjoner som benyttes til å visualisere, etter mange kalkuleringer og transformeringer, 3D-objekter på en 2D-skjerm. 3D-motoren definerer med andre ord hvor godt man greier å gjengi den virkelige verden grafisk - graden av realisme.

Slik aura- og fokusbegrepene benyttes i denne oppgaven er det ikke overlapp mellom aura og fokus som danner grunnlaget for kommunikasjon; fokus og aura satt i sammenheng med soneinndelinger gjør at det er en felles interessesone som bestemmer dette - fokus og aura trenger m. a. o. ikke dekke hverandre, så lenge de dekker en felles interesssesone.

Før man kan finne radius til fokus og aura basert på forslaget i figur 2.8 på forrige side, må man ha $initielle$ verdier som man kan ta utgangspunkt i. Med dette menes det scenariet hvor det ikke er noen ytre eller indre faktorer som påvirker fokus og aura - hvor synlige klientene er overfor hverandre under "normale" forhold[11]. Hvis vi har to klienter som begge representeres likt i omgivelsene (f. eks. at begge er mennesker eller alver) kan vi så si at de begge ser hverandre innenfor en avstand på f. eks. 500 meter. Ved da å sette radius til fokus og aura slik at resultatet ved å addere disse to svarer til denne distansen, har man så funnet verdier som tilfredsstiller kravet om overlapp mellom fokus og aura for at en interesserelasjon skal være tilstede (en overlapp mellom en klients fokus og en annen klients aura betyr at det må eksistere en felles interessesone mellom disse). Den initielle fokus og aura blir da gitt ved

$$a + f = MD$$

hvor a er radius til aura, f er radius til fokus og MD er den maksimale distanse de to klientene er synlig overfor hverandre.

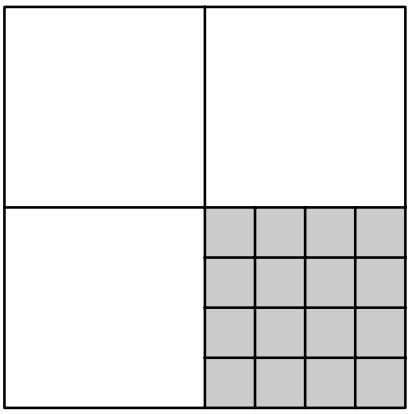
Siden det er når to klienter får en felles interessesone som gjør kommunikasjon mulig dem imellom, kan estimatet for maksimaldistansen gjerne settes noe lavere en hva denne formelen tilsier. To klienter vil i de aller fleste tilfeller opprette forbindelse til hverandre lenge før selve fokus og aura overlapper hverandre. Se f. eks. på figur 2.7 på side 24, hvor klient A og B har en toveis kommunikasjon. Dette fordi det er en felles interesssesone, mens deres respektive fokus og aura ikke dekker hverandre.

Radius som funksjon av sonetreets topologi

Hvilke soner en gitt klient har i fokus og som dekkes av dens aura kan være med på å bestemme størrelsen til fokus og aura. Hvis vi betrakter figur 2.9 på neste side ser vi at det må forekomme en konsentrasjon av klienter i området med de grå sonene.

En klient som befinner seg i en omgivelse, som vi her ser representert ved soneinndelingen, vil alltid ha en eller flere av sonene i sin fokusliste og sin auraliste. Disse listene sier så noe om hvorvidt klienten er i nærheten av, utenfor eller i en klynge. De hvite sonene på figuren er på dybde 1 i sonetreet, mens de grå er på dybde 3. Hvis klientens sett av fokus-soner kun består av

[11] Normale forhold defineres her som en omgivelse med normalt dagslys - lysforhold som er optimaliserende mht. hvor godt vi kan se. Tåke, regn, snøvær og mørke er typiske effekter som forringer våre synsinntrykk.



Figur 2.9: Soneinndeling med konsentrasjon av klienter.

soner på dybde 1 (f. eks. de to øverste hvite sonene) betyr dette at klienten ikke er i nærheten av klyngen. Har klienten derimot soner av både dybde 1 og 3 som fokus-soner betyr dette at klienten er i nærheten av klyngen eller i klyngen; den må da befinne seg i en av de hvite sonene og i tillegg ha en eller flere av de grå sonene i fokus, eller den må befinne seg i de grå sonene og i tillegg ha en eller flere av de hvite sonene i fokus. Har klienten kun soner av dybde 3 i fokus må den befinne seg i området hvor klyngen opptrer.

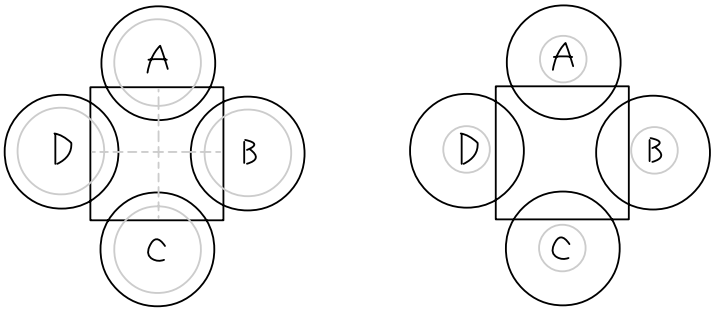
Som tidligere nevnt vil en klients aura og fokus minske som resultat av at klienten befinner seg i en klynge sammen med andre klienter (fordi vi omringes av sanseinntrykk med samme modalitet, noe som gjør det problematisk å skulle fokusere på flere av disse). Siden settet av fokus- og aurasoner for en gitt klient vil kunne si noe om konsentrasjonen av andre klienter i området hvor den befinner seg, bør dette kunne være med på å bestemme størrelsen til fokus og aura. Jo dypere i sonetreet en klients fokus- og aurasoner befinner seg (eventuelt snittverdi for dybden), desto mindre vil radius for fokus og aura kunne bli.

2.4.2 Kriterier for splitting av soner

Etter å ha introdusert prinsippet med dynamisk fokus og aura må vi igjen se på kriteriene for å splitte opp soner. I forslaget til H. A. Abrams [13] (hvor dynamisk soneinndeling presenteres) baserer man balanseringen av sonene ved å betrakte hvor mange klienter/objekter som befinner seg i hver sone. Dette er fordi to klienter har et felles interesseområde når interessesirkelen til den ene klienten har overlapp med sonen hvor den andre klienten befinner seg. Ved å splitte opp en sone som har overskredet en terskel for hvor mange som kan befinne seg i sonen, vil man så for en gitt klient kunne sørge for at

flere klienter/objekter havner utenfor interessemrådet (som vist i figur 2.5 på side 21).

Ved å kombinere denne soneinndelingen med dynamisk fokus og aura blir derimot interesseområdet mellom to klienter bestemt ved en eller flere felles soner, og hvor det for disse er samsvar mellom den ene klientens aura og den andre klientens fokus - ingen av klientene trenger m. a. o. å befinne seg i interessesonen. Dette innebærer at splittingen av en gitt sone må basere seg på hvor mange klienter som dekker den med sin fokus eller aura, eller en kombinasjon av disse to. Figur 2.10 illustrerer dette.



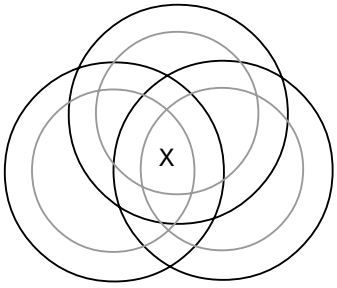
Figur 2.10: Splitting av soner basert på hvor mange som har sonen i sin fokus og aura.

Figuren viser hvordan en sone splittes opp selv om ingen klienter befinner seg i selve sonen. Det som trigger en splitt er at klientene A,B,C og D har sonen i sin fokus (sort sirkel) og aura (grå sirkel). Disse klientene vil opprette forbindelser til hver av de andre fordi de alle har sonen som et felles interesseområde. Forutsetningen for splitting her er at fire fokus og fire aura er assosiert med sonen. I tilfellet til høyre vil det dermed ikke forekomme en splitting fordi det kun er klientenes fokus som dekker sonen. Til venstre ser vi derimot at kriteriene for splitting er oppfylt, og vi ser at dersom sonen splittes vil ikke klientene A og C lenger ha et felles interesseområde - samme gjelder for klientene B og D.

Man kan tenke seg at en sone splittes opp hvis for mange har den i fokus eller for mange har den i aura, eller som figuren viser; for mange har den i aura *og* for mange har den i fokus. Det siste alternativet (som også er i samsvar med figuren) vil være det beste. Hvis f. eks. en sone har overlapp av hundre klienters fokus så er det ikke noen grunn til å splitte sonen hvis ingen har overlapp med sin aura - dette fordi kommunikasjon mellom to klienter baserer seg på at en klient har sonen i fokus mens den andre dekker den med sin aura. Tilfellet til høyre på figur 2.10 viser at det ikke er noen grunn til å splitte opp sonen fordi sonen ikke er noe felles interesseområde klientene seg i mellom. Det samme gjelder hvis en sone kun assosieres med klienters aura; så lenge ingen har sonen i fokus trenger man ikke splitte den opp.

I kapittel 5 vil vi se på simuleringsresultater av implementasjonen som er laget for denne oppgaven for å kunne vurdere om det er hensiktsmessig å splitte soner basert på en terskel for *både* hvor mange fokus og hvor mange aura som har overlapp med en sone, fremfor å kun betrakte fokus eller kun betrakte aura.

Et viktig aspekt med denne tilnærmingen for å splitte opp soner er at det må bestemmes en maksimal dybde til sonetreet. Hvis dette ikke gjøres, vil sonetreet kunne splittes opp i en uendelig rekursiv brønn. Dette vil skje når det for et bestemt punkt er overlapp av flere klienters fokus og aura og summen av disse overskrider terskelen for splitting. Dette er vist i figur 2.11.



Figur 2.11: Hvis terskelen her for splitt er tre fokus og tre aura vil vi få en uendelig rekursjon, siden punktet X overlappes av tre klienters fokus og aura. Samme hvor mange ganger vi splitter opp, så vil vi ha en sone som assosieres med de tre klientenes fokus og aura.

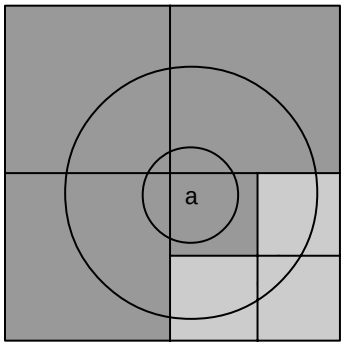
2.4.3 Meldingsutveksling via multicast og unicast

Når man så har bestemt størrelsene på fokus og aura vet man også hvilke soner som helt eller delvis dekkes av disse - m.a.o. hva som er de *aktuelle* sonene for mulig interaksjon med andre klienter. Fokus og aura vil typisk ikke ha samme størrelse, så settet av aktuelle soner vil ikke nødvendigvis være det samme. Hvordan skal man så opprette forbindelser til de andre klientene som har en eller flere aktuelle soner (felles interesseområde) til felles med deg?

Multicast

For multicast vil man kunne gjøre dette todelt; gitt at en sone assosieres med en multicast adresse, kan man så melde seg som sender (*source*) til multicastadressene til auraens aktuelle soner, og man kan melde seg som mottaker (*receiver*) for de aktuelle sonene til fokus. Figur 2.12 på neste side viser fokus (liten sirkel) og aura (stor sirkel) til entitet a. A vil melde seg som mottaker til de mørke sonene og som sender til alle sonene. Vi tar her utgangspunkt

i at man melder seg kun som sender *eller* kun som mottaker for en multicastgruppe, altså ikke som begge deler. Dette valget begrunnes i seksjon 2.5.1.



Figur 2.12: Den minste sirkelen viser fokus til a, mens den største viser aura.

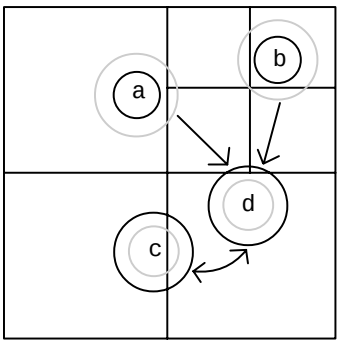
Unicast

Ved bruk av unicast vil to klienter opprette forbindelse så sant en gitt sone er i fokus for den ene klienten og dekkes av den andre klientens aura. Klienten som dekker sonen med sin aura vil da sende oppdateringspakker til klienten som dekker den med sin fokus siden sistnevnte klient *ser* førstnevnte, mens oppdateringsmeldinger ikke vil sendes andre veien. Kommunikasjon vil gå begge veier hvis klientene har en felles sone som begge dekker med både fokus og aura. For en gitt sone hvis to klienter kun dekker den med sin aura, vil det ikke opprettes forbindelse i det hele tatt.

Figur 2.13 på neste side viser et eksempel på hvordan klienter kan kommunisere via unicast. Grå sirkel viser her aura, mens sort sirkel viser fokus for klientene a,b,c og d. A og b vil ikke ha en forbindelse seg imellom, da ingen av sonene overlappes av den enes fokus og den andres aura (de har en felles aurasone - noe som ikke er grunnlag for å opprette forbindelse). D derimot har både a og b's aura i fokus, mens a og b ikke har d's aura i fokus. Oppdateringspakker vil så sendes kun fra a og b til d. C og d vil sende begge veier siden de begge har den andres aura i fokus.

2.5 Valg av hybrid arkitektur

Et hovedmål med en nettverkssimulering med et stort antall brukere er at den må være skalerbar - vi ønsker at belastningen på serveren eller en klynge av servere ikke skal øke proporsjonalt med antall deltagere.



Figur 2.13: Unicast forbindelser for kommunikasjon klientene seg imellom.

Det er serveren som holder den overordnede tilstanden til simuleringen i dagens storskala nettverksspill, men enhver hendelse trenger ikke gå via serveren. I et spill som f. eks. $Ultima\ Online$ går all kommunikasjon via en sentralisert server. Sikkerhetsmessig er dette en god løsning, da all interaksjon og oppdateringsmeldinger må valideres av serveren før de kan påvirke den virtuelle omgivelsen. Men trenger all kommunikasjon gå via serveren? Er det av global interesse hva to klienter foretar seg imellom, så lenge det ikke påvirker omgivelsene? I [14] er det foreslått at man i en virtuell omgivelse skiller mellom:

- *Omgivelser*: Objekter som ikke styres av deltakere, men som kan manipuleres av deltakere. Herunder regnes også landskap som ikke er statisk - f. eks. at det dannes et krater i bakken dersom et objekt eksploderer.
- *Interaksjon*: informasjonsutveksling mellom to eller flere deltakere hvor omgivelsene ikke påvirkes. Eksempel kan være en samtale eller en slåsskamp.

Omgivelsene er av global interesse. Hvis en klient f. eks. plukker opp et våpen som finnes på bakken vil dette påvirke omgivelsene. Serveren må bli fortalt at objektet ikke lenger ligger der hvor du plukket det opp slik at andre ikke også skal kunne finne det på samme sted.

Ved å benytte en hybrid peer-to-peer-arkitektur kan vi så ha en sentralisert server som har ansvaret for å opprettholde korrekt tilstand og distribuere oppdateringer relatert til omgivelsene. Det er da viktig at serveren får tilsendt informasjonen av denne typen, men det er ikke *kritisk* at det skjer fort. Det er viktigere for andre entiteter som er medlem av den aktuelle sonen å få slik informasjon raskt. Våpenet klienten plukket opp kunne like gjerne blitt plukket opp av klienter like i nærheten. Derfor er det av høyere prioritet at alle klienter som har sonen hvor våpenet befinner seg som *aktuell* sone

(sonen i fokus) får tilsendt denne informasjonen raskt enn at serveren får denne informasjonen.

Serveren trenger denne informasjonen for å kunne fortelle nye abonnenter (klienter som får den i fokus) av sonen hva som er forandret mht. omgivelsene i sonen. Siden fokus til en entitet vil komme over en sone (og dermed gjøre den aktuell for entiteten) før selve entiteten beveger seg inn i selve sonen, er det lite sannsynlig at våpenet du plukket opp vil kunne være "observerbart" før en stund etter at sonen ble aktuell for entiteten. Dermed gjør det ikke noe for serveren at objektet fortsetter å "eksistere" en kort stund på et sted hvor den faktisk ikke gjør det fordi alle som har sonen i sitt interesseområde har fått beskjed, og det er disse som har potensiell interesse av dette objektet.

Simuleringens arkitektur blir dermed delvis distribuert ved at tilstandsoppdatering delegeres til klientene, hvor klientmeldinger av typen interaksjon kun går direkte mellom klientene. Arkitekturen blir dermed en hybrid peer-to-peer-arkitektur siden en sentralisert server mottar oppdateringer som påvirker omgivelsene. Alternativt kan også disse oppdateringene kun gå mellom klientene, men dette krever at man finner en god algoritme for å distribuere dette ut til alle klientene i hele simuleringen[12]. Det viktigste her er at oppdateringer av typen interaksjon ikke går via en server, da vi kan anta at forekomsten av denne typen oppdateringer vil være større en oppdateringer relatert omgivelsene.

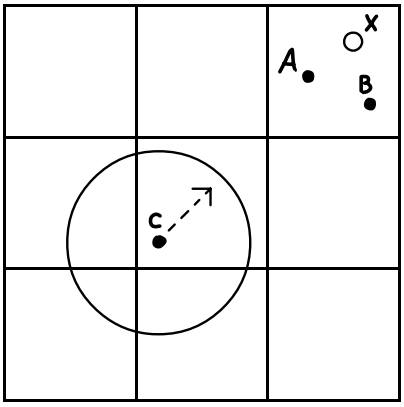
Figur 2.14 på neste side viser klientene A og B som har sonen med objekt X som aktuell sone - dvs. i fokus. Hvis A f. eks. flytter på X må A gi B beskjed om dette straks. Serveren på sin side trenger denne informasjonen for å fortelle C, som er på vei mot sonen hvor X befinner seg, at X er flyttet på. Men som vi ser på figuren vil fokus til C (sirkelen) gjøre sonen aktuell en god stund før C befinner seg i selve sonen (C's hastighet og størrelsen til fokus tatt i betraktning). Dette viser at det er av høyere prioritet å fortelle B om flyttingen av X enn at serveren får denne informasjonen.

2.5.1 Unicast eller Multicast?

Bør en nettverkssimulering benytte multicast eller unicast? Her vil vi argumentere for bruk av unicast fremfor multicast så langt det lar seg gjøre, men at det i visse scenarioer vil være mest hensiktsmessig å benytte multicast. Internett støtter ikke fullt ut multicast pr. i dag og er dermed ikke et alternativ for kommersielle applikasjoner som nettverksspill, men det er viktig å betrakte mulighetene for bruk av multicast da dette mest sannsynlig vil være et reelt alternativ i fremtiden.

Hvis arbeidsoppgavene distribueres som nevnt ovenfor vil en simulering

[12] F.eks. så kunne den klienten som har hatt sonen i fokus lengst hatt ansvar for å oppdatere nye klienter som får sonen i fokus om hva som er forandret med hensyn til omgivelsene.



Figur 2.14: Et eksempel på at serveren ikke trenger å motta informasjon som påvirker omgivelsene like raskt som klientene.

skalere bra selv om man benytter unicast. Ved å la mesteparten av informasjonsflyten foregå mellom entitetene med felles aktuelle soner vil det være fornuftig å benytte seg av unicast siden unicast benytter den korteste ruten mellom to endepunkt. Dette er i samsvar med at informasjonsutveksling som er av kritisk art (f. eks. to personer i nærkamp eller biler i et billøp) bør oppleve minst mulig forsinkelse i nettverket[13]. Forsinkelse er hva som definerer dynamikken i en virtuell omgivelse. Forsinkelse vil man uansett ha når signaler skal sendes gjennom fysiske medier. Hva som øker forsinkelse er hendelser som ruting, køer og prosessering av pakker.

Når man snakker om multicast og oppbygging av multicast-trær må man skille mellom *kildespesifikke* og *gruppedelte* trær. En multicast-gruppe som er kildespesifikk har kun én sender av data - resten er mottakere. Eksempel her kan være en server som man kan streame video fra. For å minimalisere forsinkelsen til et slikt tre er det ganske enkelt bare å finne korteste vei fra senderen til alle mottakere og fjerne eventuelle løkker.

For en virtuell omgivelse derimot som ønsker å benytte multicast må man lage multicast-grupper som benytter seg av gruppedelte trær - alle medlemmer skal kunne sende data til resten av gruppen. Her er ikke oppbyggingen av et optimalt tre, hvis minimalisering av forsinkelse er målet, like triviell. Den gjennomsnittlige forsinkelsen (FG) for et gruppedelt tre er i [16] gitt ved

$$FG = \frac{1}{|M|} \sum_{v \in \mathbf{M}} FK_v,$$

hvor FK_v er den gjennomsnittlige kildespesifikke forsinkelsen med v som kilde, og M er medlemmene i multicast-gruppen. Det å skulle optimalisere FG

[13] Når forsinkelser overskrider 100ms begynner de å bli merkbare for oss mennesker[15].

for en multicastgruppe er NP-komplett, og man må finne løsninger som kun er tilnærminger, men som lar seg løse på polynomisk tid. Slike tilnærmingsalgoritmer kan føre til at den gjennomsnittlige forsinkelsen for gruppedelte trær blir det dobbelte av den optimale løsningen.

Denne ikke-optimaliseringen av gruppedelte multicast-trær på bekostning av den gjennomsnittlige forsinkelsen er et argument for heller å benytte unicast så sant det lar seg gjøre, hvis målet er å minimalisere forsinkelsen mellom to entiteter som interagerer med hverandre, og hvor denne er av stor relevans.

Valg av unicast som kommunikasjonsform mellom entiteter som har felles aktuelle soner vil da kunne være med på å bestemme størrelsen på sonene - m. a. o. når man skal foreta splitting av en sone. En gitt klient vil typisk opprette forbindelser til alle klienter den har et felles interesseområde med, og fornuftige grenser for når man skal splitte soner vil kunne holde disse peer-to-peer-forbindelsene på et minimalt nivå.

Bruk av multicast er ideelt når man ønsker å sende samme pakke til et stort antall endebrukere (en-til-mange). Her er multicast overlegen unicast med tanke på nettverksbelastning - senderen sender kun en pakke, som kopieres ved forgreingene til multicast treet (rutere). Ved bruk av unicast vil man måtte ha $N*(N-1)$ forbindelser til endebrukere man ønsker å sende pakker til. Dette resulterer i at en bruker må sende N-1 kopier av en pakke i en en-til-mange-situasjon. Dette skalerer ikke særlig bra hvis N er stor. Her vil man heller kunne opprette en multicastgruppe som er kildespesifikk. Som nevnt ovenfor kan man enkelt bygge et slikt tre hvor forsinkelsen mellom sender og hver mottaker er optimalisert. Eksempel på en slik situasjon kan være taleren som snakker til en stor forsamling, hvor da taleren typisk vil være kilden og alle tilhørere kun er mottakere i multicastgruppen.

Topologiforandringene for sonetreet skjer hos serveren, men distribueres ut til alle klientene - altså en en-til-mange-kommunikasjon. Arkitekturen som presenteres i kapittel 4 benytter multicast for dette. En alternativ løsning som benytter unicast kan være at serveren distribuerer topologiforandringene til et sett av *dedikerte* klienter hvis hver av disse assosieres med en bestemt sone og hvor kun en klient assosieres med en sone. For hvert soneobjekt har serveren en liste over objekter som har sonen i fokus. Serveren kan da velge å sende oppdateringene til en av disse, f.eks. klienten som har vært i listen lengst (altså den som har hatt sonen lengst i fokus). Serveren vil da sende ut *n*, eller færre, oppdateringspakker hvor n tilsvarer antall soner (det vil sendes til færre enn n hvis en eller flere soner ikke assosieres med noen klienters fokus). Disse dedikerte klientene vil så distribuere oppdateringene videre til de klientene de har felles interesseområder med.

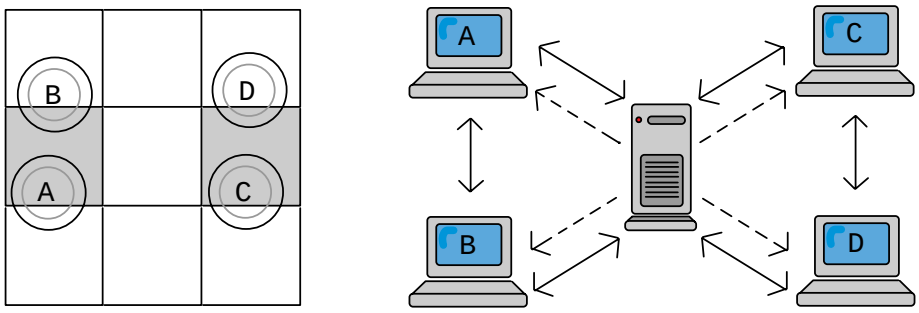
Som nevnt vil oppdateringer av typen *interaksjon* ikke sendes til serveren, mens oppdateringer av typen *omgivelser* vil sendes til serveren (hybrid løsning) eller distribueres av klientene (ren peer-to-peer løsning). Arkitekturen for selve relevansfiltreringen vil uansett være *hybrid peer-to-peer* fordi vi

har en sentralisert server som administrerer sonetreet og sender topologiforandringene for dette til klientene. Serveren har også ansvar for å distribuere informasjon til hver enkelt klient om at nye klienter har kommet i fokus (klienten må derfor sende oppdateringer av sine fokus- og auralister til serveren).

2.6 Oppsummering og definering av den foreslåtte arkitektur

Vi har i dette kapitlet presentert hva vi har valgt å betegne som en avstandsfiltrering (som her inngår som den initielle delen av en relevansfiltrering) hvis mål er å definere det subsettet av klienter som er av potensiell interesse for en gitt klient. Vi har sett på prinsippet med en dynamisk soneinndeling (som er innspirert av [13]) og prinsippene bak dynamisk fokus og dynamisk aura (som bygger på lignende konsepter presentert i [19] og [20]). Basert på en kombinasjon av disse to prinsipper har vi så kommet frem til en ny variant for avstandsfiltrering.

Figur 2.15 gir en oppsummering av hvordan arkitekturen til avstandsfiltreringen er tenkt basert på de prinsipper som er presentert i dette kapitlet.



Figur 2.15: Hvordan klienter interagerer med hverandre og med serveren.

Til venstre ser vi posisjonen til fire klienter i en virtuell omgivelse (med en soneinndeling på 3x3 soner) hvor fokus er sort sirkel og aura er grå sirkel. Klientene A og B kommuniserer med hverandre og klient C med klient D. Dette er bestemt av deres fokus og aura: klientene har felles *interesseområder* i de grå sonene. Kommunikasjonen er toveis for begge parene av klienter fordi de respektive interessesonene dekkes av både fokus og aura for samtlige av klientene.

Høyre del av figuren viser hvordan klientene kommuniserer med hverandre samt med serveren (i samsvar med venstre del av figuren). Heltrukken linje viser unicast-kommunikasjon, mens stiplet linje viser multicast. Klientene sender kun oppdateringspakker via unicast, mens serveren benytter både mul-

ticast og unicast (topologiforandringer i sonetreet distribueres via multicast, eller som nevnt i seksjon 2.5.1 via unicast hvor serveren sender oppdateringene til et sett av dedikerte klienter).

For selve relevansfiltreringen er serverens oppgave å korrekt oppdatere topologien til sonetreet. Derfor må serveren til en hver tid, for hver klient i hele simuleringen, vite hvilke soner som dekkes av aura og fokus. Klientene må regelmessig sjekke sine lister med fokus- og aurasoner og oppdatere serveren hvis det gjøres forandringer i disse. Slik vil serveren kunne underette en klient om at det eksisterer et felles interesseområde mellom denne og annen klient hvorpå klientene selv oppretter forbindelser seg imellom. To klienter som har en forbindelse vil så regelmessig utveksle informasjon om sine fokus- og aurasoner slik at forbindelsen kan brytes når det ikke lenger finnes noe felles interesseområde.

Klientene sender også oppdaterings pakker av to typer: *interaksjon* og *påvirkning av omgivelser* (se seksjon 2.5). Meldinger av typen interaksjon sendes kun til de klienter det måtte berøre (f. eks. en samtale) som vil være de klienter man har et felles interesseområde med. Oppdateringer hvor en klient har påvirket omgivelsene vil først distribueres til klientene med felles interesseområde og dernest til serveren. Dette gjør at arkitekturen blir en blanding av *sentralisert* og *distribuert*. Mye kommunikasjon vil gå direkte mellom klientene, noe som reduserer belastningen på serveren. Arkitekturen kan dermed betraktes som en hybrid peer-to-peer-arkitektur (se definisjonen i seksjon 2.1.1).

Avstandsfiltreringen i dette kapitlet er presentert i relasjon til brukerstyrte entiteter - avatarer som gjerne fremsår som menneskelignende skikkelser eller motoriserte farkoster (biler, tanks, fly, ubåt o.l.). På lik linje kan også objekter benytte de samme prinsipper mht. interesseområde for å definere området objektet kan *manipulere*. Eksempel på dette kan være et våpen hvor da rekkevidden tilsvarer dets radius for “fokus”.

KAPITTEL 3

Interessefiltrering

3.1 Innledning

En klients fokus og aura definerer som beskrevet i forrige kapittel en mulig løsning for problemet med å avgjøre hvilke andre klienter som befinner seg innenfor et sansbart felt - de $potensielle$ kandidatene for kommunikasjon.

Nå har turen kommet til å se på $interessefiltreringen$ hvor avstandsfiltreringen inngår som den initielle del. Vi vil m. a. o. foreta en filtrering av de potensielle klientene (altså de ovenfornevnte kanditatene for kommunikasjon). Avstandsfiltreringen vil for en gitt klient kunne filtrere bort en stor andel av de andre klientene i en virtuell omgivelse. Men hva så med det sub-settet av klienter man har et felles interesseområde med? Selv om en avatar (den grafiske representasjonen for en klient i en virtuell omgivelse) er innenfor et sansbart felt kan det hende vi velger å vie den minimal oppmerksomhet fordi den ikke fanger vår oppmerksomhet - den er ikke av interesse.

Vi har allerede sett hvordan man ved å betrakte to klienters aura og fokus kan bestemme om kommunikasjonen er to-veis eller en-veis. Vi vil i dette kapittelet se på hvordan man ytterligere kan begrense informasjonsflyten. Ved å definere en klients interesse eller oppmerksomhet vil vi kunne rangere de potensille klientene etter grad av interesse. Hvor hyppige og detaljerte oppdateringer man mottar fra en gitt klient vil da kunne relateres til hvor interessant klienten er; jo mindre interesse, desto mindre detaljert informasjon trenger man å utveksle.

3.1.1 Interessefiltrering i den virkelige verden

For å angripe problemstillingen angående hva som er relevant informasjon i en virtuell omgivelse vil det være hensiktsmessig å se på hvordan vi som mennesker forholder oss til strømmen av sanseinntrykk fra den verden vi lever i; m. a. o. se på prinsipper for relevansfiltrering i den reelle verden, og forsøke å overføre disse til den virtuelle.

En virtuell verden er jo en simulering av en virkelig verden, og det er ikke bare i den virtuelle det er behov for relevansfiltrering. Vi bør derfor kunne se på hvordan vi som mennesker filtrerer ut relevant informasjon i et mylder av informativ stimuli som til enhver tid omgir oss. Når du som leser vier din oppmerksomhet til denne teksten, så gjør du det på bekostning av andre meningsgivende impulser du gjerne kunne vært like oppmerksom på, bevisst eller ubevisst. Du tenker kanskje ikke over om du sitter behagelig eller ikke på stolen, eller over samtalene på andre siden av rommet. Denne relevansfiltreringen vi til enhver tid utfører bør kunne studeres og være med på å definere en relevansfiltrering i den virtuelle verden hvis hensikt er å minimalisere bruken av båndbredde som konsumeres av denne. Målet blir å simulere hvordan vi som mennesker forholder oss til og prosesserer et minimum av all informasjon vi mottar fra våre omgivelser uten at det går utover kvaliteten og opplevelsen i en virtuell verden vi ønsker å overføre disse begrensningene på.

Vi er selektive til hva vi er oppmerksomme overfor, og det er ikke tilfeldig hva som fanger vår oppmerksomhet. Dette bestemmes av et samspill mellom sanseorganene og vårt sinn. Hva som begrenser vår evne til å sanse og prosessere sansestimuli baserer seg på våre psykiske og fysiske begrensninger. Ikke bare er vi selektive, men vi er som mennesker veldig forskjellige. Hva som for en person er relevant informasjon kan virke helt uinteressant for en annen, og motsatt. Sanseapparatet varierer kanskje lite fra person til person, men det er hvordan vi gir mening til eksterne input, som i seg selv er meningsløse, hvor vi finner de store variasjonene. Det vi *ser* - energien som kommer inn gjennom øyet - er det samme, men hvordan vi *tolker* dette vil ofte variere stort. Slike variasjoner gjelder ikke bare fra person til person, men også for en bestemt person; våre tolkninger er *situasjonsbestemte*.

Hvordan vår evne til å filtrere og prosessere relevant informasjon kan videreutvikles og forbedres vil også å være relevant i denne sammenheng. Som nevnt ovenfor er vår oppmerksomhet dynamisk ved at den er situasjonsbestemt, men den er også dynamisk i den forstand at vi tilegner oss ny kunnskap og nye erfaringer som kan forbedre (eller forvrenge) vår virkelighetsoppfatning og informasjonsfiltrering. Ikke bare vil vi kunne forbedre vår filtrering basert på hva vi mottar av sanse stimuli - vi vil også kunne sanse ting vi nødvendigvis ikke mottar input om fordi vi erfaringsmessig gjenkjenner situasjoner og settinger.

Oppmerksomhet

Det er ikke alt som stimulerer sanseorganene våre som transformeres til mentale representasjoner. Vi er *selektive* overfor hvilke objekter eller hendelser vi vier vår oppmerksomhet. Dette har selvfølgelig også en sammenheng med at vi har en begrenset *prosesseringskapasitet*. Psykologen Donald Broadbent har gitt følgende definisjon på oppmerksomhet [12]: "Det er et resultat av et informasjonsprosesseringssystem med begrenset kapasitet". Hjernen, såvel som sanseapparatet har ikke mulighet til å forholde seg til mylderet av sanseinntrykk som til enhver tid omringer oss. Her kan man se en direkte forbindelse til de begrensede nettverksressursene man har innenfor en virtuell omgivelse. Kan man legge disse kriteriene til grunn for en relevansfiltrering, bør man kunne oppnå en gunstig filtrering basert på hva som er viet endebrukerens oppmerksomhet. Problemet blir å skulle predikere dette - man må m. a. o. se på hva som *bør* vie vedkommendes oppmerksomhet.

Oppmerksomhet er et viktig aspekt innen kognitiv psykologi ved at det gir oss evnen til å føre en planlagt og tilpasset oppførsel, og en målrettet *kontroll* overfor våre handlinger. Hadde vi ikke hatt denne evnen til å velge, ville vi vært redusert til å forholde oss til de signaler som til enhver tid er sterkest.

3.2 Interessegrader

Som det går ut av navnet, $interessefiltrering$, er det her snakk om å forsøke å definere graden av interesse to klienter seg imellom. Man kan tenke seg en skala med to ytterpunkt:

- *Ingen interesse*. Selv om to klienter er fullt sansbare overfor hverandre kan det tenkes at det allikevel ikke er grunnlag for meldingsutveksling mellom disse. Det kan f. eks. være at begge klientene har all oppmerksomhet rettet mot en annen klient eller et annet objekt. Et annet tilfelle vil også kunne være når man har et felles interesseområde, men ikke er synlig overfor hverandre. To klienter som befinner seg i samme sone vil *uansett* ha denne som felles interesseområde selv om klientene er aldri så usynlig overfor hverandre[1].

- *Maksimal interesse*. Hvis to klienter med felles interesseområde har en interaksjon med hverandre av $kritisk$ art, hvor informasjonsflyten er stor og det kreves minimal forsinkelse. Eksempel her kan være to klienter i nærkamp med hverandre. Her vil man da kunne si at klientene kun har "interesse" for hverandre - at all fokus er rettet mot den andre klienten. Dette vil da kunne føre til at andre klienter innenfor

[1] Sonen en klient befinner seg i vil alltid være fokus-sone og aura-sone for klienten, så sant man tar utgangspunkt i at en klient til en hver tid skal ha en aura og en fokus.

interesseområdet blir å betrakte som klienter av ingen eller minimal interesse, som beskrevet ovenfor.

Med utgangspunkt i dette vil vi her se på interessefiltreringen av en gitt klients sett av potensielle klienter som en inndeling av disse etter graden av interesse for hver enkelt klient. Vi kan betrakte en klients interesse, eller oppmerksomhet, som *endelig*. Dette blir da i relasjon til menneskets faktiske begrensninger mht. oppmerksomhet (se 3.1.1). Tar vi dette i betrakting kan vi så si at en klient vi vier interesse (altså en potensiell klient) konsumerer en bestemt del av denne endelige interessen. En klient som assosieres med en høy interessegrad vil så legge beslag på en større del av en bestemt klients interesse enn en klient som assosieres med en lavere interessegrad. M. a. o. så kan vi ha flere klienter gradert til en lav interessegrad enn til en høy interessegrad før den endelige interessen er "brukt opp". Denne verdien som bestemmer den endelige interessen vil så være dynamisk under simuleringens/spillets gang - som da vil reflektere hvor skjerpet, eller oppmerksom klienten er. Dette vil da være med på å bestemme/sette en grense for hvor mye båndbredde en gitt klient konsumerer hvis vi betrakter de oppdateringspakker som sendes fra andre klienter (de potensielle).

Man må så se på hvilke kriterier som skal ligge til grunn for en slik gradering av hvor interessant hver av de potensielle klientene er, og hvor finkornet den skal være - hvor mange interessegrader man skal dele inn i.

3.2.1 Interessegradene som prioritetskøer

Klienter som betraktes som klienter av høy interesse vil så prioriteres fremfor klienter av lavere interesse. Dette kan manifesteres ved å betrakte interessegradene som prioritetskøer slik at pakker av kritisk karakter eller høy interessegrad vil prosesseres før pakker av lavere interessegrad. Den enkleste måten å implementere dette på er å prosessere pakkene i køen til den enkelte interessegrad i samsvar med $FIFO$-prinsippet (first-in-first-out), og at man alltid tømmer køen til pakker av høyere prioritet før man prosesserer klientene i interessegraden under. Et problem som da kan oppstå er at køer av høy prioritet totalt kan blokkere de av lavere prioritet - så lenge det er minst én pakke i køen for en bestemt interessegrad vil dette da hindre køen under i å sende sine pakker.

Frekvensen av pakker pr. tidsenhet vil typisk være større for klientforbindelsene av høy interesse enn hva den vil være for de med liten interesse. Synlige klienter som klassifiseres med lav interesse kan kanskje nøye seg med å sende såkalte $keep-alive$-pakker for å fortelle at klienten fortsatt eksisterer i simuleringen.

3.2.2 Interessegrader bestemmer detaljrikdom

I tillegg til at pakker av større interessegrad prioriteres fremfor pakker av lavere interessegrad kan også de forskjellige gradene bestemme *hva* slags oppdateringspakker som skal sendes. Vi ønsker mer detaljert informasjon om objekter som er av stor interesse enn hva tilfellet er for de med lav interesse.

Pakker som forteller om objekters lokasjon kan betraktes som det mest grunnleggende og vil inngå i alle interessegrader - på laveste interessenivå vil man kanskje kunne si at det kun er disse som er av relevans. Hvor unøyaktig man vil tillate å representere lokasjonen og bevegelsene til et objekt av lav interesse kan igjen avhenge av hvor unøyaktige $dead\ reckoning$ algoritmer man tillater (se avsnitt 1.4.3 på side 9).

Hva slags pakker som skal transmitteres innenfor de forskjellige interessegradene vil være meget avhengig av simuleringen/spillet - dette vil kunne *bestemme* hvor mange interessegrader man ønsker å dele inn i og hvordan hver enkelt av disse skal defineres. Utover pakker som inneholder en klients eller objekts lokasjon, vil man kunne ha pakker som beskriver deres oppførsel - visuelt (bevegelser) og lyd (f. eks. tale). Bevegelser vil kunne klassifiseres etter hvor detaljerte de er, eller i hvor stor grad de fanger den enkeltes oppmerksomhet. F. eks. så er det mest sannsynlig at det kun er de som vier deg direkte oppmerksomhet som ønsker å motta oppdateringer som går på forandring i ansiktsuttrykk og mindre kroppsbevegelser. Klienter som vier en gitt klient oppmerksomhet av mer sekundær art vil overse slike bevegelser, men kanskje heller legge merke til større bevegelser.

I et nettverksspill vil det ofte være et stort utvalg av bevegelser en avatar kan utføre. Det kan dreie seg om et stort utvalg av kampbevegelser og bevegleser som representerer mer "sosial" atferd (f.eks. dans eller gestikulering i en samtale). I $Anarchy\ Online$ er det eksempelvis 64 slike sosiale bevegelser. Et slikt sett av mulige bevegelser en klient kan foreta seg, kan så fordeles utover de forskjellige interessegradene, slik at man overfor en klient av en gitt interesse kun sender oppdateringer for bevegelser som tilsvarer interessegraden klienten har blitt gradert til, samt de bevegelsene som finnes i alle de lavere interessegradene.

I tillegg til at en bestemt oppdateringspakke relateres til en interessegrad ut ifra hvor interessant klienten er, bør også *avstanden* til til hver av de potensielle klientene avgjøre hvilke pakker som skal sendes til hver og enkelt av dem. En klient av stor interesse kan befinne seg såpass langt unna at man ikke vil kunne ha noen oppfatning av de mindre og mer detaljerte bevegelsene den foretar seg[2]. Man kan da definere en maksimal avstand til en klient

[2]Siden det i denne oppgaven er de felles interessesonene som bestemmer om to klienter er av potensiell interesse for hverandre er det da her viktig å ta i betraktning avstanden til en klient som et kriterie for interessefiltreringen. Dette fordi to klienter ofte vil få en interserelasjon selv om det ikke er noe overlapp mellom fokus og aura (som f.eks. klient c og d i figur 2.13 på side 32). Klienter man ikke har noe fokus-/auraoverlapp med, men som likevel

for hver enkelt bevegelse. For tale vil dette kanskje fremstå som enda mer opplagt; her vil f.eks. normal tale nå lenger enn hvisking.

Dermed vil vi overfor klienter som er lokalisert nære nok til at vi kan oppfatte detaljerte bevegelser velge å filtrere bort disse hvis interessen er lav, samtidig som slike bevegelser (og lyd) til en klient av stor interesse også vil kunne filtreres bort pga. en stor avstand til klienten.

3.3 Hvem er av interesse?

Før man kan assosiere en klient med en interessegrad, må det defineres kriterier for hvorfor, eller hvordan, klienten er gjenstand for interesse av den gitte grad. Her vil vi se på hvordan dette kan løses, manuelt og automatisk.

3.3.1 Manuell styring av interesse

En løsning er at brukeren selv under simuleringen/spillets gang bestemmer hva som er av interesse. Typisk her vil være at man ved hjelp av musepekeren kan klikke på den avataren eller objektet man ønsker å vie oppmerksomhet (dette kan sammenlignes med menneskets *selektive* oppmerksomhet). Museklikk kombinert med bestemte taster på tastaturet vil så igjen kunne bestemme *hvor* interessant avataren/objektet er.

Denne interaksjonen kan så påvirke og avgjøre hvilke klienter som skal assosieres med de forskjellige interessegradene. Forskjellige algoritmer kan så benyttes for å oppnå dette. Et eksempel hvor man kombinerer museklikk med en tast kan være som gitt i tabell 3.1 på neste side. For å unngå at for mange klienter assosieres med en bestemt interessegrad settes det en maksgrense for dette - jo større interesse, jo færre klienter pr. interessegrad (som er i samsvar med det ovenfornevnte konseptet om en endelig oppmerksomhet). Slik vil klienter av høy prioritet prosesseres raskere enn klienter av lav prioritet, da køene blir kortere for førstnevnte.

3.3.2 Automatisk styring av interesse

Ut ifra predefinerte regler kan vi forsøke å automatisere interesseklassifiseringen av de potensielle klientene. En avatar vil ha et sett av variable som kan sies å definere en personlighet for denne. Disse personlighetsvariablene kan så bestemme hvor interessante to klienter er overfor hverandre. .

Variable som bestemmer en personlighet kan f .eks. deles inn i følgende grupper:

- *Utseende*: Hvis en avatar, slik det er definert i denne oppgaven, har en stor aura er dette fordi den skiller seg ut i omgivelsene - trekker

er potensielle pga. felles interessesone, kan man da velge å automatisk assosiere med den laveste interessegraden (som nevnt ovenfor kan dette da være ytterpunktet *ingen interesse*).

venstremuseknapp	Flytt klienten til høyeste prioritet. Hvis antall klienter som assosieres med interessegraden = max antall tillatt, så flyttes klienten til interessegraden under - sjekker så om den er full osv.
shift + venstremuseknapp	Flytt klienten til høyeste prioritet. Hvis antall klienter som assosieres med interessegraden = max antall tillatt, så flyttes den klienten som har vært assosiert lengst med interessegraden ned til interessegraden under.
ctrl + venstremuseknapp	Flytt klienten ned en interessegrad. Gjentar man operasjonen vil så klienten til slutt få minimal interesse.

Tabell 3.1: Eksempel på hvordan forskjellige brukerkommandoer kan bestemme klienters plasseringer mht. interessegradene

til seg oppmerksomhet. Dette kan gå på størrelse og farger. F.eks. så vil en avatar kledd i rødt skille seg mer ut i grønne omgivelser enn en avatar kledd i blått, fordi rødt og grønt er komplimentærfarger overfor hverandre[3]. Novalitetsmetaforen er en teori som går ut på at vi fokuserer på hva som er nytt og uvant, eller utenom det vanlige. Befinner vi oss i en omgivelse vi kjenner vil vår oppmerksomhet automatisk rettes mot inkonsistens overfor mentale bilder vi har av denne.

- *Mål*: Vi mennesker har i den virkelige verden en målbevisst oppførsel som vil være med å bestemme hva vi vier vår oppmerksomhet og interesse. I rollespill vil en klient til enhver tid kunne ha opptil flere oppgaver (eng. *quests*) som skal løses. Dette blir da typiske *mål* som vil påvirke klientens interesse. En slik oppgave vil gjøre visse klienter eller objekter mer interessante enn andre.

- *Holdning*: En klients holdninger overfor bestemte grupper, allianser o. l. vil også bestemme hvor interessante to klienter er vis-a-vis hverandre. To klienter som er medlem av hver sin gruppering, og hvor disse grupperingene er i strid med hverandre, vil så gjøre klientene mer interessante for hverandre.

Det kan benyttes en blanding av denne automatiske styringen av interesse og den manuelle. Det vil da være naturlig å i første instans gruppere de

[3]To farger som er komplimentære i forhold til hverandre skaper en maksimal kontrast fordi de ikke deler noen felles farge. Det er tre primærfarger: rødt, blått og gult. Et par av komplimentærfarger består av en primærfarge og en farge som er blandingen av de to andre primærfargene. Slik blir grønt (blanding av gult og blått) komplimentært til rødt.

potensielle klientene automatisk, for så å la brukeren manuelt påvirke denne om ønskelig. Den automatiske interessefiltreringen bør gjøres mest mulig effektiv, slik at klienten har minst mulig ansvar for dette.

KAPITTEL 4

Arkitekturen til en relevansfiltrerings-modul

4.1 Innledning

Dette kapittelet vil beskrive et forslag til en arkitektur for relevansfiltrering, og det er da *avstandsfiltreringen* som behandles (som ble bahandlet i kapittel 2). Arkitekturen er implementert i programmeringsspråket $Java$. Vi vil se på hvordan server- og klientsiden kan bygges opp - hvordan hver klient kommuniserer med serveren, og hvordan klientene kommuniserer med hverandre.

Målet for denne avstandsfiltreringer er som nevnt ovenfor å skulle plukke ut det sett av klienter som faller innenfor *interesseområdet* til en bestemt klient (de klienter en gitt klient har et felles interesseområde med). Klienten vil som resultat opprette forbindelser til dette sub-settet av alle klientene som inngår i en virtuell omgivelse.

Arkitekturen tar utgangspunkt i en dynamisk soneinndeling og prinsippene om fokus og aura som beskrevet i kapittel 2. Implementasjonen definerer en generisk *AoIM-modul*[1] som ikke sier noe om det spill/simuleringsspesifikke - den skal kunne benyttes sammen med en hvilken som helst virtuell omgivelse. Arkitekturen som beskrives nedenfor er laget for å kunne benyttes sammen med en 2-dimensjonal virtuell omgivelse, men kunne modifiseres til å fungere sammen med en 3-dimensjonal. Dermed vil fokus og aura her defineres ved sirkler og sonetreet vil få en struktur hvor en foreldre node vil ha fire sub-soner (se figur 2.4 på side 20). Dette valget ble gjort med tanke på testingen av modulen som er tema for kapittel 5 - en 2D-verden vil for oppgavens formål være langt mindre kompleks å generere. Prinsippene for avstandsfiltrering blir de samme.

[1] For resten av oppgaven benyttes denne termen for relevansfiltrerings-modulen (AoIM-modul = Area of interest management modul).

Modulen som her beskrives kan sies å skulle tjene to formål:

- I denne oppgaven som et *eksperiment* hvis hensikt er å benytte den i simuleringer som kan manifistere dens nytteverdi.
- Som *faktisk* å kunne benyttes sammen med en fullverdig applikasjon på linje med dagens distribuerte flerbrukerspill.

AoIM-modulen, dens arkitektur og algoritmer er implementert fra bunnen av og uten påvirkning fra hvordan andre verktøy for relevansfiltrering er designet eller implementert. Kildekoden finnes på:
http://sigve.digidix.com/hovedfag/kildekode/

4.2 AoIM-serverens arkitektur

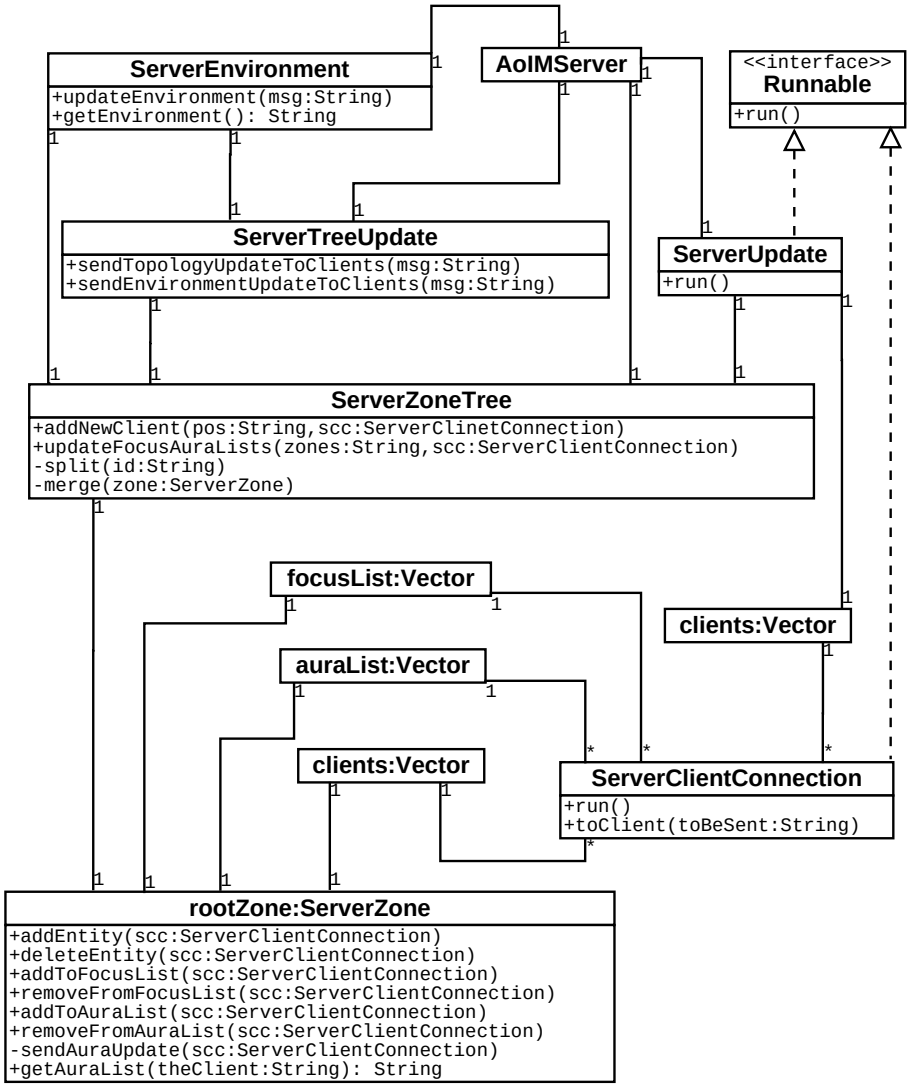
Oppgaven til AoIM serveren vil her defineres som følger:

- Akseptere tilkobling av nye klienter[2].
- Ha ansvaret for at sonetreets topologi til enhver tid er korrekt - dvs. avgjøre når en sone skal splittes eller når fire subsoner skal termineres til foreldrenoden.
- Distribuere topologiforandringene i sonetreet til alle klientene. Dette skjer via multicast.
- For hver sone holder AoIM-serveren informasjon om hvilke klienter som har sonen i fokus og hvilke klienter som har sonen dekket av sin aura. Dette er informasjon som distribueres via unicast til en gitt klient idet denne har fått en ny sone i fokus, eller hvis en sone får en ny aura-klient så vil denne informasjonen distribueres til klientene som har den i fokus.

Figur 4.1 på neste side viser hvordan AoIM-serveren er bygget opp, nedenfor vil vi se på de forskjellige klassene og deres metoder.

class AoIMServer Det eneste denne klassen gjør er å lage et objekt av klassen `ServerTreeUpdate`, et objekt av klassen `ServerZoneTree` og et objekt av klassen `ServerUpdate` samt opprette en tråd for sistnevnte objekt og starte denne.

[2]Det er vanlig å benytte egne oppkoblingsservere (*connection-servers*) som kun har som oppgave å håndtere oppkoblings-forespørsler fra nye klienter. Dette for å redusere belastningen på selve spillserverene.



Figur 4.1: UML-diagram for AoIM-serverens arkitektur.

class ServerUpdate Denne klassen oppretter en serversocket og har ansvaret for å opprette unicast-forbindelser til klientene som kobler seg til serveren.

- `run()`:
  aksepterer oppkobling av nye klienter og oppretter et objekt av klassen `ServerClientConnection` for hver ny klient, samt at det spinnes en tråd for hvert av disse.

class ServerClientConnection Objekt av denne klassen representerer klientene og opprettes i `ServerUpdate` for hver ny klient som kobler seg til serveren. Server og klient kommuniserer via dette objektet ved hjelp av unicast - dette objektet tar seg m. a. o. av kommunikasjonen mellom en bestemt klient og serveren (via unicast).

- `run()`:
  lytter til oppdateringsmeldinger fra klienten om at klienten har endret sitt sett av fokus- og/eller aurasoner.

- `toClient(String msg)`:
  sender oppdateringer om at andre klienters aura har kommet i fokus. Strengen msg kan ha N,F eller A som prefix. N benyttes når klienten har koblet seg opp og ikke kjenner til trestrukturen for sonene. Basert på resten av pakkens innhold vil så klienten kunne bygge opp et korrekt sonetre. F. eks. så forteller strengen $Nr, r4$ at rotsonen (r) skal splittes og at sonen r4 skal splittes - resultatet blir treet som vist i figur 2.4 på side 20. En streng med prefix F returneres til klienten etter at klienten har blitt lagt til i fokuslisten til en sone. Resten av strengen inneholder ip-adressene til samtlige klienter som er i sonens auraliste (slik at klienten kan opprette unicast-forbindelser til listen over klienter). Strenger med A som prefix returneres til alle klientene i en sones fokusliste hvis klient x har blitt lagt til i sonens auraliste. Resten av strengen vil da kun inneholde ip-adressen til klient x (slik at disse klientene kan opprette unicast-forbindelser til klient x).

class ServerTreeUpdate Denne klassen oppretter en multicastsocket for å kunne oppdatere alle klienter om topologiforandringer i sonetreet. Serveren er eneste sender for multicast-gruppen - den er m. a. o. kildespesifikk (se seksjon 2.5.1)[3].

[3]Alternativt kan det benyttes unicast ved at serveren distribuerer topologiforandringene til et sett av dedikerte klienter (som nevnt i seksjon 2.5.1). Siden dette er en en-til-mange-kommunikasjon er det gunstig å benytte multicast fordi det for et kildespesifikt multicast-tre er lett å finne den korteste veien fra sender til hver enkelt mottaker.

- `sendTopologyUpdateToClients(String msg)`:
  sender en streng som forteller om en sone skal splittes eller fire subsoner smeltes sammen. Et prefix etterfulgt av en sone-id forteller om sonen skal splittes eller om sonens fire bladnoder skal slettes (sammenslåing). Strengen $Sr1$ forteller at sonen r1 skal splittes i fire sub-soner (hvor S er prefix). For sammenslåing benyttes M (*merge*) som prefix.

- `sendEnvironmentUpdateToClients(String msg)`:
  kalles fra updateEnvironment i ServerEnvironment for å distribuere forandringer i omgivelsesvariable til klientene[4].

class ServerZoneTree Objektet av denne klassen har ansvaret for sonetreet og topologiforandringene som vil forekomme under kjøretid.

- `addNewClient(String pos, SCC scc)`[5]:
  kalles kun en gang for hver klient. Når en klient suksessfullt har koblet seg til serveren vil serveren umiddelbart sende en melding til klienten som inneholder topologien til sonetreet. Denne meldingen sendes via metoden `toClient` i `scc` objektet (av klassen `ServerClientConnection`) som er beskrevet ovenfor.

- `updateFocusAuraLists(String zones, SCC scc)`:
  kalles fra en klients `scc`-objekt som følge av at klienten har sendt en oppdatering av sitt sett med fokus- og/eller aurasoner. Strengen $zones$ er på formen $F, x, ..., x, A, x, ..., x, F, x, ..., x, A, x, ..., x$ hvor F er fokus etterfulgt av et sett av soner (hvor x angir sone id) og A er aura (hvor x angir sone id). Første F og A forteller hvilke nye soner som er kommet inn i klientens fokus og aura, mens resten av strengen refererer til sonene som ikke lenger er i klientens fokus/aura. F. eks. vil strengen $F, r2, A, r2, F, r1, A$ fortelle at klienten nå dekker sonen r2 både med sin fokus og med sin aura, og at r1 ikke lenger er i fokus, mens klientens aura ikke har "forlatt" noen soner (se figur 4.2 på neste side). Ut ifra dette oppdateres de forskjellige sonenes fokuslister og auralister. Metoden kaller til slutt `scc.toClient(auraClients)`, hvor $auraClients$ angir ip-adressene til klientene som befinner seg i en, for klienten, ny fokus-sone (i eksemplet ovenfor vil dette gjelde sone r2).

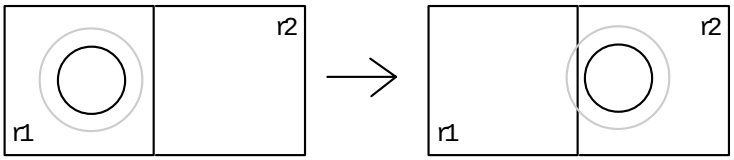
- `split(String id)`:
  splitter en sone hvis både terskelen for maks antall fokusklienter pr. sone *og* terskelen for maks antall auraklienter har blitt overskredet. Denne metoden kalles fra `updateFocusAuraLists` ved at det for hver

[4]Denne metoden er ikke implementert. For simuleringene som behandles i kapittel 5 er omgivelsene konstante for den enkelte simulering. Det er to omgivelsesscenarier som simuleres (normal og tåke), og overganger mellom disse kunne skjedd under en simulerings kjøretid ved å implementere denne metoden.

[5]SCC = ServerClientConnection.

gang en sone får oppdatert sin fokus- eller auraliste gjøres en sjekk på om sonen må splittes . Metoden opperetter fire nye objekter av klassen `ServerZone`. Det gjøres et kall på metoden `sendTopology-UpdateToClients` i objektet av klassen `ServerTreeUpdate` som vil distribuere sonens id til alle klientene ved hjelp av multicast. Slik vil hver klient kunne oppdatere sine lokale kopier av sonetreet.

- `merge(ServerZone zone)`:
  kalles fra $zone$ som er foreldrenoden til en sone som har fått redusert antall fokusklienter eller auraklienter. De fire bladnodene vil så sjekkes for å avgjøre om en sammenslåing er nødvendig (dvs. at det samlede antall fokusklienter og det samlede antall auraklienter i de fire bladnodene har kommet under en gitt terskel). Det gjøres et kall på metoden `sendTopologyUpdateToClients` i objektet av klassen `Server-TreeUpdate` som vil distribuere sonens id til alle klientene ved hjelp av multicast. Metoden `merge` vil rekursivt kalle seg selv for å sjekke om $zone$ og dennes tre søsken også skal sammenslåes. Hvis f. eks. samtlige klienter forlater omgivelsene, så vil denne metoden kalles rekursivt helt til sonetreet kun består av rotsonen.



Figur 4.2: En klients forflytning mellom sonene r1 og r2 som resulterer i et kall på metoden `updateFocusAuraLists`. Grå sirkel er aura, fokus er sort.

class ServerZone Når AoIM-serveren starter opp vil det opprettes en sone - rotsonen. Under kjøretid vil det så bygges opp et dynamisk sonetre bestående av objekter av denne klassen. En sone har to lister: en for alle klienter som har sonen i fokus og en for alle klienter som har sonen dekket av sin aura.

- `addToFocusList(SCC scc)`:
  legger til klienten scc i sonens fokusliste.
- `removeFromFocusList(SCC scc)`:
  fjerner klienten scc fra sonens fokusliste.
- `addToAuraList(SCC scc)`:
  legger til klienten scc i sonens auraliste. Det gjøres et kall på metoden `sendAuraUpdate` som beskrives nedenfor.

- `removeFromAuraList(SCC scc)`:
  fjerner klienten scc fra sonens auraliste.

- `sendAuraUpdate(SCC scc)`:
  kalles av metoden `addToAuraList`. Metoden kaller `toClient` for alle klienter (dvs. objekter av klassen `ServerClientConnection`) som er i fokuslisten til sonen. Pakken som distribueres til disse vil da inneholde ip-adressen til klienten som ble lagt til i auralisten. Det er mulig at klienten som legges til i auralisten til den gitte sonen også skal legges til i auralisten til andre soner (hvis det er flere enn en for klienten ny aurasone i stringen som behandles i metoden `updateFocusAuraLists` i `ClientZoneTree`). En og samme klient kan ha flere av disse sonene som sin fokus-sone. Etter at pakken som er nevnt ovenfor er sendt til en bestemt klient i fokuslisten, vil denne klienten legges til i en liste i objektet til auraklienten, slik at denne kan benyttes for å kontrollere om fokusklienten allerede har blitt tilsendt pakken (så man unngår å sende duplikate meldinger til en og samme klient).

- `getAuraList(String theClient)`:
  kalles som følge av at en klient har fått sonen i sin fokus fra metoden `updateFocusAuraLists i ServerZoneTree`. Metoden returnerer en streng med ip-adressen til alle klientene i sonens auraliste.

class ServerEnvironment Objektet av denne klassen holder informasjon om omgivelsene i simuleringen av typen lysstyrke, tåke, regn, dominerende farger og lignende. Dette er variable som sammen med hver klients personlige variable vil bestemme størrelse på fokus og aura (se figur 2.8 på side 26). Avhengig av hva slags simulering det er - hvor stort geografisk område som simuleres - må det avgjøres hvor mange objekter av klassen `Server-Environment` som trengs (hvor stort område hvert skal representere)[6].

- `updateEnvironment(String msg)`:
  kalles utenfor AoIM-modulen (fra en simulerings-/spillspesifikk applikasjon som benytter seg av AoIM-modulen) for å sette de riktige omgivelsesvariablene under kjøretid. Oppdateringer her vil så kalle metoden `send-EnvironmentUpdateToClients` i `ServerTreeUpdate`, og dermed benytte multicast for å distribuere disse oppdateringene til klientene.

- `getEnvironment()`:
  returnerer omgivelses-variablene for å kunne sendes til én klient - noe som vil være nødvendig når en ny klient har koblet seg opp.

[6] I simuleringene som presenteres i kapittel 5 vil det kun benyttes ett objekt for hele området, og det vil kun være en lysstyrkevariabel (denne lysstyrkevariabelen vil påvirke den initielle fokus og aura for tåkescenariet i simuleringen - se avsnitt 5.1.2 på side 69) som definerer omgivelsene.

4.3 AoIM klientens arkitektur

Oppgavene til AoIM klienten vil her defineres som følger:

- Bygge sonetreet i samsvar med de oppdateringer som sendes fra serveren.
- Regelmessig oppdatere de sett av soner som dekkes av aura og fokus og sende en oppdateringsmelding til serveren hvis listen over fokus og/eller aura-soner er blitt modifisert.
- Opprette forbindelse til nye klienter på bakgrunn av melding fra server om at nye klienter befinner seg innenfor interesseområdet.
- Ved en forbindelse mellom to klienter vil den ene være ansvarlig for å regelmessig sjekke om forbindelsen skal vedvare (sjekke om det fortsatt finnes et felles interesseområde).
- Sende oppdateringspakker[7], alle forbindelsene til en klient i betraktning, til det sett av klienter man er synlig overfor (se figur 2.13 på side 32).
- Få omgivelsesvariable fra server, og på grunnlag av dette definere størrelse på fokus og aura.

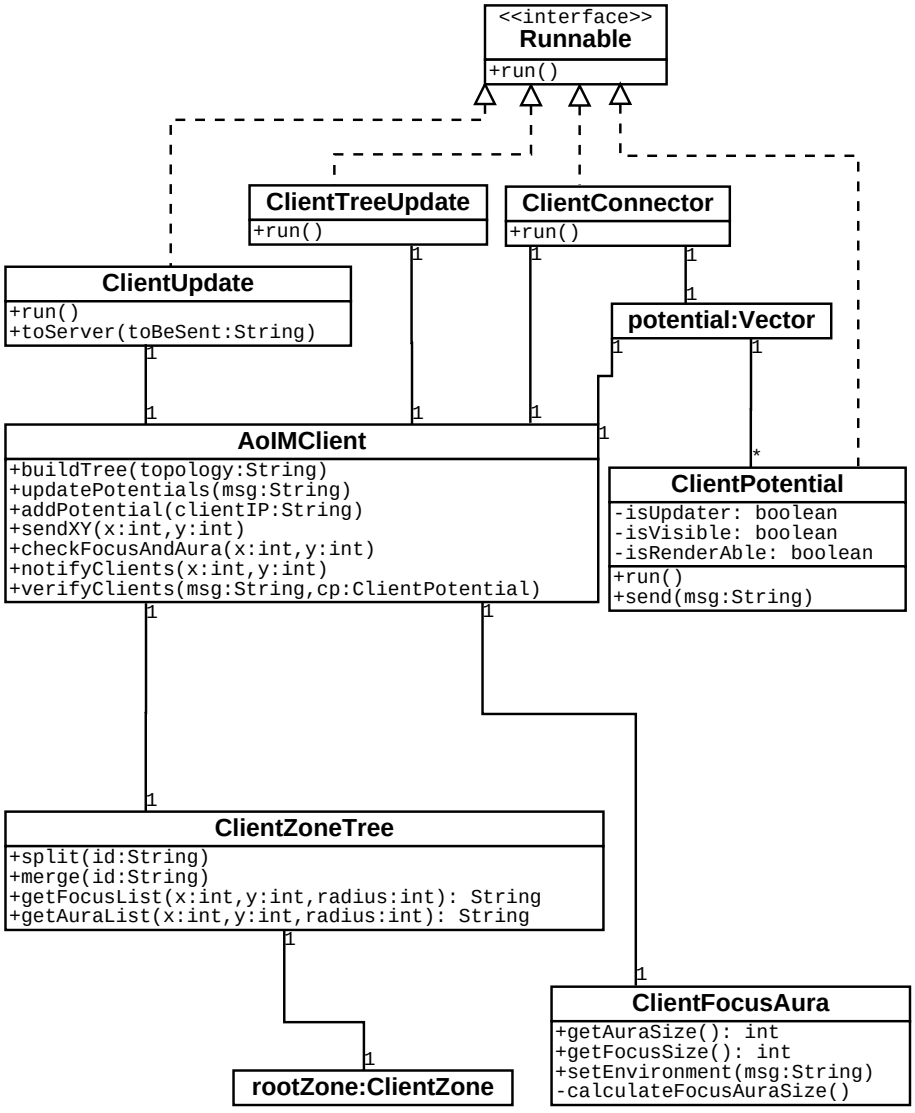
Alternativt kunne serveren alene sitte med sonetreets topologi. Klientne ville da heller kunne sende sin posisjon (koordinater) i omgivelsen til serveren, hvorpå denne da kunne returnert oppdaterte sonelister for fokus og aura til hver enkelt klient. Serveren vil så slippe å sende oppdateringer til klientene vedrørende sonetreets topologi, men dette på bekostning av at det vil sendes flere meldinger fra klientene til server, samt at belastingen på serveren vil øke fordi den må prosessere fokus- og auralistene for alle klientene. Oppdateringen av fokus- og auralister bør derfor gjøres distribuert hos klientene, noe som tilsier at klientene må ha lokale kopier av sonetreet.

Figur 4.3 på neste side viser hvordan klienten er bygget opp - nedenfor følger en beskrivelse av de forskjellige klassene.

class AoIMClient Objektet av denne klassen er hovedkomponenten i klientens del av AoIM-modulen. Herfra administreres sonetreet, listen over fokus- og aurasoner og listen over klienter den gitte klient har forbindelse med.

- `buildTree(String topology)`:
  kalles fra `run` i `ClientUpdate` og tar imot den første meldingen fra serveren etter at klienten har koblet seg til. Innholdet av pakken er en liste over hvilke soner som skal splittes for at topologien til sonetreet

[7] I denne simuleringen begrenser det seg til x- og y-koordinatene for klientens lokasjon.



Figur 4.3: UML-diagram for AoIM-klientens arkitektur.

skal bli korrekt (se metoden `addNewClient` under seksjon 4.2 på side 51).

- `updatePotentials(String msg)`:
  kalles fra `run` i `ClientUpdate` og inneholder auralistene til en eller flere soner som har kommet i klientens fokus. Det er resultatet av en toveiskommunikasjon mellom klient og server, hvor klienten har sendt en oppdatering til serveren over hvilke soner som er i fokus og innenfor aura, og serveren svarer med å sende auralistene over klientene i de nye fokus sonene (se metoden `updateFocusAuraLists` under seksjon 4.2 på side 51). Metoden sjekker om klienten allerede er i listen over *potentials*[8]. Hvis dette ikke er tilfelle vil det opprettes et nytt objekt av klassen `ClientPotential` som sørger for at det opprettes forbindelse mellom klientene. Den andre klienten vil så også opprette et *cp*-objekt slik at begge klientene er representerer hos den andre klienten ved et objekt av denne klassen. Sagt med andre ord: klient x får melding om at klient y sin aura finnes i sonen z som er blitt en ny fokus-sone for x. Klient x oppretter en forbindelse som aksepteres av klient y. Sistnevnte klient får så ansvar med å bryte forbindelsen når det ikke lenger eksisterer noe felles interesseområde mellom x og y.

- `addPotential(String clientIP)`:
  kalles også fra metoden `ClientUpdate` som et resultat av at en klients aura har entret en sone som en gitt klient har i fokus. Strengen $clientIP$ vil kun inneholde én ip-adresse, og samtlige klienter som har sonen i sin fokus vil motta denne strengen fra serveren. Her er det altså én klient som forteller serveren at aura har kommet inn i en ny sone, mens serveren sender beskjed om dette til sonens sett av fokusklienter. I likhet med metoden `updatePotentials` må det sjekkes om det allerede eksisterer en forbindelse med klienten.

- `sendXY(int x, int y)`:
  løper gjennom listen av $potentials$ og sender for en gitt klient lokasjonskoordinater til alle klienter som denne er synlig overfor.

- `checkFocusAndAura(int x, int y)`:
  kalles regelmessig for å oppdatere listen over soner som inngår i klientens fokus og aura. Listene over disse returneres fra `ClientZoneTree`. Det gjøres så en sjekk opp mot de gamle fokus- og auralistene. Soner i de nye listene som ikke finnes i de gamle, og soner i de gamle som ikke finnes i de nye danner så grunnlag for en melding som sendes til

[8]Potentials er listen over alle klientene man har peer-to-peer unicast-forbindelser med. Navnet potentials kommer av at man *potensielt* kan kommunisere med hver enkelt, men at forbindelsene vil være en blanding av enveis og toveiskommunikasjon siden nødvendigvis ikke begge er synlig for hverandre (se figur 2.13 på side 32).

serveren for å oppdatere serversonenes fokus- og auralister. Strengen med disse sonene mottas i `ServerZoneTree` sin metode `updateFocusAuraLists` (se seksjon 4.2 på side 51).

- `notifyClients(int x, int y)`:
  sender klientens posisjon og radius for klientens fokus og aura til alle klientene i potentials. Disse klientene mottar i sin tur denne informasjonen i metoden `verifyClients`, som beskrives nedenfor.
- `verifyClients(String msg, ClientPotential cp)`:
  mottar for alle klientene i potentials informasjon om posisjon, aura- og fokusradius. Poenget er i første instans å avgjøre om forbindelsen til klienten skal opprettholdes - dvs. om det er minst én sone hvor det er match mellom klient *x* sin fokus og en klient *y* sin aura (eller omvendt). I andre instans avgjøres det om klient x er synlig overfor klient y - at x sin aura er i en av y sine fokus-soner. Dette vil avgjøre om det skal sendes oppdateringspakker eller ikke fra x til y. Det siste som avgjøres er om klient x ser klient y - om x har fokus i en av y sine aurasoner. Denne informasjonen trengs utenfor AoIM-modulen der hvor simuleringens grafikk oppdateres og tegnes på skjermen. Hvis begge har falt utenfor hverandres interesseområder, vil den av de to som har fått oppgaven med å bryte forbindelsen gjøre det.

class ClientTreeUpdate Et objekt av denne klassen lager en multicast-socket og sørger for at klienten blir medlem av den samme multicast-gruppen som serveren er medlem av.

- `run()`:
  lytter til multicast-socketen og mottar oppdateringer for sonetreet som er sent fra serveren. Pakken inneholder en string med et prefix og en sone id. Prefix S forteller at sonen id skal splittes, mens prefix M betyr at bladnodene til sone id skal termineres (sammenslåing). Det vil så gjøres et kall på `split(id)` eller `merge(id)` hos `clientZoneTree`. Metoden lytter også til oppdateringspakker for omgivelsesvariable. Disse har prefix E, og innholdet vil så sendes til `updateEnvironment` i `AoIMClient`.

class ClientUpdate Objektet av denne klassen oppretter forbindelsen til serveren og er ansvarlig for kommunikasjonen mellom klient og server (utenom topologi-oppdateringene for sonetreet som beskrevet ovenfor).

- `run()`:
  lytter til meldinger fra serveren. Det er tre typer meldinger:
  - Hele sonetreets struktur, som sendes til metoden `buildTree` i `AoIMClient`.

- Lister over auraklienter i klientens nye fokus-soner, som sendes til metoden `updatePotentials` i `AoIMClient`.
- IP-adressen til en enkelt klient som har entret en av klientens fokus-soner, som igjen trigger et kall på metoden `addPotential` i `AoIMClient`.

- `toServer()`:
  sender meldinger til serveren av type fokus- og aura-oppdateringer.

class ClientConnector Oppgaven til objektet av denne klassen er å opprette en server-socket for å akseptere tilkoblingen av andre klienter (klienter som ønsker å lage en forbindelse fordi en av klientens aura-soner har blitt sammenfallende med en annen klients fokus-sone, som beskrevet i metodene `addPotential` og `updatePotentials` i `AoIMClient`).

- `run()`:
  lytter til socketen og aksepterer nye forbindelser fra andre klienter. Hvis det ikke allerede eksisterer en forbindelse til klienten, vil det opprettes et nytt objekt av klassen `ClientPotential`.

class ClientPotential Når to klienter ønsker å kommunisere med hverandre, blir det hos begge opprettet et objekt av denne klassen for å representere denne forbindelsen (unicast). Klienten som aksepterer forespørselen fra en annen klient om å opprette forbindelse, får ansvaret for å lukke forbindelsen når det blir aktuelt.

- `isUpdater(boolean)`:
  settes til true dersom klienten har ansvaret for å lukke forbindelsen når det ikke lenger eksisterer noe felles interesseområde.

- `isVisible(booloan)`:
  forteller om du er synlig for den andre klienten. Variablelen forteller m. a. o. om informasjonsflyten skal gå én vei, eller begge veier. Denne variabelen oppdateres i metoden `verifyClients` i `AoIMClient` som beskrevet ovenfor.

- `isRenderable(boolean)`:
  forteller om klienten på den andre siden av forbindelsen er synlig for deg, og om den da skal grafisk oppdateres på skjermen. Denne variabelen oppdateres i metoden `verifyClients` i `AoIMClient` som beskrevet ovenfor.

- `run()`:
  lytter til meldinger fra den andre kienten. Meldinger er av type:

- Oppdatering av klientens lokasjon (x- og y-koordinatene). Dette er meldinger man kun sender til dem man er synlig overfor.
- Informasjon om radius til fokus og aura, som sendes videre til metoden `verifyClients` i `AoIMClient` som beskrevet ovenfor.
- Melding om at forbindelsen skal opphøre å eksistere, og at *cp*-objektet kan kastes.

- `send(String msg)`:
  benyttes for å sende meldinger til de potensielle klientene, som mottar disse i metoden run (som beskrevet ovenfor).

class ClientZoneTree Objektet av denne klassen opprettholder et korrekt sonetre på grunnlag av oppdateringsmeldinger fra serveren.

- `split(String id)`:
  splitter sonen id på oppfordring fra serveren.

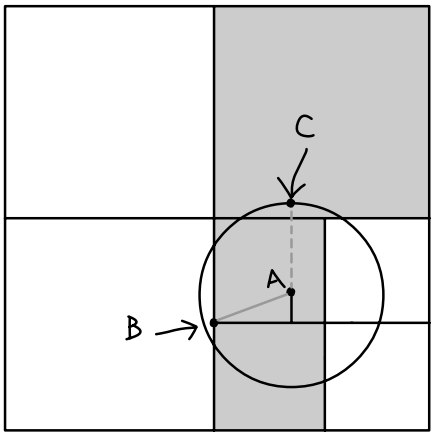
- `merge(String id)`:
  terminerer bladnodene til sonen id på oppfordring fra servren.

- `getFocusList(int x, int y, int radius)`:
  kalles fra metoden `checkFocusAndAura` i `AoIMClient` og returnerer en streng med alle sonene som helt eller delvis dekkes av fokus. Hele treet traverseres, hvorpå bladnodene (de aktive sonene) plukkes ut for å sjekkes om hver enkelt av dem skal legges til i listen. Her ble det laget en algoritme hvor hver sone sjekkes på to forskjellige måter. Først sjekkes det om minst ett av sonens hjørner dekkes av fokus. Et hjørne er innenfor fokus-sirkelen hvis avstanden fra klientens posisjon til hjørnets posisjon (som er hypotenusen i en rettvinklet trekant) er mindre eller lik radius til fokus (se figur 4.4 på neste side). Denne sjekken vil ikke fange opp de sonene hvor fokus har overlapp, men ikke dekker noen av hjørnene. Man må derfor sjekke ett av de fire punktene $x - radius$, $y - radius$, $x + radius$ og $y + radius$ (hvor x og y angir geografisk posisjon og radius er radius for fokus) ligger innenfor sonen. Sirkelen på figur 4.4 på neste side angir fokus til en klient. Punkt A er klientens posisjon. De to minste grå sonene faller innenfor fokus fordi avstanden fra A til hjørnet i punkt B er kortere enn radius til fokus (avstanden, den grå linjen, finner man ved å regne ut hypotenusen til trekanten som vist på figuren). Den store grå sonen er også i fokus selv om ingen av hjørnene er innenfor sirkelen. For å finne ut om denne sonen er innenfor ser man heller på punktet C, som har koordinatene Ax, $(Ay - radius)$. Etter å ha foretatt en slik hjørne-/overlapp-sjekk på alle sonene i figuren, vil man så kunne slå fast at alle sonene er innenfor, utenom sonen øverst til venstre.

- `getAuraList(int x, int y, int radius)`:
  fungerer på samme måte som metoden `getFocusList`, men returnerer en liste over soner som helt eller delvis dekkes av aura.

class ClientFocusAura Objektet av denne klassen har ansvaret for å oppdatere den korrekte størrelsen (radius) for fokus og aura.

- `getFocusSize()`:
  returnerer radius til fokus.
- `getAuraSize()`:
  returnerer radius til aura.
- `setEnvironment(String msg)`:
  kalles for å sette omgivelsesvariable som er i samsvar med serverens omgivelsesvariable. Klientens fokus- og aurastørrelse vil så kalkuleres på nytt.



Figur 4.4: Hvordan fokus soner og aura soner regnes ut.

4.4 En applikasjons bruk av AoIM-modulen

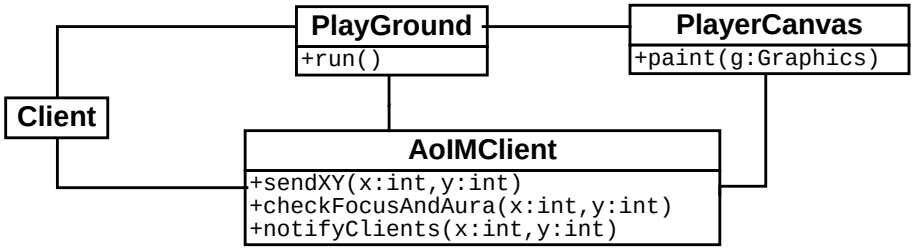
Ovenfor ble det beskrevet hvordan serversiden og klientsiden for AoIM-modulen ser ut og fungerer. Det neste er å se på hvordan denne kan benyttes i en applikasjon hvor det spill-spesifikke ligger. Poenget med AoIM-modulen er jo at den skal være så generisk som overhodet mulig, slik at den kan benyttes i spill/simuleringer av varierende type uten at man skal behøve å gå inn og modifisere modulen.

AoIM-modulen oppretter forbindelse til serveren og oppretter forbindelse til de klienter som etter beskrivelsene over har interesse for hverandre - m. a. o. avstandsfiltreringen, og bestemmer om kommunikasjonen skal være enveis eller toveis. Disse hovedoppgavene vil være felles for alle applikasjoner som ønsker å benytte AoIM-programvaren, men det vil være fornuftig å definere visse aspekter ved den som må konfigureres før den tas i bruk for en bestemt simulering. Ved hjelp av et grensesnitt (grafisk eller bare en fil man editerer direkte) vil man kunne bestemme f. eks. maksimaldybden på sonetreet (hvor små soner man vil tillate), hvilke attributter som skal være gjeldende for å kalkulere størrelsen på fokus og aura og hvor mange klienter man skal tillate at assosieres sin fokus eller aura med en sone før man splitter den.

4.4.1 Arkitekturen til en enkel simulering

Vi vil her se på hvordan AoIM-modulen kan benyttes sammen med en spillspesifikk applikasjon kalt $PlayGround$. Dette er et klientprogram hvor man kan bevege seg over en kvadratisk flate, og hvor man skal kunne se bevegelsene til andre klienter som også kjører applikasjonen (m. a. o. de klientene man har et felles interesseområde med).

Oppgaven til AoIM-modulen blir her da å administrere kommunikasjonen mellom klienten og serveren, samt klientene seg i mellom. Diagrammet på figur 4.5 viser arkitekturen til klientsiden.



Figur 4.5: Arkitektur som viser hvordan AoIM-modulen kan benyttes. Metodene som vises i AoIMClient definerer grensesnittet mot PlayGround, som inneholder det simuleringsspesifikke.

Klienten oppretter en instans av klassen `AoIMClient` og `PlayGround`. Sistnevnte objekt inneholder som nevnt det simulerings-spesifikke som i dette tilfellet begrenser seg til å lese av brukerens interaksjon med tastaturet som igjen bestemmer brukerens koordinater i det kvadratiske, virtuelle, området[9]. Klientens koordinater vil så danne grunnlaget for avstandsfiltreringen.

[9]Arkitekturen ble først implementert med tanke på at en klient skulle kunne styre sin avatar i omgivelsene. For simuleringene i kapittel 5 er ingen av klientene menneskestyrte.

Beskrivelse av en runde i spill-løkken

Under vises spill-løkken[10] i metoden `run` i `PlayGround`:

```
canvas.repaint();
if(checkFAALim == AoIMVar.CHECK_FAA_LIM){
    ac.checkFocusAndAura(x_pos,y_pos);
    checkFAALim = 0;
}
if(checkClientsLim == AoIMVar.CHECK_CLIENTS_LIM){
    ac.notifyClients(x_pos,y_pos);
    checkClientsLim = 0;
}
try{
    th.sleep(200);
}catch(InterruptedException e){}
if(checkMoveLim == AoIMVar.CLIENT_SPEED){
    moveAcross();
    ac.sendXY(x_pos,y_pos);
    checkMoveLim = 0;
}
checkFAALim++;
checkClientsLim++;
checkMoveLim++;
```

Løkken begynner med å kalle `repaint` metoden i $canvas$[11] som er en instans av klassen `PlayerCanvas`. Skjermbildet oppdateres ved at brukerens grafiske representasjon tegnes i samsvar med koordinatene som hentes fra `PlayGround`. Metoden `repaint` vil så hente ut listen over potentials fra AoIMClient. For klientene som måtte befinne seg i listen vil de som har attributten *renderable* satt til sann (de som er synlige for brukeren), bli tegnet ut på skjermen. Skjermbildet består da av en kvadratisk flate som skal simulere en virtuell omgivelse hvor det tegnes inn små prikker som representerer klientene (se kapittel 5 angående simuleringens omgivelser). I tillegg tegnes også sonenes grenser (sonetreet) og den lokale klientens aura- og fokus-sirkler (se figur 5.2 på side 70).

Det neste som skjer er at det gjøres en sjekk på om settet av fokus-soner og aurasoner er forandret. I praksis vil det ikke skje så ofte at fokus-sonene og aura-sonene vil forandre seg, så her vil det være fornuftig å sette en terskel

[10] En spill-løkke (eng. *gameloop*) er noe man finner i nesten alle spill. Den inneholder en serie med metoder, eller metodekall som mottar *input* og viser *output* til spilleren - oppdateringen av spillet. Denne løkken vil typisk gå fra man starter spillet til man avslutter det.

[11] $Canvas$ er en komponent i Java som representerer et blankt rektangulært område som en applikasjon kan tegne på eller fange opp interaksjoner fra brukeren.

for når dette skal skje (variablelen CHECK_FAA_LIM i klassen AoIMVar[12]). Denne terskelen kan nås etter et bestemt tidsintervall (er hva som benyttes i implementasjonen) eller etter at brukeren har flyttet seg en bestemt avstand i én retning. I en virtuell omgivelse vil det være stor sjanse for at klienter til tider beveger seg lite (f.eks. hvis en klient er engasjert i en samtale eller beveger seg innenfor et mindre, avlukket område). Derfor er det nok mest hensiktsmessig å gjøre sjekken på fokus- og aurasoner basert på bevegelse (fremfor tid). Det kan også være fornuftig i tillegg å la en splitt eller sammenslåing av soner trigge denne sjekken.

Figur 4.6 på neste side viser et scenarium hvor auraen til klient a (grå sirkel) har kommet over sone r2. Aktivitetsdiagrammet viser gangen fra kallet på metoden `checkFokusAndAura` til opprettelsen av en forbindelse mellom klient a og klient b.

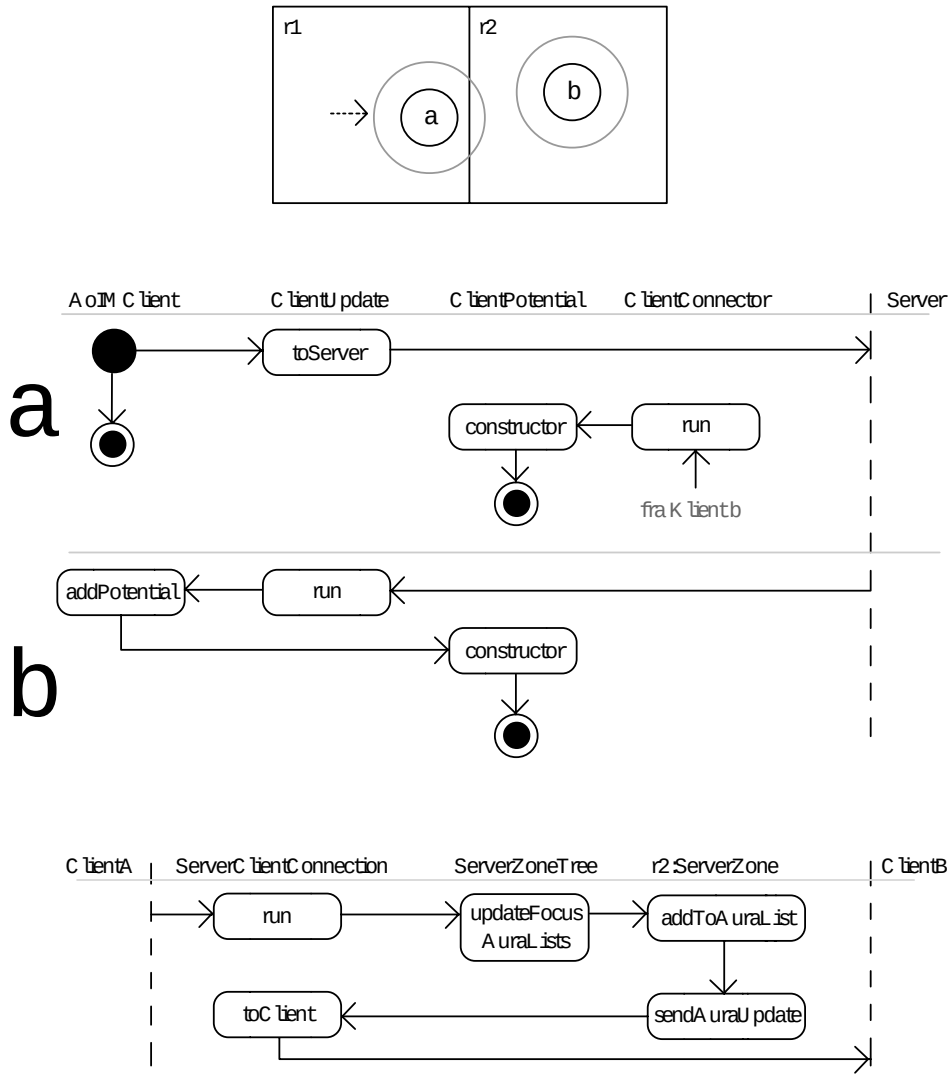
Den øverste delen av aktivitetsdiagrammet viser a sitt kall på `checkFocusAndAura`. Metoden vil terminere i en lovlig slutttilstand hvis settet av fokus- og aurasoner er det samme som før metodekallet. Hvis ikke vil oppdateringene sendes til serveren via metoden `toServer` i `ClientUpdate` (i dette tilfellet vil pakkens innhold være $F, Ar2, F, A$, som forteller at klient a skal legges til i auralisten til sone r2.

Nederste del av diagrammet viser hvordan pakken kommer inn i metoden `run` i `SCC` (objektet av denne klassen som represtenterer klient a hos serveren), hvordan klienten legges til i auralisten til r2 og hvordan denne oppdateringen distribueres til klientene i fokuslisten til r2 ved et kall på metoden `toClient` i `SCC` (i dette tilfellet kun i objektet som representerer klient b), hvor 'A<a's IP-adresse>' er innholdet i pakken.

Pakken fra serveren blir mottatt i `run` i `ClientUpdate` hos klient b (diagrammet i midten). Så sant a og b ikke har et felles interesseområde (m. a. o. ingen forbindelse seg imellom) vil b opprette et objekt av klassen `ClientPotential`, som igjen vil forsøke å opprette forbindelse til klient a. Klient a mottar denne oppfordringen i metoden `run` i `ClientConnector` som sørger for at a også oppretter et objekt av klassen `ClientPotential`. Forbindelsen mellom klientene a og b er dermed opprettet, og den er representert i begge ender ved objektene av klassen `ClientPotenital`.

Det neste som skjer i spill-løkken er at det gjøres en sjekk på om brukeren fortsatt har et felles interesseområde med samtlige av klientene i $potentials$ - metoden `notifyClients`. På lik linje med oppdateringen av fokus- og aurasoner, er det også her satt en grense for hvor ofte det er hensiktsmessig å gjøre denne sjekken. Det koster mindre å opprettholde en forbindelse noe lenger enn nødvendig, enn å skulle bruke ressurser på å gjøre en sjekk på klientene for hver runde i spill-løkken da dette vil øke nettverkstrafikken.

[12]Klassen `AoIMVar` består av statiske variable som definerer f. eks. klientens hastighet, grensene for når settet av aura- og fokus-soner skal sjekkes, maksimaldybden til sonetreet og grensene for splitting/sammenslåing av soner.



Figur 4.6: Tredelt aktivitetsdiagram for metoden checkFokusAndAura, hvor øverste del representerer klient a, midterste del klient b og nederste del serveren. Sonene r1 og r2 viser situasjonen som trigger hendelsesforløpet i diagrammet.

Etter at de potensielle klientene er verifisert oppdaterer klienten sin lokasjon. Dette skjer ved at det gjøres et kall på en metode. Det er implementert to metoder for å flytte klienten i omgivelsene. Den som er angitt her er metoden `moveAcross` som ganske enkelt sørger for at klienten flytter seg diagonalt over planet (se kapittel 5).

Det neste som skjer er at brukerens koordinater blir distribuert via metoden `sendXY` til klientene i potentials som brukeren er synlig overfor, slik at disse også kan oppdatere sine grafiske representasjoner av klienten de har i sine lister over potensielle klienter.

Det siste som skjer er at alle grensevariablene som trigger sjekk av fokus-/aurasoner, oppdatering av klienter og oppdatering av egen lokasjon inkrementeres.

KAPITTEL 5

Simulering

5.1 Innledning

I dette kapitlet vil vi se på resultater av simuleringer med utgangspunkt i implementasjonen som ble beskrevet i forrige kapittel. Simuleringene skal m. a. o. teste ut prinsippene om *avstandsfiltrering* som har blitt presentert i denne oppgaven - en dynamisk soneinndeling kombinert med dynamisk fokus og aura.

Først vil vi se på simuleringer for å vurdere *hvilke kriterier* som bør ligge til grunn for splitting og sammenslåing av soner (som ble diskutert i seksjon 2.4.2), dernest vil den *dynamiske* soneinndelingen bli testet opp mot en *statisk* for å kunne vurdere om den dynamiske er å foretrekke fremfor en statisk. Til slutt vil vi se på hvor godt avstandsfiltreringen skalerer.

Simuleringene vil så gi en indikasjon på hvor effektiv denne avstandsfiltreringen er: i hvor stor grad den reduserer antall klientforbindelser en gitt klient til enhver tid har og om hvor stor trafikken mellom server og hver klient blir.

5.1.1 Forutsetninger og begrensninger

Grunnlaget for simuleringen bygger på den arkitekturen som er beskrevet i seksjon 4.4, hvor AoIM-modulen benyttes sammen med en applikasjon som representerer en virtuell omgivelse. Denne applikasjonen har et meget begrenset grafisk grensesnitt - ideelt sett burde AoIM-modulen blitt testet opp mot en virtuell omgivelse i tre dimensjoner slik vi kjenner den fra dagens nettverksspill.

Prinsippene for vår todimensjonale omgivelse vil uansett være det samme som for en tredimensjonal, grafisk, omgivelse hva interessefiltreringen angår.

Problemet blir å si noe om hvordan filtreringen påvirker det visuelle - hvordan brukeren som interagerer med simuleringen faktisk opplever den. Eksempel på dette kan være at man velger for små verdier til radius til fokus og aura. Man vil da kunne komme i skade for å filtrere bort for mye. Dette vil da kunne føre til at to klienter får et felles interesseområde, som vil gjøre dem synlig for hverandre for "sent". Visuelt vil dette oppleves som om deres grafiske representasjoner (avatarer) dukker opp fra intet (såkalt $pop - up$ grafikk)[1].

Det grafiske brukergrensesnittet begrenser seg i vårt tilfelle til en kvadratisk flate hvor hver klient er representert med et punkt i det todimensjonale plan. Ut i fra dette blir det dermed umulig å skulle si hvorvidt simuleringen forringer den visuelle opplevelsen.

Basert på forsøk gjort med det grafiske 3D-verktøyet $3D\ Studio\ Max$[2] gjøres det her to forutsetninger om hvor langt man kan "se" i den virtuelle omgivelsen denne simuleringen er en representasjon for - vi gjør oss m. a. o. *antagelser* for hvilke verdier som kan være fornuftige å benytte i en omgivelse representert i 3D-grafikk. Resultatene vil vi senere benytte for å bestemme størrelsene til fokus og aura for klientene som er med i simuleringen.

Ved å modellere en virtuell omgivelse som kun består av en flate (bakke) og en bakgrunn (himmel), hvor det så plasseres tre avatarer (som representerer mennesker på 180 cm), kan man så si noe om hvor synlige avatarene er ved bestemte avstander (se figur 5.1 på neste side). Settingen i denne moddeleringen kan vi da si representerer optimale forhold mht. hvor langt man kan se uten hjelp av eksterne hjelpemidler (f. eks. en kikkert) hvor det heller ikke eksisterer konkurrerende stimuli som kan være med på å gjøre avatarene mindre synlige. To scenarier ble testet med en bildeoppløsning på 800x600:

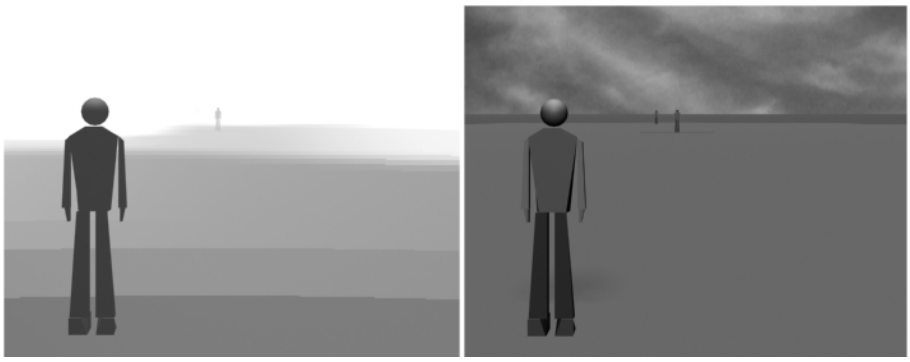
- *Dagslys*: Ved å simulere dagslys, og hvor det ikke ble lagt til atmosfæriske effekter, viste det seg at en avatar med en avstand på over 700 meter fra kamera ble vanskelig å se. Ved å flytte avataren til avstander mellom 600-1000 meter fra kamera var det vanskelig å se noen forskjell på det visuelle resultatet - avataren ble representert med bare noen få pixler, og det var vanskelig å se *hva* den representerte.

- *Tåke*: Ved å legge til tåke[3] som atmosfærisk effekt ble synligheten for avataren drastisk svekket. Ved en avstand på 160 meter kunne man ikke lenger se avataren (se figuren - i scenen med tåkeeffekt synes ikke den midterste avataren).

[1] Et unntak er selvfølgelig hvis en avatar har en egenskap som tilsier at den helt plutselig kommer til syne for andre klienter.

[2] 3ds Max er et av de mest brukte 3D modelleringsverktøy innenfor film (f.eks. $South\ Park$, $Godzilla$ og $Lost\ in\ Space$), tv (f.eks. $The\ Simpsons$ og $Ally\ McBeal$) og spill-utvikling (f.eks. $Tomb\ Raider\ I$, II og III og $Need\ for\ Speed$).

[3] Det ble benyttet en standard tåkeeffekt, hvor verdien for tetthet ble satt til 50/100.



Figur 5.1: To scener fra 3ds Max; den til høyre simulerer vanlige lysforhold, mens den til venstre har en atmosfærisk tåkeeffekt. Scenene har tre avatarer hvor de to nærmest kamera har en avstand på 10 og 50 meter. En tredje avatar kan skimtes mellom disse med en avstand på 160 meter (scenen til høyre).

5.1.2 Simuleringens omgivelser

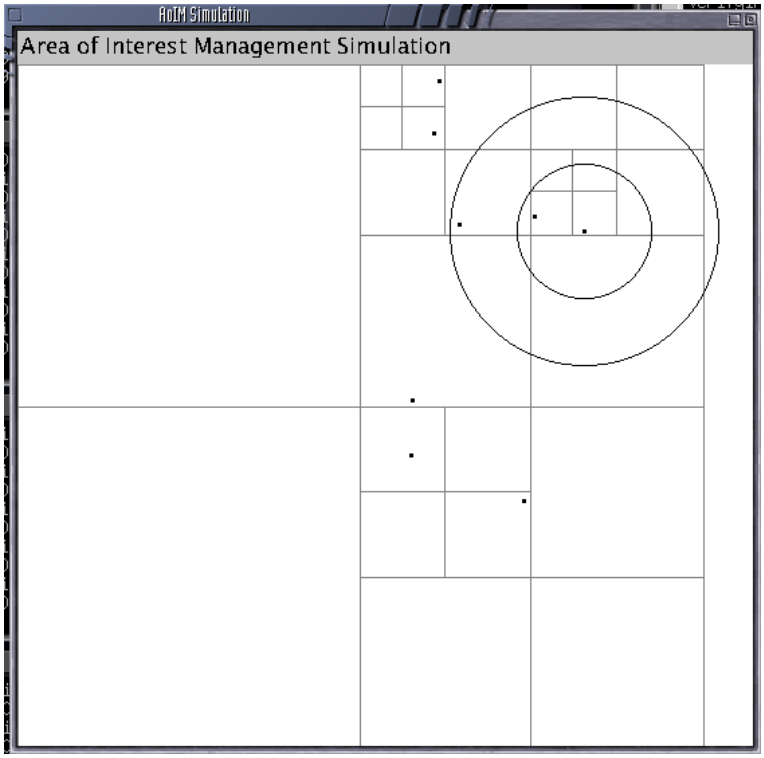
Arkitekturen i seksjon 4.4 presenterte applikasjonen $PlayGround$ til bruk sammen med AoIM-modulen. Denne applikasjonen kan sies å være en virtuell omgivelse i sin "enkleste" form. Det grafiske grensesnittet består av et kvadratisk felt på 512x512 pixler (altså en 2D-verden - i samsvar med at AoIM-implementasjonen ble designet til å brukes med en verden med to dimensjoner). Dette feltet representerer en omgivelse på 4x4 km (dvs. 1 pixel = 7,8 meter).

Denne omgivelsen blir befolket av klienter som visuelt fremstår som små prikker. Hver bruker vil også se sin egen fokus og aura og soneinndelingen, i tillegg til de klientene man kan se (de klienter som dekker brukerens fokussoner med sin aura) . Figur 5.2 på neste side viser skjermbildet til simuleringen under kjøretid. Andre objekter finnes ikke i omgivelsene.

Simuleringene ble kjørt med 100 *AI-klienter*[4], hvorav en dannet grunnlag for simuleringsresultatene. Denne var også den eneste som ble kjørt med grafisk grensesnitt slik at simuleringene kunne overvåkes (f. eks. sjekke at brukeren ikke kommuniserte med en klient det ikke fantes noe felles interesseområde med).

Klientsiden og serversiden ble først programmert slik at kommunikasjonen faktisk benyttet seg av sockets for unicast- og multicastkommunikasjon. All testing under implementasjonsfasen ble gjort med en serverapplikasjon og et fåtall klientapplikasjoner som kjørte på separate maskiner. Men for selve simuleringene, hvis resultater blir presentert i dette kapitlet, ble implementasjonen skrevet om slik at alle klientene og serveren kunne kjøres som

[4]AI (Artificial Intellegence - kunstig intelligens) er betegnelsen som benyttes for å beskrive oppførselen til ikke-menneske styrte avatarer.



Figur 5.2: Simuleringens grafiske grensesnitt.

separate tråder på én maskin (all nettverks-kommunikasjon ble byttet ut med vanlige metodekall). Dermed ble det enklere å kjøre simuleringer med 100 klienter (på én maskin i stedet for 100 maskiner som da ville ha kjørt hver sin klientapplikasjon og én maskin som ville ha kjørt serverapplikasjonen). Dette kunne gjøres fordi forsinkelse og pakketap ikke er relevant for simuleringene og tolkningene av resultatene[5].

Klientene

For å kunne teste ut prinsippene om variabel størrelse på fokus og aura deles klientene i simuleringen inn i to grupper. Disse kaller vi $alver$ og $dverger$. Valget av to forskjellige typer klienter ble gjort med tanke på å kunne velge forskjellige verdier for fokus og aura for de respektive gruppene. Termene alv og dverg ble valgt for å lettere kunne assosiere de to gruppene med de

[5]Vi forutsetter da at forsinkelse og pakketap ikke vil bety noe for metodene. F. eks. så vil det kunne by på problemer hvis det er store variasjoner m. h. t. forsinkelse mellom klientene og server hva topologien til sonetreet angår. Man vil da kanskje måtte ta i bruk metoder for at oppdateringer relatert til sonetreet synkroniseres, men dette vil ikke bli adressert i denne oppgaven.

egenskapene de gis lenger ned (som her er verdier for fokus og aura).

Ut ifra forsøket med $3D\ Studio\ Max$ kan vi bestemme at representasjoner for mennesker, i 3D-omgivelsen som ble modellert, ikke ser hverandre på avstander større enn 700 meter. Basert på dette kan vi så, kombinert med formelen for hvordan man kan bestemme den initielle fokus og aura for en gitt klient (se seksjon 2.4.1 på side 27), velge fokus- og auraverdier for henholdsvis alver og dverger:

- For alvene setter vi her maksimaldistansen til 800 meter, som betyr at en alv kan se en klient på sin egen størrelse på den distansen. Vi tar her høyde for at alver ser godt, men ikke synes like godt. Derfor velger vi å sette initiell fokus til 600 meter, og initiell aura til 200 meter (200 + 600 = 800 - som er i samsvar med formelen). Ved å legge til en tåkeeffekt setter vi maksimaldistansen til 200 meter. Siden forholdet for den initielle aura og fokus er 1:3, setter vi så 'tåkefokus' til 150 meter og 'tåkeaura' til 50 meter.

- Dvergene er kortere enn alvene, og vi setter her deres maksimaldistanse til 600 meter. Vi bestemmer så at disse dvergene ser dårligere i forhold til hvor godt de synes, og setter initiell aura til 400 meter og initiell fokus til 200 meter. Med tåkeeffekten blir så fokus 67 meter og aura 133 meter.

Dette betyr at f. eks. alver vil observere dverger ved en avstand på 1000 meter, mens dvergene ikke sanser alvene på avstander over 400 meter under normale forhold. Disse verdiene vil så synke til 280 meter og 115 meter når vi legger til tåke som atmosfærisk effekt. Utover disse antagelsene ville det også vært naturlig med variasjoner alvene seg imellom og dvergene seg imellom - verdiene som er valgt her kan da heller betraktes for gjennomsnittsverdier for disse to gruppene. Tabell 5.1 oppsummerer fokus- og auraverdiene for alver og dverger.

Både alver og dverger beveger seg i omgivelsene med en fart på 10 km/t, som blir ca. 2,8 sekunder pr. pixel.

	initiell fokus	**initiell aura**	**fokus (tåke)**	**aura (tåke)**
alv	600 m	200 m	150 m	50 m
dverg	200 m	400 m	67 m	133 m

Tabell 5.1: Fokus og aura for alver og dverger som benyttes i simuleringen.

Fire scenarier for simulering

Som nevnt ovenfor vil vi benytte to forskjellige omgivelser for simuleringene: normale og tåke, hvor ovenfornevnte verdier for fokus og aura vil være gjeldene. I praksis er det da lysforhold som simuleres. Fokus og aura blir dermed

dynamisk når vi setter disse to omgivelsene opp mot hverandre - fokus og aura vil ikke forandre seg under en gitt simulering.

For hver av de to omgivelsene vil det så bli testet to forskjellige bevegelsesmønstre for klientene:

- *Tilfeldig*: Her vil klientene plasseres ut i omgivelsene på tilfeldige steder og dernest følge et tilfeldig bevegelsesmønster[6]. Klienter som beveger seg over grensene til omgivelsene, vil dukke opp igjen på motsatt side og kan betraktes som tilfeller hvor en klient forlater simuleringen, mens en annen kommer til.

- *Målbevisst*: Her vil klientene fordeles på fire forskjellige steder idet de entrer omgivelsene. Klientene vil så bevege seg innenfor en liten radius til disse punktene. Dette scenariet vil dermed simulere tilfellet hvor klienter klynger seg innenfor et mindre område.

Det er for det målbevisste scenariet den dynamiske soneinndelingen har sin styrke, og vi antar derfor at den vil overgå den statiske.

5.1.3 Målevariable

For å kunne fastslå nytteverdien av denne arkitekturen vil vi her se på resultatet av flere simuleringer hvor trafikken mellom server og en bestemt AI-klient måles, samt trafikken mellom alle klientene og den bestemte klienten:

- *Server-Klient*: Fra klienten vil det måles frekvensen av meldinger som inneholder en oppdatert liste over fokus- og aurasoner. Fra server til klient vil multicast-pakkene for splitt og sammensmelting måles, samt unicast-pakkene som forteller en klient om nye klienter som skal legges til i, eller slettes fra, listen av potensielle klienter.

- *Klient-Klient*: For klientene seg imellom vil det måles hvor mange klienter en bestemt klient har en forbindelse til, og for disse om kommunikasjonen er en-veis (to klienter har overlapp med den ene klientens aura og den andre klientens fokus for en eller flere av de to klientenes felles interessesoner) eller to-veis (det eksisterer en eller flere felles interessesoner for to klienter hvor det er overlapp av både fokus og aura for begge).

[6] Algoritmen for dette bevegelsesmønstret kalles for hver runde i spill-løkken. Med én prosent sannsynlighet vil klienten stoppe opp, med 89 prosent sannsynlighet vil klienten fortsette i samme retning som ved forrige sjekk (forrige runde i spill-løkken) og med 10 prosent sannsynlighet vil klienten fortsette bevegelsen i en ny retning (med like stor sannsynlighet for de åtte mulige retningene. Med retninger menes her oppover, nedover, til venstre, til høyre samt de fire diagonalretningene).

5.2 Simuleringsresultater

Først vil vi kjøre simuleringer for å teste ut kriteriene for splitting og sammenslåing av soner som ble diskutert i seksjon 2.4.2.

Dernest vil vi kjøre simuleringer hvor det benyttes en statisk sone-inndeling med forskjellige dybder for sonetreet, for å kunne vurdere hvilke fordeler den dynamiske inndelingen kan ha sammenlignet med den statiske.

Til slutt vil vi se på hvordan implementasjonen skalerer ved å teste applikasjonen med forskjellig antall klienter (mellom 100 og 500).

5.2.1 Kriterier for splitting og sammenslåing av soner

Som nevnt i seksjon 2.4.2 vil det for vår implementasjon være hensiktsmessig å splitte eller slå sammen soner basert på hvor mange klienter som assosierer sin fokus og aura med hver enkelt sone (til forskjell for hva som er foreslått av H. A. Abrams - se seksjon 2.2).

I denne seksjonen vil vi se på resultater av simuleringer hvor splitting og sammenslåing trigges av tre forskjellige kriterier:

- Hvor mange klienter som dekker sonen med sin fokus (en terskel).
- Hvor mange klienter som dekker sonen med sin aura (en terskel).
- Hvor mange klienter som dekker sonen med sin fokus *og* hvor mange som dekker sonen med sin aura (to terskler).

Dette for å kunne slå fast om det er hensiktsmessig å basere splitting/-sammenslåing på både fokus og aura, eller om det er en bedre løsning å splitte/slå sammen ved å kun betrakte klienters fokus eller aura.

For hver av disse ble det gjort simuleringer med tre forskjellige grenser for når en sone skal splittes eller for når fire soner skal terminere til en sone. Disse grensene ble henholdsvis satt til 10, 20 og 30. Eksempelvis vil det da for den første verdien skje en splitt av en sone dersom sonen assosieres med 10 klienters fokus (fokus som kriterium), 10 klienters aura (aura som kriterium) eller 10 klienters fokus *og* 10 klienters aura (fokus og aura som kriterium).

Det at en sone splittes opp ved at man tar i betraktning både hvor mange som har den i fokus og aura betyr at en gitt klient kan ha *både* fokus- og auraoverlapp med sonen, eller *bare* fokus- eller *bare* auraoverlapp. Poenget er at summen av antall fokus betraktes (her 10, 20 og 30), samt summen av antall aura (også 10, 20 og 30). En splitt kan m. a. o. forekomme for verdier mellom N og $2 * N$, hvor N er antall klienter som assosieres med sonen, og hvor hver klient assosieres med aura eller fokus, eller både aura og fokus.

Simuleringene for å sjekke de tre forskjellige kriteriene opp mot hverandre ble kjørt med omgivelser hvor 98 AI-klienter (49 $dverger$ og 49 $alver$)

beveget seg i tilfeldige retninger i planet. To AI-klienter (en dverg og en alv) beveget seg diagonalt over planet, og det er alven som dannet grunnlaget for simuleringsresultatene[7].

Figur 5.3 på neste side viser resultatene av simuleringene. Grafene (representert for alven) viser henholdsvis hvor mange potensielle klienter (hvor mange klienter en gitt klient til enhver tid har unicast-forbindelser til), og hvor mange klienter alven var synlig overfor (hvor mange klienter alven sender oppdateringspakker til). Dermed er vi i stand til å kunne se om alven sender oppdateringsmeldinger til færre klienter sammenlignet med hvor mange forbindelser den til enhver tid har. Vi forventer at antall synlige klienter skal være lavere enn antall potensielle fordi alven har en fokus som er betraktelig større enn aura (se tabell 5.1), og siden dvergen har en større aura enn fokus skal det oppstå tilfeller hvor en gitt alv kan se en eller flere dverger og hvor alven ikke er synlig overfor disse. Simuleringene ble kjørt under både normale forhold og under tåke. X-aksen representerer de tre grensene for når splitting av soner skal forekomme (10, 20 og 30).

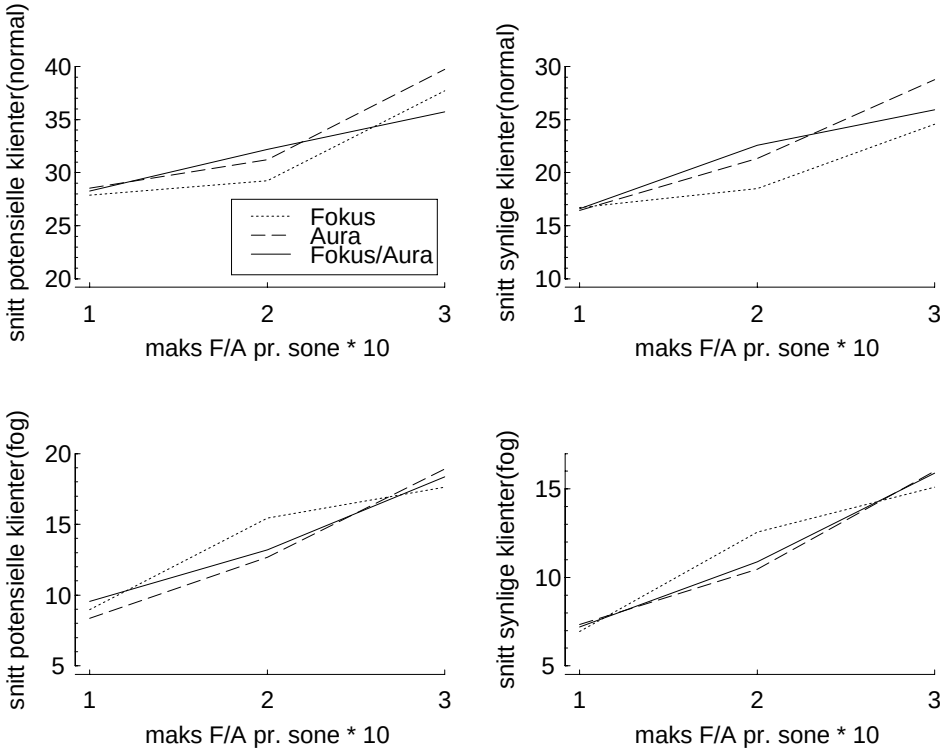
Trafikk mellom klientene

Grafene viser at de tre forskjellige kriteriene gir ganske så like resultater under simuleringens forhold. Det betyr at man ved å betrakte både fokus og aura får et like godt resultat sammenlignet med kun å betrakte fokus eller kun aura. Det viktige her er at man skaper et *strengere* kriterium når man har to terskler - en for fokus og en for aura, siden man her ikke vil kunne splitte en sone hvis man f. eks. kun overskrider grensen for hvor mange fokus som kan assosieres med en sone (se figur 2.10 på side 29). Hvis en sone f. eks. overlappes av 100 klienters fokus mens ingen har den i sin aura, er det ingen hensikt å splitte sonen, siden en sone som kun assosieres med klienters fokus ikke vil opptre som et interesseområde mellom par av klienter. Et eksempel på et slikt scenarium kan være hvis mange klienter klynger seg inntil en vegg eller lignende. Hvis da alle klientene har større fokus enn aura vil det på andre siden finnes et kvadratisk eller rektangulært område som kun dekkes av klienters fokus (se figur 5.4 på neste side).

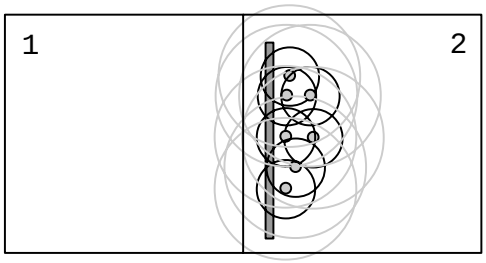
Forskjellige størrelser på fokus og aura er hensiktsmessig

Grafene på figur 5.3 viser at det for både normale forhold og for tåkeforhold er færre klienter det sendes oppdateringspakker til (de man er synlig

[7]Bevegelsen langs diagonalen til omgivelsene ble valgt for å lettere kunne sammenligne resultatene. Skulle klienten som dannet grunnlaget for resultatene beveget seg etter det tilfeldige mønsteret ville også resultatene blitt for tilfeldige. Dessuten tilsier dette bevegelsesmønsteret for en klient at den vil måtte krysse flere soner (som vil føre til oppdateringer mht. settet av fokus- og auralister). For å ha muligheten å sammenligne resultatene for alv og dverg seg imellom (men som denne oppgaven ikke gjør) ble det benyttet både en dverg og en alv til å bevege seg langs diagonalen.



Figur 5.3: Klientforbindelser for alven. X-aksene representerer de tre forskjellige tersklene for splitting (10, 20 og 30) for de tre forskjellige kriteriene (fokus, aura både fokus og aura). Y-aksene på plotene til venstre viser totalt hvor mange klientforbindelser alven hadde i snitt (potensielle), mens de to plotene til høyre viser hvor mange av de potensielle alven i snitt sendte oppdateringspakker til - (hvor mange alven var synlig overfor). De to øverste plotene representerer normale forhold, mens de to nederste representerer tåkeforhold.



Figur 5.4: Klienter som flokker seg inntil en vegg hvor grå sirkler representerer fokus og sort representerer aura. Samme hvor mange klienter som kommer til på høyre side av veggen (vist på figuren som en tykk, grå strek) vil det ikke være grunnlag for å splitte sone 1 (så lenge den kun assosieres med klientenes fokus).

overfor) sammenlignet med det totale antall klientforbindelser (de potensielle). Ved maksimalgrense på 10 fokus og aura pr. sone sender alven til 59% av de potensielle (under normale forhold) og til 75% av de potensielle (under tåkeforhold). For maksimalgrense på 20 er tallene 70% og 82% - for maksimalgrense på 30 er tallene 73% og 87%.

Dette viser at det tjener sin hensikt å benytte fokus og aura med forskjellig radius. I dette tilfellet er aura mindre enn fokus (alven), noe som resulterer i at man er synlig overfor langt færre enn de man selv ser - noe som er et forventet resultat ut i fra prinsippene som ble presentert i kapittel 2.

Trafikk mellom klient og server

Det vi nå har sett er at man ved å benytte to terskler (fokus og aura) ikke vil øke antall forbindelser mellom klientene sammenlignet med å benytte kun én terskel (fokus eller aura), men vi må også ta i betraktning hvorvidt valget av to terskler øker trafikken mellom server og klient. Grafene på figur 5.5 på neste side viser trafikken mellom en klient (alven) og server. De fire øverste viser antall splitt og sammenslåinger av soner (snitt pr. min) for henholdsvis normale forhold og for tåke. Dette blir meldinger fra server til hver klient (via multicast).

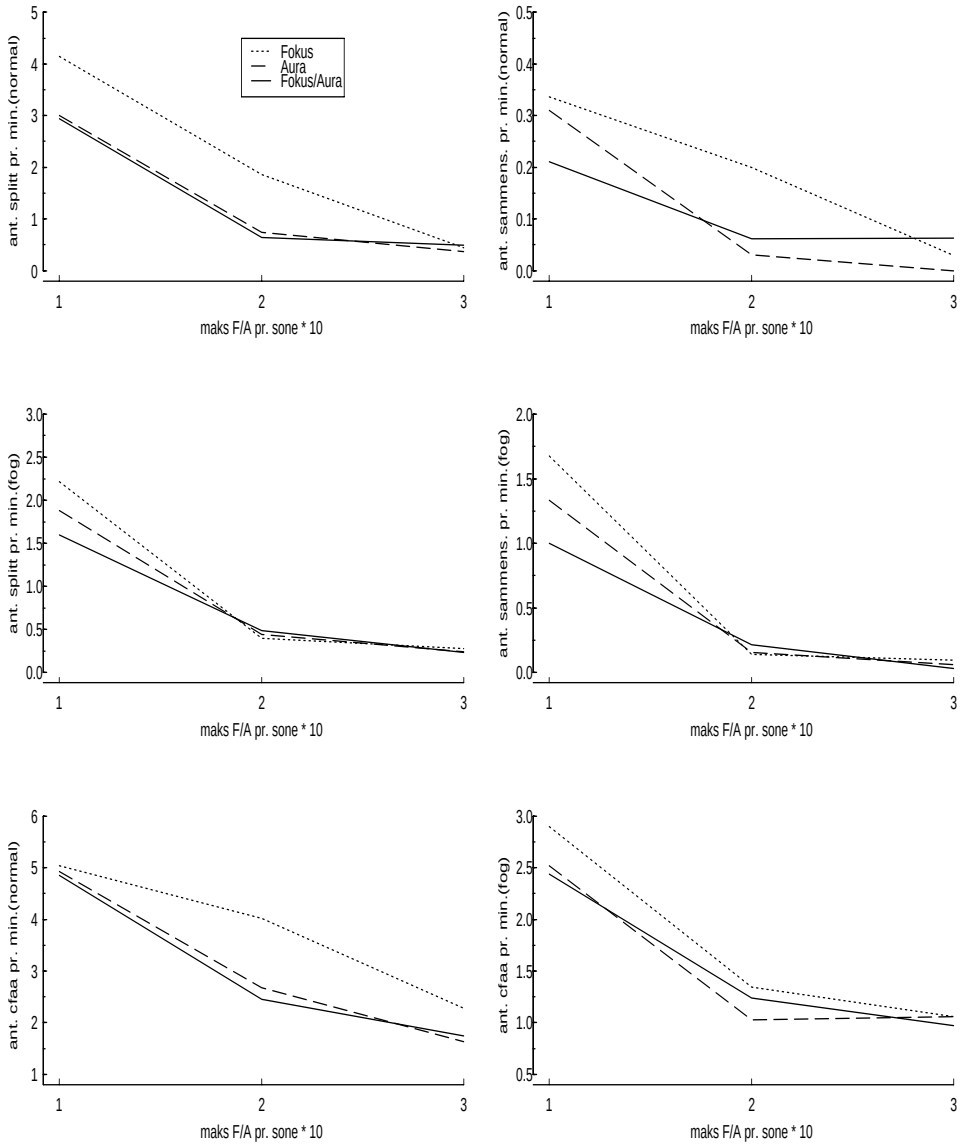
De to nederste grafene viser hvor ofte metoden `checkFocusAndAura` hos klienten resulterer i at det sendes en oppdateringspakke til serveren som forteller at klienten har oppdatert sine fokus- og aurasoner. Ut ifra disse resultatene kan vi også si at det strengeste kriteriet for splitting og sammenslåing (altså to terskler) kommer like godt ut som om vi betrakter kun bruk av en terskel (fokus eller aura).

Konklusjon

Simuleringsresultatene viste ikke forskjeller av betydning mellom de tre kriteriene for splitting og sammenslåing av soner. Allikevel kan vi si at det er mest hensiktsmessig å favorisere kriteriet hvor både en fokusterskel og en auraterskel betraktes. Dette er en påstand vi kan legge ut *på grunnlag av* resultatene og ikke *til tross for* resultatene; fordi de tre løsningene følger hverandre så tett skiller ingen av de andre to kriteriene seg ut som *bedre* alternativer *og* fordi vi (som tidligere nevnt) må tenke oss de situasjoner hvor en sone dekkes av kun mange klienters aura eller kun mange klienters fokus.

Ut ifra grafene i figur 5.3 ser vi at antall klientforbindelser øker ettersom grensene for splitting av soner øker. For fokus og aura som kriterium ligger snittet for potensielle på 28.3 ved 10 som grense, 32.2 ved 20 som grense og 35.7 ved 30 som grense (normale forhold), noe som ikke er en drastisk økning.

Ser vi på tallene for trafikken mellom server og klient (figur 5.5) ser vi at trafikken synker betraktelig mellom grensene 10 og 20, men minimalt

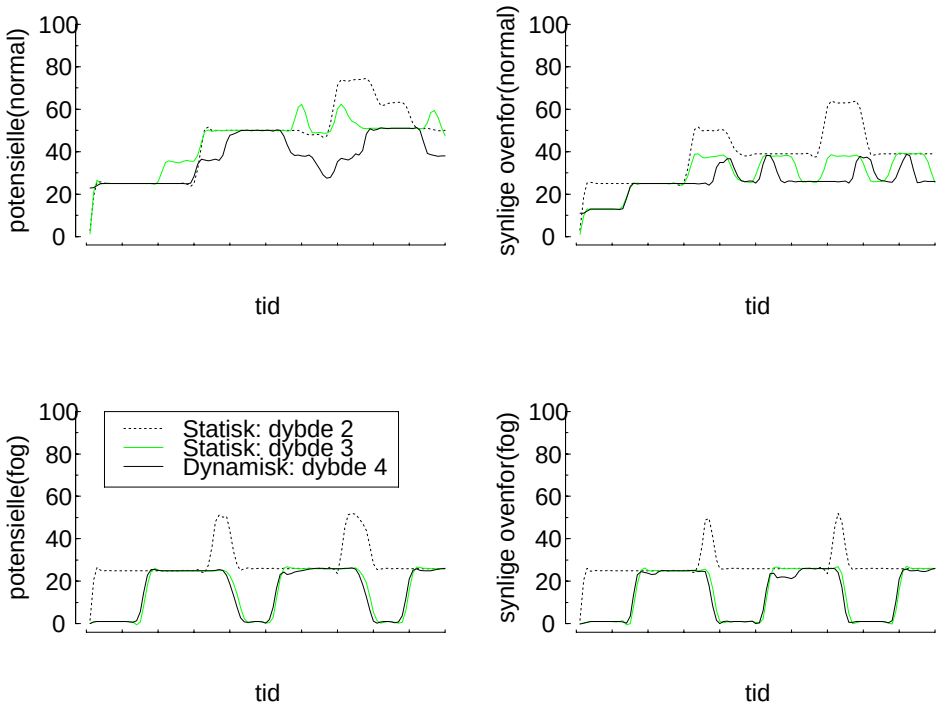


Figur 5.5: Trafikk server-klient. X-aksene representerer de tre forskjellige tersklene for splitting (10, 20 og 30) for de tre forskjellige kriteriene (fokus, aura og både fokus og aura). De fire øverste plotene viser hvor mange multicast-pakker alven mottok fra serveren (snitt pr. minutt) - de to øverste plotene representerer normale forhold mens de to plotene under viser resultatene for tåkeforhold. De to nederste plotene representerer i snitt pr. minutt hvor mange unicast-meldinger klienten sendte til serveren som resultat av at klientens sett av fokus- og/eller aurasoner måtte oppdateres. Plotet til venstre er under normale forhold og plotet til høyre under tåkeforhold (cffa = metoden checkFocusAndAura).

mellom grensene 20 og 30. På bakgrunn av dette kan vi slå fast at det for disse simuleringene er mest hensiktsmessig å benytte 20 som grensene til hvor mange fokus og hvor mange aura som kan assosieres med en sone før den splittes. Dette tallet kan kanskje settes i sammenheng med og bestemmes av hvor mange klienter simuleringen huser og/eller hvor stort område som simuleres. Dette er kun en påstand, men noe som potensielt kan testes (ved å kjøre simuleringer hvor antall klineter varierer og hvor utstrekningen av det simulerte området varierer).

5.2.2 Dynamiske soner vs. statiske soner

I dette avsnittet vil vi se på om den dynamiske soneinndelingen, hvor splitting og sammenslåing baserer seg på to terskler, har noen fordeler sammenlignet med en statisk soneinndeling.



Figur 5.6: Trafikk server-klient. X-aksene viser tidsforløpet for alvens diagonale bevegelse gjennom omgivelsene (gjennom klyngene). For plotene til venstre viser Y-aksene hvor mange klientforbindelser alven hadde til enhver tid (normale forhold og tåkeforhold). For plotene til høyre representerer verdiene på Y-aksen hvor mange klienter alven sendte oppdateringspakker til (hvor mange den var synlig overfor).

Figur 5.6 viser resultater av to simuleringer med forskjellige dybder for en

statisk soneinndeling og en testkjøring med dynamisk soneinndeling (under normale forhold og tåkeforhold):

- Statisk med dybde 2 - dvs. et rutenett på 4x4 soner.
- Statisk med dybde 3 - dvs. et rutenett på 8x8 soner.
- Dynamisk med dybde 4 - dvs. en oppdeling av soner på variabel størrelse hvor antall soner ligger mellom 1 (kun rotsonen) og 256 (16x16) soner.

Grunnen til det ikke ble testet med statisk soneinndeling med dybde 4 er at det tilsvarer "worst case scenario" for den dynamiske soneinndelingen (en maksimal oppsplitting). Selv om den dynamiske inndelingen her potensielt kan deles inn i 16x16 soner så er det viktig å bemerke at det er et mål at omgivelsene deles inn i færrest mulig soner. Jo flere soner desto større belastning på serveren - dette fordi klienter oftere vil måtte oppdatere sine fokus- og aurasoner. Samtidig vil for få soner resultere i at en gitt klient vil få mange forbindelser til andre klienter (mens trafikken til servren vil minke). Det må inngås et kompromiss hvor belastningen mellom server-klient og mellom klient-klienter til en størst mulig grad balanseres.

Grafene på figur 5.6 viser alvens forbindelser (potensielle) og hvor mange alven var synlig overfor (hvor mange den sendte oppdateringspakker til). Scenariet for simuleringene var den hvor det forekom konsentrasjoner av klienter på fire forskjellige steder (med 24 eller 25 klienter i hver klynge), og hvor klientene som la grunnlag for simuleringsresultatenen beveget seg diagonalt gjennom disse klyngene (se figur 5.7).

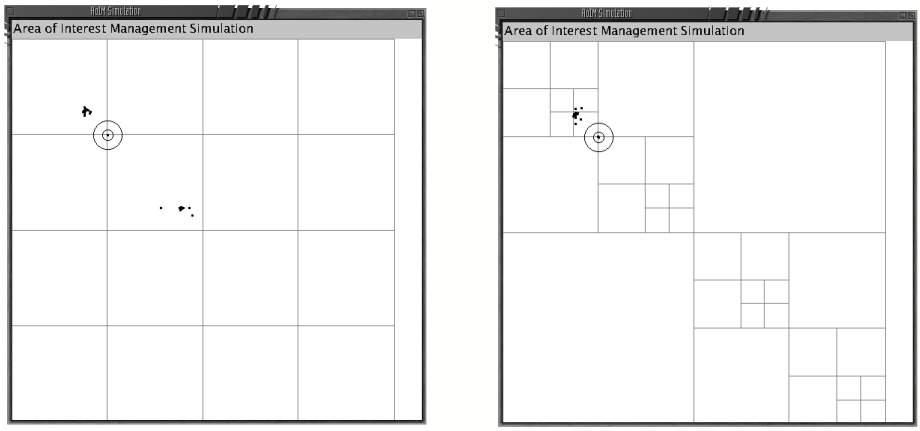
Klientforbindelser: statisk vs. dynamisk

I plottet øverst til venstre på figur 5.6 (ant. potensielle under normale forhold) faller de tre grafene sammen på to intervaller på x-aksen hvor y-verdien er rundt 50. Det er der hvor alven beveger seg *mellom* to klynger av klienter, og hvor alle klientene i begge klyngene har et felles interesseområde med alven (hver klynge består av 24 eller 25 klienter).

Det interessante her er å se hvordan antall potensielle synker mellom disse intervallene for klienten med dynamisk soneinndeling. Den når en bunn der hvor den befinner seg midt i en klynge, mens klientene med statisk soneinndeling ikke kommer under 50 potensielle klienter - fordi de ikke greier å "kvitte seg med" klientene fra forrige klynge når de passerer gjennom neste klynge - de når tvert imot sine maksimalverdier her hva klientforbindelser angår. Klientene med statisk soneinndeling har til tider forbindelse med klienter fra tre forskjellige klynger (den med dybde 2 har maksimalt 74 forbindelser, mens klienten med dybde 3 har en maksimalt forbindelse med 63 klienter),

mens klienten med dynamisk soneinndeling har maksimalt ca. 50 forbindelser (alle klientene fra to soner) og når en bunn på 27 forbindelser i intervallet hvor de to statiske når sine topper.

For scenariet med dynamisk inndeling oppretter m. a. o. klienten forbindelser til klientene i klyngene *senere* enn hva tilfellet er for den statiske inndelingen. Samtidig greier klienten for den dynamiske inndelingen å bryte forbindelsene raskere med klientene i en klynge idet den beveger seg bort fra den. Dette er fordi det oppstår en mer finkornet inndeling av sonene rundt hver klynge. Figur 5.7 viser det grafiske grensesnittet for en klient med statisk soneinndeling (til venstre) og grensesnittet for en klient med dynamisk soneinndeling (til høyre). De to klientene befinner seg på samme punkt i planet. Klienten med statisk soneinndeling har forbindelse med klienter fra to klynger mens klienten med dynamisk inndeling kun har forbindelser med klienter fra en klynge. Dette skjer på grunn av at den dynamiske inndelingen hindrer klienten å opprette forbindelser med den ene klyngen - en mer finkornet soneinndelig der hvor klyngene opptrer gjør at klienten ikke vil kunne opprette forbindelser med klienter i den andre klyngen før den har beveget seg nærmere klyngen.



Figur 5.7: Statisk soneinndeling til venstre og dynamisk soneinndeling til høyre. Begge viser viser grensesnittet til alven under tåkeforhold.

Antall man er synlig overfor: statisk vs. dynamisk

Plottet øverst til høyre på figur 5.6 viser at det ikke er så stor forskjell mellom statisk soneinndeling med dybde 3 og den dynamiske soneinndelingen mht. hvem alven er synlig overfor. Siden det er auraen som definerer hvem klientene er synlige overfor, og denne er liten sammenlignet med fokus for alven, greier begge løsningene å holde seg et sted mellom noe over 20 og 40 klienter (den statiske har et snitt på 30 klienter, mens den dynamiske har et snitt

på 26), men vi ser at den dynamiske inndelingen har kortere tidsintervaller for maksimalverdiene enn hva den statiske har. Igjen ser vi altså at en mer finkornet soneinndeling rundt klyngene gjør at tiden alven har forbindelser med alle klientene i klyngen kortes ned sammenlignet med den statiske.

Stor radius for fokus og aura

Betrakter vi de to nederste plottene på figur 5.6 ser vi derimot at statisk inndeling med dybde 3 og dynamisk inndeling følger hverandre meget jevnt. Ut ifra dette kan vi trekke den slutning at jo større fokus og aura en klient har, jo større gevinst er det å benytte seg av en dynamisk soneinndeling.

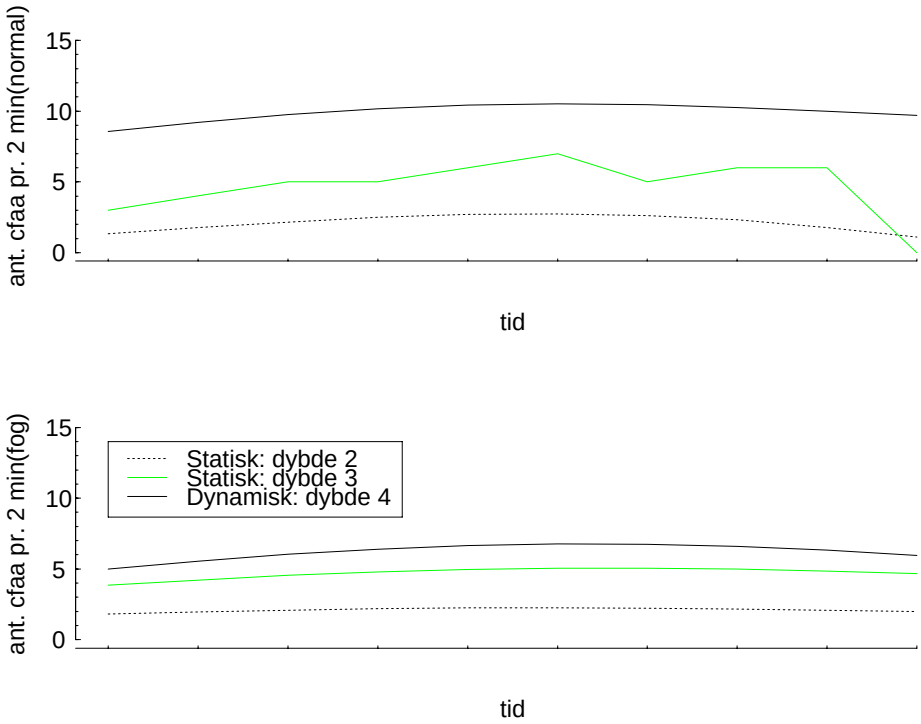
Trafikk mellom klient og server

Som nevnt kunne den dynamiske soneinndelingen i verste fall komme opp i 256 soner (et rutenett på 16x16). Jo flere soner en omgivelse deles inn i betyr at den gjennomsnittelige størrelsen for sonene minker, som igjen betyr en større frekvens av soneskifter for en klients fokus og aura. For simuleringene som er beskrevet i denne seksjonen hadde vi statiske soneinndeligner på 16 (dybde 2 - 4x4) og 64 (dybde 3 - 16x16). For den dynamiske inndelingen fikk vi for normale forhold en inndeling på 130 soner (i snitt), og for tåke forhold en inndeling på 34 soner (i snitt). Dette viser at man ved å benytte dynamisk soneinndeling minsker forbindelsene klientene seg imellom, men at man som et kompromiss til dette får en verden oppdelt i flere soner enn hva man får med en statisk. Men som vi ser får vi færre soner ved den dynamiske inndelingen enn den statiske med dybde 3 ved tåkeforhold. Så selv om de to løsningene fulgte hverandre tett her hva klientforbindelser angår, så fikk vi for den dynamiske en halvering i antall soner sammenlignet med den statiske.

Figur 5.8 på neste side viser at den dynamiske soneinndelingen trigget sending av flere meldinger som følge av metoden `checkFocusAndAura` (fra klient til server) sammenlignet med de to statiske løsningene, men at forskjellene er mindre for tåkescenariet enn for normalscenariet.

Konklusjon

Klyngene av klienter i scenariet for disse simuleringene var plassert relativt nære hverandre, noe som i utgangspunktet ikke favoriserte den dynamiske soneinndelingen hva de forventede resultatene angikk; fordi antall klienter, i relasjon til avstandsfiltreringen, ikke vil kunne holdes mer nede for den dynamiske enn for den statiske når man befinner seg midt i en klynge. Som nevnt ovenfor er den dynamsike inndelingen overlegen den statiske når man nærmer seg eller forlater en klynge, men slik simuleringen er satt opp vil hver klient som inngår i en klynge ha felles interesseområde med alle de andre klientene



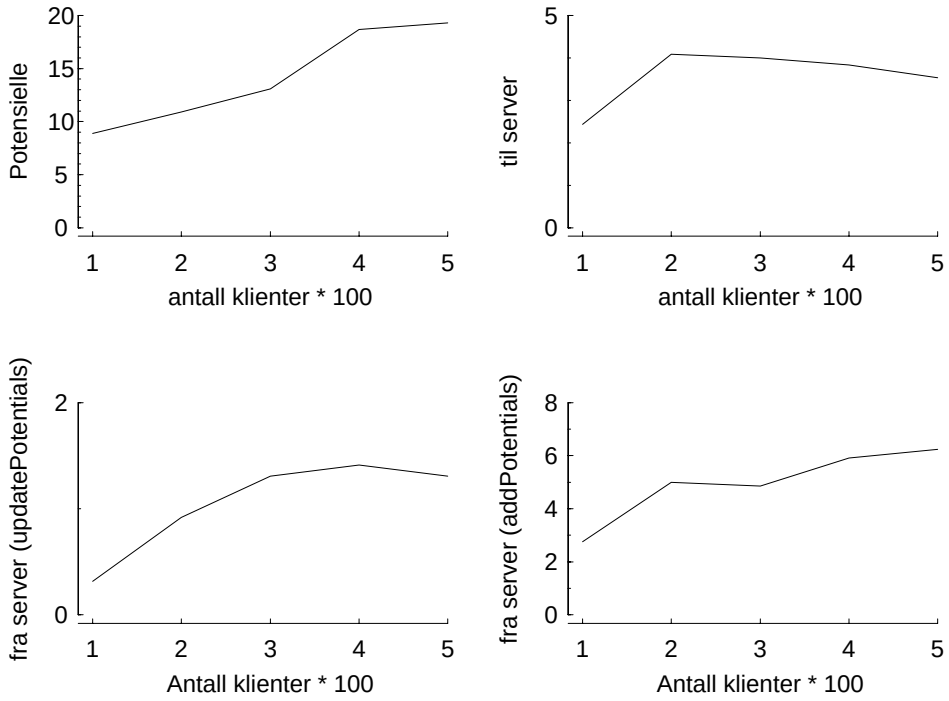
Figur 5.8: Trafikk klient-server. X-aksene viser tidsforløpet for alvens diagonale bevegelse gjennom omgivelsene (gjennom klyngene) Hvert punkt markerer 2 minutter i tid. Y-aksene viser summen av hvor mange unicast meldinger alven sendte til serveren for hver tidsenhet (2 min.). Øverste plot representerer normale forhold og nederste plot tåkeforhold.

i klyngen - til tross for den finkornete inndelingen av soner vil klienten dekke alle disse med sin aura og fokus. Dette forsvarer prinsippet om at radius for fokus og aura til klienter i en klynge vil minke når det oppstår en slik klynge. Fokus vil minske fordi en klient kun oppfatter klientene i en mindre omkrets, og aura minsker for en gitt klient fordi den kun er sansbar for klienter i umiddelbar nærhet. M. a. o. så ville man kunne holde klientforbindelsene betraktelig lenger nede for den dynamiske soneinndelingen hvis vi hadde hatt en dynamisk fokus og aura under simuleringen. Dette hadde ikke hjulpet for den statiske; når alle klientene i klyngen befinner seg innenfor samme sone vil det ikke hjelpe å minske fokus og aura aldri så mye.

5.2.3 Skalerbarhet

For å teste hvordan implementasjonen av relevansfiltreringen skalerer vil vi her se på resultater av simuleringer med 100, 200, 300, 400 og 500 klienter og hvor kriteriene for splitting og sammenslåing baserer seg på to terskler.

Figur 5.9 viser resultatene av simuleringene representert ved alven (tåkescenarium med tilfeldig bevegelsesmønster).



Figur 5.9: Trafikk server-klient ved forskjellig antall klienter. X-aksene representerer antall klienter for hver simulering. Øverst til høyre vises antall potensielle - hvor mange klientforbindelser alven hadde (snitt pr. minutt). Øverst til høyre ser vi antall unicastmeldinger fra klient til server (snitt pr. minutt). De to nederste grafene viser frekvensen av unicastmeldinger fra serveren til alven.)

Grafen øverst til venstre viser i snitt hvor mange potensielle alven hadde under de forskjellige simuleringene. Fra 100 til 500 klienter er det en økning fra 8.9 til 19.3 klientforbindelser, som er litt mer en det dobbelte (en økning på 118%).

Øverst til høyre ser vi hvor mange oppdateringspakker alven sendte i snitt pr. minutt til serveren angående oppdateringer for hvilke soner alven hadde i sine fokus- og auralister. Vi ser det er en klar økning fra 100 til 200 klienter og at grafen flater ut og synker litt fra 200 til 500. Dette betyr at topologien for sonetreet "stabiliseres" for verdiene mellom 200 og 500 - det er en tilnærmet maksimal oppsplitting (med dybde 4).

De to nederste grafene viser trafikken fra server til den enkelte klient (her alven). Til venstre ser vi resultatene av meldinger fra serveren som inneholder en liste over en sones auraklienter. Denne listen sendes når klienten har fått

en ny sone i fokus. Her har vi en klar økning fra 100 til 300 klienter, men deretter flater grafen ut. Fra 100 til 500 klienter er økningen av denne typen oppdateringspakker tredoblet.

Nederst til høyre viser grafen frekvensen av meldinger fra serveren til alven om at en ny klient har entret en sone med sin aura som alven har i fokus. Her ser vi en svak og relativ jevn økning fra 100 til 500 klienter (fra et snitt på 2.75 til et snitt på 6.23 meldinger pr. min - en økning på 127%).

Disse resultatene viser at trafikken mellom en gitt klient og serveren ikke øker drastisk når antall klienter øker fra 100 til 500.

KAPITTEL 6

Avslutning

6.1 Konklusjon

Denne oppgaven har presentert et forslag til en relevansfiltrering hvis hensikt er å redusere nettverkstrafikken for en virtuell omgivelse. Det er i hovedsak avstandsfiltreringen som er blitt behandlet, hvor hovedideen er å skulle minimalisere antall forbindelser en gitt klient til enhver tid har til andre klienter. Valg av en hybrid peer-to-peer-arkitektur ble valgt fordi serveren i en klient-server-arkitektur raskt setter grenser for hvor mange klienter som kan koble seg til en simulering.

Prinsippet med å benytte en kombinasjon av *dynamisk soneinndeling* og *dynamisk fokus og aura* for å kunne definere felles interessesoner mellom klienter ble testet ut gjennom simuleringer. Resultatene tilsa at det er hensiktsmessig å benytte en dynamisk soneinndeling fremfor en statisk, og at løsningen skalerer bra (den ble testet for opptil 500 klienter).

Vi har i denne oppgaven adressert nytteverdien for en variant av relevansfiltrering. Noe denne oppgaven ikke behandler er problemstillingen rundt de tilgjengelige ressursene en endeterminal har. Prosessering av grafikk, lyd, input fra bruker, oppdateringsmeldinger fra andre klienter og annet relatert til spillets logikk vil typisk kreve mye ressurser. En AoIM-modul er bare en del av et spill og det er viktig at ressursene denne benytter ikke "sulter ut" de andre prosessene.

Vi kan dermed si at det absolutt er nødvendig med metoder for å holde nettverkstrafikken nede for en virtuell omgivelse, og at dette realiseres ved et samspill mellom metoder for relevansfiltrering og *dead reckoning*. Det må så inngås et kompromiss mellom disse teknikker og de ressurser de krever å prosessere hos endebrukerne.

6.2 Fremtidig arbeide

Slik vi her har beskrevet begrepet relevansfiltrering er avstandsfiltreringen den initielle delen. Denne oppgaven begrenser seg til å beskrive avstandsfiltreringen og interessefiltreringen hvor hovedvekten ligger på avstandsfiltreringen - hvor kun denne har blitt implementert og testet via simuleringer. For videre arbeide med de ideer som har blitt presentert i denne oppgaven vil det så være naturlig å først implementere interessefiltreringen.

6.2.1 Implementasjon av interessefiltreringen

Prinsippene for interessefiltrering som er presentert i denne oppgaven bør også implementeres for å verifisere nytteverdien. Rent intuitivt kan vi påstå at teknikkene vi har sett på i kapittel 3 vil tjene sin hensikt med å redusere trafikken mellom en gitt klient og dens sett av potensielle klineter. For å avgjøre om en implementasjon som baserer seg på den foreslåtte styringen av interesse (manuell og automatisk) lar seg benytte som en gradering av de potensielle klientene, må den testes opp mot en virtuell omgivelse som har et grafisk grensesnitt i samsvar med en applikasjon som tilsvarer hva den er tiltenkt å bli benyttet sammen med (altså en representativ virtuell omgivelse brukeren kan interagere med jfr. definisjonen i avsnitt 1.2). Den manuelle styringen av interesse må vise seg å være brukervennlig før den kan betraktes som et alternativ for interessefiltrering. Dette kan kun oppnås ved testing opp mot en faktisk virtuell omgivelse.

6.2.2 Dynamisk fokus og aura under kjøring

Som nevnt i kapittel 5 vil det ideelt sett være mest gunstig å skulle teste en AoIM-modul opp mot en 3-dimensjonal verden. I en slik verden vil man kunne simulere faktiske atmosfæriske forandringer som tåke, regn og forskjellige grader av lysstyrke. Det kan dermed bli gjort simueringer hvor fokus og aura har en dynamisk natur under en enkelt simulering. Da vil man kunne få bekreftet om det er hensiktsmessig å forandre radius for en klients fokus og aura basert på atmosfæriske forhold, og man ville kunne tallfeste hvor stor gevinst dette har uten at det forringer den visuelle opplevelsen en bruker har av en virtuell omgivelse.

I annen instans vil man så kunne teste prinsippene med å regulere aura ut ifra omgivelsene - konkurrerende stimuli og støy. Her kan man ved grundig testing prøve å finne verdier eller spesielle scenarier som kan påvirke (minske) en klients aura. Hvis omgivelsene har en dominerende farge (eller består av farger innenfor et meget begrenset fargespekter) kan man kanskje finne en grense for når den er å betrakte som konkurrerende overfor avatarer og objekter som befinner seg i omgivelsene. Hvis man har et område som er befolket av flere avatarer kan man definere dominerenede farger, former,

raser, lengder o. l for så å kunne se hvor mye hver enkelt avatar skiller seg ut (jo mer man skiller seg ut, jo større aura).

6.2.3 Synsfelt og hovedfokus

I denne oppgaven defineres fokus og aura ved sirkler - en klients interesseområde defineres ved å bestemme radius for fokus og aura og hvor klienten befinner seg i midtpunktet. Dette vil si at en klient mottar like mye informasjon om hendelser utenfor synsfeltet som for hendelser i synsfeltet.

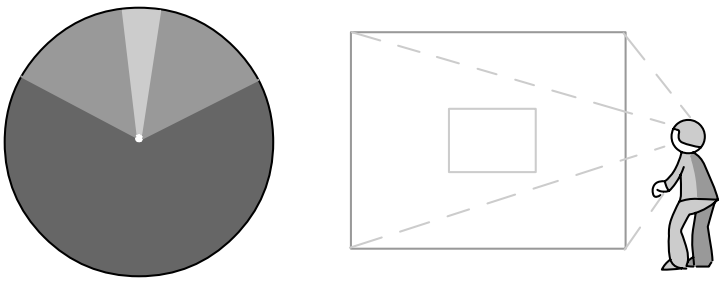
Alternativt til den sirkulære fokus hvor alt innenfor sirkelen er med på å definere interesseområdet, kan man så tenke seg en fokus som retter seg etter en klients *synsfelt*. Et problem blir da at settet av fokus-soner må oppdateres ofte hvis klientens synsfelt roteres. Løsningen er kanskje å fortsatt tenke fokus som en sirkel/kule, men at alt som faller utenfor synsfeltet blir gradert til den laveste interessegraden. Synsfeltet er ekvivalent med hva en bruker ser på skjermen[1]. Dermed vil en klient ha forbindelser til alle klienter den har felles interesseområde med (som defineres av den sirkulære fokus og aura), men hvor synsfeltet er med på å gradere klientene. Hvis en klient f. eks. snur 180 grader rundt vil ikke dette påvirke interesseområdet til klienten, men graderingen av de potensielle klientene vil måtte forandres. For synsfeltet vil man så også kunne definere en *hovedfokus* som vil være en sub-del av synsfeltet hvor man har fokus (som typisk vil være et felt midt på skjermen). Klienter og objekter som faller innenfor denne hovedfokusen graderes initielt som mer interessante enn klienter utenfor.

Figur 6.1 på neste side viser eksempler på hvordan fokus kombineres med synsfelt for både en 2-dimensjonal og en 3-dimensjonal verden. Til venstre vises synsfeltet i grått, hovedfokus i lysegrått mens den mørkegrå delen representerer området hvis klienter befinner seg blir gradert til den laveste interessegraden (visuelt). Til høyre er interesseområdet tenkt som det en bruker ser på skjermen (stor firkant). Den lille firkanten viser hovedfokus.

6.2.4 Subkategorier for fokus og aura

Som tidligere nevnt vil en relevansfiltrering ikke kunne testes fullt ut før den benyttes sammen med en faktisk virtuell omgivelse (som f. eks. spill på linje med *Anarchy Online* og *EverQuest*). Dagens spill og simuleringer har en visuell del og en audio-del. Det vil da også være naturlig å dele inn fokus og aura i disse subkategoriene. Det er den visuelle fokus og aura som har blitt omtalt i denne oppgaven, men en fokus og aura for audio bør kunne defineres og implementeres ved å benytte de samme prinsipper. Størrelsene til visuell fokus og aura vil da kunne variere fra størrelsen til den audiobaserte fokus og aura. Skiller man mellom oppdateringspakker for lyd og bilde vil man så

[1] Dette bestemmes igjen av hva slags "linse" som benyttes for å frembringe omgivelsene visuelt. En vidvinklet linse vil typisk gi et større synsfelt.

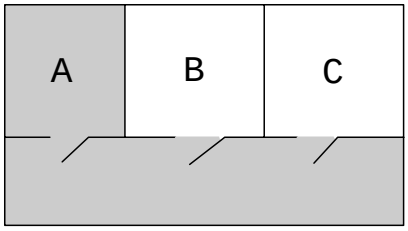


Figur 6.1: Fokus med synsfelt og hovedfokus.

kunne benytte to sett av interessegrader. Slik vil en klient f. eks, kunne gis en høyere interessegrad for audio enn for visuelt.

6.2.5 Dynamisk soneinndeling kombinert med *LOCALES*

Meningen med en sirkulær fokus og aura er opplagt når man befinner seg på store og åpne områder (utendørs). Hvis man derimot befinner seg i en omgivelse som består hovedsakelig av bygninger hvor disse igjen er inndelt i mindre enheter (som rom og etasjer), vil det så være mer hensiktsmessig å definere interesseområdet ut ifra de klare grenser som disse enheter har. Dette er prinsippet for $SPLINE$ (se 2.1.3 på side 17) hvor da interessesonen bestemmes av enheten klienten befinner seg i samt enheter som ligger inntil denne - hva som i SPLINE kalles $LOCALES$. Et eksempel er en klient befinner seg i et rom som har en åpen dør ut til en gang. Interessesonen vil da være rommet og gangen (se figur 6.2).



Figur 6.2: Fokus med synsfelt og hovedfokus. En klient som befinner seg i rom A vil som interessefelt ha rommet og gangen som rommet har en dør til.

En virtuell omgivelse vil typisk bestå av både bygninger og utendørslandskap. En mulighet er da å implementere en AoIM-modul som kombinerer den dynamiske soneinndelingen med en enhetsbasert, hvor man så bytter mellom disse to alt ettersom klienten befinner seg i en bygning eller utendørs.

Bibliografi

[1] K. Y. Oldfield: *Issues in a Very Large Scale Distributed Virtual Environment*, Department of Computer Science and Engineering, The Chinese University of Hong Kong (1997)

[2] R. Wodaski: *Virtual Reality Madness! 1996*, Sams Publishing (1995)

[3] T. L. Disz, M. E. Papka, M. P. Pellegrino, R. Stevens: *Sharing Visualization Experiences among Remote Virtual Environments*, Mathematics and Computer Ascience Division, Argonne National Laboratory, Il, USA (1995)

[4] S. Singhal, M. Zyda: *Networked Virtual Environments*, Addison-Wesley Pub Co (1999)

[5] M. Macedonia, M. Zyda, D. Pratt, P. Barham, S. Zeswitz: *NPSNET: A Network Software Architecture for Large Scale Virtual Environments*, Naval Postgraduate School, Dep. of Computer Science, Monterey (1994)

[6] L. E. Pfeffer, D. T. Latimer: *Toward Open Network Data-Exchange Protocols For Construction Metrology and Automation: Liveview*, Construction Automation and Metrology Group, NIST (1999)

[7] S. Brand: *Spacewar*, Rolling Stone Magazine 7. desember (1977)

[8] J. Mulligan: *History of Online Games*, Imaginary Realities (2000)

[9] F. Haugen: *Anarchy Online inspirerte til ny Warcraft*, VG, 3. august (2001)

[10] M. Capps, D. McGregor, D. Brutzman, M. Zyda: *NPSNET-V, A New Beginning for Dynamically Extensible Virtual Environments*, Naval Postgraduate School, Monterey, California (2000)

[11] M. Fuchs, C. Diot, T. Turletti, M. Hofmann: *A Framework for Reliable Multicast in the Internet*, Technical Report RR-3363, INRIA, Sophia Antipolis, France (1998)

[12] R. L. Solso: *Cognitive Psychology*, Sixth Edition, University of Nevada, Reno (2001)

[13] H. A. Abrams: *Extensible Interest Management for Scalable Persistent Distributed VE*, Naval Postgraduate School, Monterey, California (1999)

[14] S. Fiedler, M. Wallner, M. Weber: *A Communication Architecture for Massive Multiplayer Games*, Dept. of Multimedia Computing, University of Ulm, Germany (2002)

[15] M. Macedonia, M. Zyda: *A Taxonomy for Networked Virtual Environments*, Naval Postgraduate School, Monterey, California (1995)

[16] L. H. Sahasrabuddhe ,B. Mukherjee: *Multicast Routing Algorithms and Protocols: A Tutorial*, University of California (2000)

[17] J. W. Barrus, R.C. Waters, D.B. Anderson: *Locales and Beacons: Efficient and Precise Support For Large Multi-User Virtual Environments*, Mitsubishi Electric Research Laboratory (1996)

[18] T. Barron, A. LaMothe: *Multiplayer Game Programming*, Premier Press, Inc. (2001)

[19] S. Benford, L. Fahlén: *A Spatial Model of Interaction in Large Virtual Environments*, ECSCW'93, Milan, September (1993)

[20] B. Ng, A. Si, R. W. H. Lau, F. W. B. Li: *A Multi-Server Architecture for Distributed Virtual Walkthrough*, ACM VRST'02, Hong Kong, November 11-13 (2002)

[21] W. Fenner: *Internet Group Management Protocol, Version 2*, RFC 2236, Xerox PARC (1997)

[22] M. Pritchard: *How to Hurt the Hackers: The Scoop on Internet Cheating and How You Can Combat It*, Gamasutra (2000)

[23] J. S. d'Attenhoven: *Management of Networked Virtual Environments*, University of Brussels, Polytechnics Faculty (2002)

[24] C. Diot, L. Gautier: *A Distributed Architecture for Multiplayer Interactive Applications on the Internet*, SPRINT ATL, USA (1999)