Topias Vesalainen

Server-client verkkomoninpelitekniikat

Opinnäytetyö
Kajaanin ammattikorkeakoulu
Pelialan
Tietojenkäsittelyn koulutusohjelma
5.11.2012

KAJAANIN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ
TIIVISTELMÄ

Koulutusala Luonnontieteiden ala	Koulutusohjelma Tietojenkäsittelyn koulutusohjelma
Tekijä(t) Topias Vesalainen	
Työn nimi Server-client verkkomoninpelitekniikat	
Vaihtoehtoiset ammattiopinnot	Ohjaaja(t) Matti Härkönen
	Toimeksiantaja -
Aika Syksy 2012	Sivumäärä ja liitteet 31

Opinnäytetyön tarkoituksena on tutkia ja selvittää, millaisia moninpelitekniikoita nykypäivän moninpeleissä käytetään, sekä palvelimella että pelaajalla. Aihe on ajankohtainen, koska nykyisin yhä useammassa pelissä on mukana jonkinlainen moninpeli perinteisen yksinpelin rinnalla. Osa peleistä jopa keskittyy ainoastaan moninpeliin, eikä niissä ole yksinpelimahdollisuutta.

Aluksi työssä tehdään katsaus moninpelien historiaan ja missä tilanteessa nykypäivänä ollaan, sekä verrataan, mitä eroa TCP- ja UDP-verkkoprotokollilla on moninpelien kannalta. Tämä jälkeen esitellään yleisesti käytettyjä moninpelitekniikoita sekä palvelimen ja pelaajan näkökulmasta. Suurin osa esitellyistä tekniikoista koskee FPS pelejä, mutta samoja tekniikoita voidaan soveltaa monissa muissakin pelityypeissä. Lopuksi käydään läpi toteutettua ohjelmointiprojektia, jossa esiteltyjä tekniikoita on toteutettu Unity3D-pelimoottorissa käyttäen lidgren network gen3 -verkkokirjastoa.

Ohjelmointityön tuloksena on toimiva prototyyppi palvelimen ja pelaaja ohjelmistosta, jota voidaan tulevaisuudessa jatkokehittää. Toteutettu prototyyppi ei ole täysin onnistunut ohjelmistoarkkitehtuurin osalta, vaikkakin kaikki siinä käytetyt moninpelitekniikat toimivat kuten pitääkin. Prototyypin ongelmaksi koitui sen toiminta suuremmilla pelaajamäärillä, jolloin siinä ilmeni erilaisia toiminnallisia virheitä. Kokonaisuutena opinnäytetyöprosessi on kuitenkin onnistunut, koska sen aikana tekijä on henkilökohtaisesti oppinut paljon uutta tietotaitoa käsitellystä aiheesta.

Kieli	Suomi
Asiasanat	Verkkopeli, Moninpeli, Palvelin, Asiakas Ohjelmointi
Säilytyspaikka	☒ Verkkokirjasto Theseus ☒ Kajaanin ammattikorkeakoulun kirjasto

KAJAANIN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

THESIS
ABSTRACT

School	Degree Programme
Kajaani University of Applied Sciences	Business Information Technology
Author(s) Topias Vesalainen	
Title Server-client Multiplayer Techniques	
Optional Professional Studies	Instructor(s) Matti Härkönen
	Commissioned by -
Date Fall 2012	Total Number of Pages and Appendices 31

The purpose of this thesis is to study and investigate what kind of multiplayer techniques are used in today's multiplayer games on server and client. The topic is currently important because nowadays more and more games have some kind of multiplayer mode with the traditional single player experience. Some games even focus entirely on the multiplayer option, without the proper single player mode.

At first, there will be a brief look on the history of multiplayer games, where they are now and what differences TCP and UDP network protocols have. After this comes an introduction to the mostly used multiplayer techniques from the server and client perspective. Most of the introduced techniques deal with first person shooter games, but the same techniques can also be used on other games types. Lastly, the thesis includes the introduction of the actual prototype that was constructed during the project. The prototype uses the techniques mentioned in this thesis on Unity3D game engine utilizing the lidgren networking gen3 networking library.

The result of the programming work is a working prototype of server-client software that can be further developed in the future. The completed prototype is not entirely successful on the architecture side, even though all the used multiplayer techniques work as they should. Large simultaneous user counts accrued to be problematic because they seemed to cause different operational errors on the prototype. As a whole the thesis project was successful because the author has personally learned lots of new know-how on the subject.

Language of Thesis	Finnish
Keywords	Server, Client, Multiplayer, Programming
Deposited at	☒ Electronic library Theseus ☒ Library of Kajaani University of Applied Sciences

ALKUSANAT

Aihe tähän opinnäytetyöhön moninpeliohjelmointi, syntyi opintoihin liittyvän työharjoittelun aikana. Työharjoittelussa asian tutkimisen jälkeen mielenkiinto heräsi ja samalla selvitin, voisiko tätä aihetta tutkia enemmän tulevassa opinnäytetyössä. Mielenkiinto heräsi erityisesti pelimekaniikan prosesseihin, joilla moninpeli saadaan näyttämään pelaajille yhtäaikaiselta, vaikka todellisuudessa pelissä on tiedonsiirron viive olemassa.

Opinnäytetyön projektin toteutuksessa tavoitteena on saada käytännön kokemusta mekaniikkojen toteutuksesta sekä palvelimella että asiakkaalla. Projektin aikana opitut tiedot ja kokemukset ovat tärkeitä pelienkehityksessä, sillä nykypäivänä useissa peleissä on jonkinlainen moninpelimuoto perinteisen yksinpelin lisäksi. Opinnäytetyöllä ei ole toimeksiantajaa, vaan tavoite on myöhemmin käyttää toteutettua tulosta omissa projekteissa sekä jatkokehittää ohjelmaa edelleen. Jatkokehittäminen ja -muokkaaminen ovat tulevaisuudessa helpompia kun tuotetta ei ole sidottu yrityksiin.

SISÄLLYS

SYMBOLILUETTELO

Client	Englanninkielinen termi asiakkaalle, verkkopelien yhteydessä sanalla tarkoitetaan pelaajaa, joka on liittynyt peliä pyörittävään palvelimeen.
Ekstrapolointi	Matemaattinen ratkaisutapa, jolla pyritään etsimään tuntemattoman funktion arvoja, jotka sijaitsevat tiedossa olevien pisteiden ulkopuolella.
Hitscan	Peleissä käytetty tapa mallintaa aseella ammuttujen luontien liikettä. Hitscan-mekaniikassa aseen luotien liikerata mallinnetaan toteuttamalla Raycast aseesta suuntaan, johon luoti ammutaan.
Interpolointi	Matemaattinen ratkaisutapa, jolla pyritään etsimään tuntemattoman funktion arvoja, jotka sijaitsevat tiedossa olevien pisteiden sisäpuolella.
Peer-to-peer	Englanninkielinen termi vertaisverkolle, joka on verkko, jossa ei ole erillistä palvelinta ja asiakasta.
Pelaajan tila	Kokoelma tietoja, jotka kuvaavat pelaaja hahmon tai objektin tilaa pelimaailmassa. Esimerkiksi pelaajan paikka tai pelaajan energiatilanne.
Projektiili	Ammuksen yleisnimitys, tämä käsittää kaikenlaiset ampuma-aseilla ammuttava luodit ja heittokappaleet.
Raycast	Tietokonegrafiikassa ja -peleissä käytetty menetelmä, jolla voidaan ratkoa erilaisia ongelmia. Raycast-menetelmässä 3D-maailmaan luodaan suora viiva, jonka avulla voidaan esimerkiksi todeta, näkeekö kamera tietyn objektin.
Replikointi	Tarkoitetaan pelissä olevan objektin tai tapahtuman tilan samaistamista palvelimen ja pelaajan välillä.
Server	Englanninkielinen termi, jolla tarkoitetaan palvelinta. Palvelin on keskitetty tietokone, joka välittää erilaisia palveluita. Moninpeleissä

	palvelin välittää pelimaailman tietoja siihen yhteydessä oleville asiakkaille.
Synkronointi	Menetelmä, jolla pyritään pitämään kaksi tai useampi objektia mahdollisimman lähellä palvelimella olevan objektin tilaa.
Syöte	Tarkoitetaan asiakkaan antamia komentoja tietokoneelle.
Verkko-objekti	Tarkoitetaan moninpelimaailmassa olevaa objektia, joka on synkronisoitu muille asiakkaille.

1 JOHDANTO

Opinnäytetyön aiheena on selvittää ja tutkia verkkopelien server-client arkkitehtuurin toimintaa ja toteuttaa server-client ympäristön prototyyppi. Aihe on tärkeä pelinkehityksen kannalta, koska entistä useammassa pelissä on mukana jonkinlainen verkkopeli perinteisen yksinpelin rinnalla ja jotkin pelit keskittyvät yksinomaan verkkopeliin. Tavoitteena on tutkia ja selvittää, minkälaisia tekniikoita server-client moninpeli ympäristöissä käytetään, jotta lähetettävän datan määrä saadaan pidettyä mahdollisimman pienenä ja illuusio samanaikaisesta pelaamisesta saadaan säilymään.

Tavoitteena ei ole tutkia kaikkea, mikä liittyy verkkopelien toteuttamiseen, vaan työ keskittyy palvelin- ja asiakasohjelmien toiminnallisuuteen verkkopelin kannalta, esimerkiksi miten tieto välitetään palvelimelta pelaajalle, miten lähetettävän tiedon määrä saadaan minimoitua ja miten illuusio samanaikaisesta pelaamisesta samassa pelimaailmassa saadaan aikaan. Työssä ei käsitellä verkkopistokkeiden toimintaa tai sitä, miten varsinainen tieto liikkuu internetissä, kun se on lähetetty palvelimelle tai pelaajalle. Esittelen kuitenkin, mitä eroa TCP- ja UDP-verkkoprotokollilla on verkkopelin kannalta, sillä näiden verkkoprotokollien erot on hyvä tietää, vaikkei kehittäjä suoranaisesti olekaan tekemisissä näiden protokollien kanssa.

Projektiosuudessa tarkoituksena on toteuttaa server-client ympäristö Unity3D-pelimoottorille. Toteutetun prototyyppiympäristön päälle rakennetaan yksinkertainen FPS-pelidemo (First Person Shooter), jotta prototyypin toimivuutta voidaan testata käytännössä. Tavoitteena on toteuttaa toimiva verkkopeli prototyyppi, jota voidaan jatkokehittää muissakin projekteissa. Prototyypissä verkkoliikenteen välittämiseen käytetään avoimen lähdekoodin Lidgren networking generation 3 -kirjastoa, joka tarjoaa valmiin ratkaisun tiedonvälittämiseen palvelimen ja pelaajan välillä.

2 VERKKOMONINPELIN TAUSTAT JA TAVOITTEET

Moninpelit ovat virtuaalimaailman jakamista. Toisin sanoen kaikki pelin pelaajat näkevät saman maailman ja samat tapahtumat omasta näkökulmastaan. Alkuaan moninpelit olivat pieniä, muutaman pelaajan pelejä, kuten aikoinaan ID Softwaren Doom-pelissä (1993). Moninpelit ovat muuttuneet pienistä muutaman pelaajan peleistä suuriksi ja vapaammiksi, kuten Ultima Online (1997), World Of Warcraft (2004) tai Battlefield-pelit (2002 – 2012). Samalla moninpeliteknologiat ovat kehittyneet valtavasti. (Nalezynski & Polge & Sweeney, 2010)

2.1 Peer-to-peer eli vertaisverkkomalli

Alkuaan, kun ensimmäiset moninpelit saapuivat markkinoille, ne käyttivät peer-to-peer (vertaisverkko) mallia moninpelin toteuttamiseen. Tässä mallissa kaikilla pelaajilla oli samat oikeudet pelimaailman muokkaamisessa. Jokainen pelaaja synkronoi omat näppäinpainallukset ja muut toiminnallisuudet muiden pelaajien kanssa. Kaikki synkronoidut tiedot suoritettiin täsmälleen saman pelilogiikan läpi, samoilla arvoilla ja samalla päivitysnopeudella. (Nalezynski & Polge & Sweeney, 2010)

Mallin vahvuus on sen yksinkertaisuus ja helppo toteutus. Mutta sillä on myös ongelmia, joiden takia sitä käytetään nykyään todella harvoin. Esimerkiksi kaikkien pelaajien on aloitettava peli samaan aikaan, eikä peliin voi jälkeenpäin liittyä. Toinen mallin ongelma piilee pelaajamäärissä. Koska kaikki pelaajat päivittävät maailmaa samaan aikaan, samanaikaisesti välitettävän tiedon määrä kasvaa pelaajamäärän mukana. Tällöin mahdolliset verkkohäiriöt tiedon siirrossa kasvavat. Kolmas ongelma on ruudunpäivityksen nopeuksissa. Myös tämä ongelma johtuu siitä, että pelaajat päivittävät maailman samaan tahtiin. Koska ruudun päivitysnopeus on aina sama kaikilla pelaajilla, on hankala toteuttaa peliä, joka toimisi kaikilla tietokoneilla. (Nalezynski & Polge & Sweeney, 2010)

2.2 Server-client -malli

Peer-to-peer mallin jälkeen seuraava suurempi muutos tapahtui ID Softwaren Quake-pelin (1996) myötä ja myöhemmin, kun Ultima Online (1997) julkaistiin. Tuolloin uudessa server-client -mallissa yksi tietokone määriteltiin palvelimeksi (server), joka on vastuussa kaikista pelissä tapahtuvista päätöksistä. Tässä mallissa toiset tietokoneet olivat asiakkaita (client), jotka eivät itse suorittaneet pelilogiikkaa. Nämä asiakastietokoneet vain piirsivät pelimaailman palvelimen antamien tietojen mukaan ja välittivät käyttäjien näppäinpainallukset palvelimelle. Nämä muutokset mahdollistivat verkkopelaamisen kasvun, kun palvelimia eri peleihin alettiin ylläpitää ympäri maailman. (Nalezynski & Polge & Sweeney, 2010)

Server-Client -malli kehittyi edelleen, kun QuakeWorld (1996) ja Quake 2 (1997) julkaistiin. Näiden pelien mukana server-client -malli muuttui siten, että osa pelilogiikasta ja simulaatioista siirrettiin myös asiakkaalle. Näiden muutoksien tarkoituksena oli vähentää lähetettävän tiedon määrää ja samalla nostaa pelien visuaalista laatua. Näissä peleissä asiakkaalle vielä lähetettiin tieto objekteista, jotka sen tulisi piirtää, mutta myös mahdollisten lentävien objektien liikeradan tieto. Tämän tiedon avulla asiakas pystyi tekemään alkeellisen arvion objektin liikkeestä. (Nalezynski & Polge & Sweeney, 2010)

2.3 Yleistetty server-client -malli

Nykyään server-client -mallin toteutukset ovat pikemminkin evoluutiota ensimmäisistä server-client -mallin toteutuksista. Yleistetyssä server-client -mallissa palvelimella on edelleen viimeinen sana kaikessa, mitä pelimaailmassa tapahtuu, joissain peleissä asiakkailla voi olla jotain pientä valtaa pelimaailman tapahtumissa. Nykyisin myös asiakas ylläpitää tietoja pelitilasta ja objekteista. Näiden tietojen avulla asiakas voi ennustaa tulevia tapahtumia ja täten vähentää lähetettävän tiedon määrää. Näiden muutoksien lisäksi palvelimen ja asiakkaan päähän on tullut paljon erilaisia ominaisuuksia, joilla pyritään parantamaan illuusiota yhtenäisestä pelimaailmasta. Esimerkiksi pelaajan liikkumisessa ei enää odoteta palvelimen päätöstä siitä, voiko asiakas liikkua. Sillä asiakkaan liike vain varmistetaan palvelimella, ja jos liikkeen paikat eroavat toisistaan, palvelin kertoo asiakkaalle oikean tiedon tämän liikkumisesta. Muita uusia ominaisuuksia ovat muun muassa viivehyvitys, jolla pyritään hyvittämään verkkoviivet-

tä, kun tietoa lähetetään palvelimen ja asiakkaan välillä. (Nalezynski & Polge & Sweeney, 2010)

3 VERKKOPROTOKOLLAT MONINPELEISSÄ

Internetissä käytetään kahta eri tiedonsiirtoprotokollaa TCP (Transmission Control Protocol) ja UDP (User Datagram Protocol). Pelin suunnitteluvaiheessa on jo hyvä miettiä, kumpi protokollista sopii kyseiseen pelin tarpeisiin paremmin. Yleisesti ottaen TCP on luotettava tapa lähettää tietoa, koska hävinneet paketit lähetetään uudestaan. UDP-protokollaa käytettäessä ei ole minkäänlaista takuuta siitä, että paketit saapuisivat perille. Jotkin pelit käyttävät sekä UDP- että TCP-protokollaa rinnakkain. Esimerkiksi pelaajien keskustelut voidaan lähettää TCP-protokollaa käyttäen ja muut tiedot UDP-protokollalla. (Spurling, 2004)

3.1 TCP-protokolla

TCP on yleisesti käytössä oleva verkkoprotokolla tiedonsiirrossa: esimerkiksi kaikki internetistä löytyvät verkkosivut, sähköpostit ja IRC (Internet Relay Chat) keskustelukanavat käyttävät tätä siirtoprotokollaa. TCP-protokollassa kahden tietokoneen välillä muodostetaan yhteys, jonka kautta tietoa välitetään. Tämä yhteys on luotettava ja järjestyksellinen. Eli kaikki data, jota TCP:n kautta lähetetään, saapuu perille samassa järjestyksessä kuin se lähetettiin. Näiden ominaisuuksien lisäksi TCP-protokolla toteuttaa lähetettävän tiedon jakamisen pienempiin paketteihin, jotka se lähettää vastaanottajalle. (Fiendler, 2008 & Spurling, 2004)

TCP-protokollassa on joitakin ongelmia, jotka estävät sen käyttöä kaikissa moninpeleissä. TCP:n kautta lähetetyt paketit kasataan puskuriin, jota tarkkaillaan, kunnes siinä on tarpeeksi tietoa, joka sitten lähetetään yhtenä suurempana pakettina vastaanottajalle. Tämä voi olla ongelmallista joissakin moninpeleissä, joissa lähetetään paljon tietoa mutta pienissä paketeissa. Ongelma piilee siinä, että TCP katsoo tiedon määrän olevan liian pieni paketin lähettämiseen, eikä pakettia lähetetä heti. Nopeatempoisissa moninpelissä, tieto halutaan välittää palvelimelta pelaajalle mahdollisimman nopeasti. Mikäli näiden pakettien lähettämistä viivytetään ja ne lopulta saapuvat pelaajalle, ei saapunut tieto ole enää relevanttia, koska se on vanhentunutta ja moninpelin pelattavuus kärsii. Vaikka TCP-protokollasta löytyy mahdollisuus pakettien lähettämiseen ilman puskuria, ei tämä protokolla sovellu nopeatempoisiin moninpeleihin. (Fiendler, 2008 & Spurling, 2004)

Seuraavaksi ongelmaksi osoittautuu tapa, jolla TCP-protokolla käsittelee pakettien lähettämisen luotettavuuden ja pakettien saapumisen lähetysjärjestyksessä. Ilman että syvennymme TCP:n toimintamalliin, perusmekaniikaltaan TCP:n pakettihallinta on seuraavanlainen: protokolla lähettää paketin, jonka jälkeen se tarkkailee, onko paketti hävinnyt internetiin. Mikäli näin on, se lähettää hävinneen paketin uudelleen. Pakettien kopiot hylätään vastaanottajalla ja paketit, jotka saapuvat epäjärjestyksessä, siirretään sivuun odottamaan aikaisempaa pakettia. Ongelmana on se, että TCP-protokolla pysäyttää tiedon käsittelyn, jos jokin paketti on hävinnyt. Vastaanottajan saadessa uudempia paketteja ennen hävinneen paketin saapumista, ei se voi käsitellä uudempien pakettien sisältöä ennen hävinneen paketin vastaanottamista. Tämä tarkoittaa sitä, että olemme jälleen samassa tilanteessa, kuin jos pienet paketit puskuroitaisiin ja lähetettäisiin isommissa paketeissa. Saatu tieto on jälleen vanhentunutta ja pelikokemus kärsii. (Fiendler, 2008 & Spurling, 2004)

Yleisesti ottaen TCP-protokollaa käytetään moninpeleissä jonkin verran. Pelejä, joissa TCP-protokollaa voitaisiin käyttää, ovat hidastempoiset ja vuoropohjaiset moninpelit, joissa ei ole välttämättä väliä, kuinka tuoretta vastaanotettu tieto on. Nopeatempoisissa toimintapeleissä TPC-protokollaa ei kannata käyttää, koska tieto halutaan välittää palvelimelta käyttäjälle mahdollisimman nopeasti, jotta pelikokemus pysyy pelattavana. (Fiendler, 2008 & Spurling, 2004)

3.2 UDP-protokolla

UDP on toinen verkkoprotokolla tiedonsiirrossa ja yleisesti moninpeleissä käytetty protokolla. UDP on myös todella yksinkertainen tiedonsiirtoprotokolla, ilman TCP:n hienoja ja monimutkaisia ominaisuuksia. Tällä protokollalla voimme lähettää paketteja haluttuun IP-osoitteeseen (esim. 173.194.32.24) ja porttiin (esim. 25005), vastaanottajan tulee vain kuunnella tiettyä porttia (esim. 25005). Paketin saapuessa tähän porttiin miltä tahansa tietokoneelta, saamme tietää, mistä IP-osoitteesta ja mistä portista tämä paketti on lähetetty. Lisäksi saamme tietään paketin koon ja voimme lukea paketin sisällön odottelematta. (Fiendler, 2008 & Spurling, 2004)

UDP on epäluotettava tapa siirtää tietoa, koska lähettäjällä ei ole mitään varmuutta siitä, saapuuko lähetetty paketti koskaan vastaanottajalle tai missä järjestyksessä paketit saapuvat. Yleensä kaikki paketit saapuvat vastaan ottajalle, mutta muutama prosentti paketeista häviää

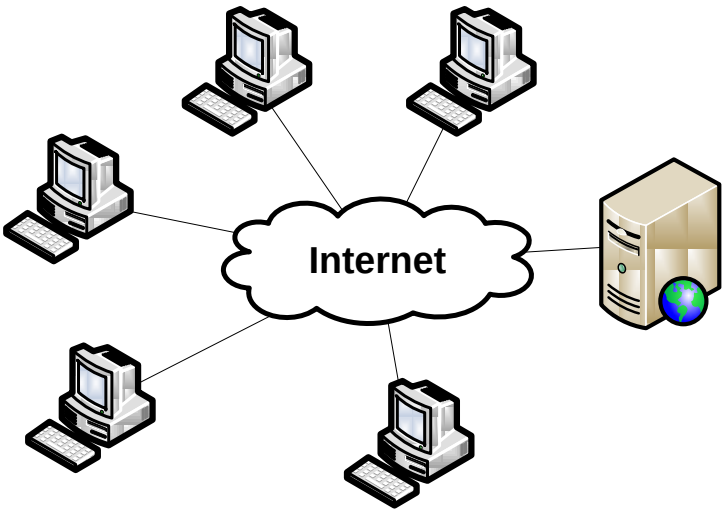
matkalla, eivätkä koskaan saavu vastaanottajalle. Paketit voivat myös saapua vastaanottajalle eri järjestyksessä kuin ne on lähetetty. Esimerkiksi jos lähetämme neljä pakettia järjestyksessä 1, 2, 3 ja 4, ne voivat saapua vastaanottajalle järjestyksessä 4, 2, 1 ja 3. Yleensä paketit saapuvat samassa järjestyksessä kuin ne on lähetetty, mutta tähän ei voi luottaa. Yksinkertaisuudessaan jos lähetät paketin UDP protokollalla, se joko saapuu vastaanottajalle tai ei. Protokolla takaa kuitenkin, että 256 bitin kokoinen paketti vastaanotetaan kokonaisena eli 256 bittinä. UDP-protokollan merkittävä etu TCP-protokollaan verrattuna on UDP:llä läheteltyjen pakettien otsikkokoko. UPD:n otsikkokoko on kooltaan 28 bittia ja TCP kooltaan 40 bittiä. (Fiendler, 2008 & Spurling, 2004)

Useimmat nopeatempoiset moninpelit käyttävät UDP-protokollaa TCP-protokollan sijasta, koska UDP mahdollistaa suurten tietomäärien lähettämisen välittömästi. Yleensä lähetettävä tieto on aikasidonnaista, jolloin TCP:n viivytykset heikentäisivät pelin pelattavuutta. Tiedon vastaanottamisen varmuus ei yleensä ole niin tärkeää näissä peleissä, koska muuttunut tieto lähetetään lyhyen ajan päästä uudelleen. Ei ole järkevää lähettää paljon tietoa ja saada se perille vanhentuneena, sillä parempi vaihtoehto on vain odottaa seuraavaa pakettia. (Fiendler, 2008 & Spurling, 2004)

4 SERVER-CLIENT -MALLI

Verkkopelien server-client -mallin on perusidealtaan hyvin samanlainen kuin mikä muukin server-client -malli, eli kaikki pelaajat ovat yhteydessä samaan pelipalvelimeen, jolla pelimaailma niin sanotusti pyörii. Pelaajat lähettävät kaiken tarvittavan tiedon omasta liikkeestään ja tilastaan pelipalvelimelle, joka sitten päivittää pelaajien tietoja pelimaailmassa. Samaan aikaan pelipalvelin välittää pelihahmojen tiedot muille pelaajille, jotta jokainen pelaaja voi omassa pelissään päivittää muiden pelaajien tilan ja paikan pelimaailmassa. (Fiedler, 2010 & Valve developer community b, 2011)

Kuva 1. Server-client -malli



4.1 Verkkotoiminta malli server-client ympäristössä

Nykyisin FPS-pelien pelipalvelimet toimivat niin sanotusti kiinteällä päivitysnopeudella. Tämä tarkoittaa sitä että palvelimella pelimaailmaa päivitetään esimerkiksi noin 15 ms välein, joka on noin 66,66666… kertaa sekunnissa. Tällä tavalla voimme säästää palvelimen käyttämiä resursseja, koska sen ei tarvitse päivittää pelimaailman tilaa niin nopeasti, kuin se pystyisi. Riippuen pelistä ja pelityypistä palvelimen päivitysnopeutta voidaan säädellä tarpeen mukaan. Korkeampi päivitysnopeus tarkoittaa sitä, että pelimaailma voidaan simuloida tarkem-

min, mutta samalla tarvitsemme enemmän laskentatehoa itse palvelimelta. (Fiedler, 2010 & Valve developer community b, 2011)

Jokaisella päivityskerralla palvelin käsittelee pelaajien lähettämät viestit, suorittaa fysiikan laskemisen, tarkistaa pelin säännöt ja päivittää peliobjektien tilaa saamiensa tietojen perusteella. Jokaisen päivityskerran jälkeen palvelin tarkistaa, mitä tietoa pelaajille tulee välittää. Voimme vaikka välittää kaikkien pelaajien ja maailman objektien tilamuutokset, mutta se ei olisi järkevää, varsinkin jos pelimaailmassa on kymmeniä erilaisia liikkuvia objekteja. Alla olevassa taulukossa 1 on laskettu, kuinka paljon liikennettä syntyisi yhtä pelaajaa kohti eri pelaajamäärillä, kun kaikki tiedot lähetetään 30 kertaa sekunnissa ja vain pelaajan paikka ja kierto lähetetään. Pelaajan paikka ja kierto lähetetään viidellä float-tietotyypillä jonka koko on 4 tavua. (Fiedler, 2010 & Valve developer community b, 2011)

Taulukko 1. Tiedonsiirtomääriä.

Pelaaja määrä	Paketti koko (tavua)	Tiedon määrä sekunnissa (tavua)	Tiedon määrä sekunnissa (kilotavua)
1	20 t	600 t	0,58594 kt
6	120 t	3600 t	3,5156 kt
12	240 t	7200 t	7,0313 kt
24	480 t	14400 t	14,063 kt
32	640 t	19200 t	18,75 kt
64	1280 t	38400 t	37,5 kt

Lähetettävän tiedon määrää voidaan pienentää esimerkiksi Dead Reckoning-, Objekti priorisointi- ja Delta Pakkaus -tekniikoilla, joita käsittelen hieman myöhemmin. Yleensä peleissä, joissa on paljon liikkuvia objekteja, nämä objektit priorisoidaan halutulla tavalla, ja kullekin pelaajalle välitetään vain tieto, joka on sillä hetkellä heille tärkeintä. (Fiedler, 2010 & Valve developer community, 2011)

Pelaajilla on yleisesti rajallinen määrä verkkokaistaa, joka voi olla muidenkin ohjelmien käytössä. Pahimmassa tapauksessa pelaaja käyttää vanhaa modeemia, joka voi maksimissaan vastaan ottaa 0,625 – 0,875 kt/s. Pelipalvelimen lähettäessä pelaajalle suuria määriä tietoa joka sekunti, eivät kaikki paketit tulisi perille asti. Joissakin moninpeleissä pelaajilla on mahdollisuus kertoa palvelimelle oman internet-yhteytensä nopeus, jolloin palvelin voi tämän tiedon avulla räätälöidä lähetettävien pakettien koon ja lähetettävät tiedon jokaiselle pelaajalle sopi-

vaksi. Mikäli internet-yhteyden nopeuden ilmoitus ominaisuutta ei ole toteutettu, on hyvä tavoite pitää verkkoliikenteen määrä mahdollisimman pienenä, jotta pelikokemus pysyisi pelattavana myös hitaammilla internet-yhteyksillä. (Fiedler, 2010 & Valve developer community b, 2011)

4.2 Replikoidut, replikoimattomat objektit ja tehosteet

Replikoinnilla tarkoitetaan pelin objektin tai jonkin tapahtuman tilan samaistamista palvelimen ja pelaajan välillä. Tämä tarkoittaa, että kaikki pelin kannalta tärkeät tiedot objekteista tai pelaajista välitetään palvelimelta muille pelaajille. Näin pyritään pitämään pelaajien ja palvelimen välillä yhtenäinen pelimaailma. Karkeasti jaoteltuna pelissä voi olla replikoituja ja replikoimattomia objekteja tai visuaalisia tehosteita. (Nalezynski & Polge & Sweeney, 2010)

Replikoimattomat objektit ovat yleensä visuaalisia tehosteita, joita ei tarvitse näyttää palvelimella, jos kyseessä on keskitetty palvelin (dedicated server), koska nämä tehosteet eivät yleensä vaikuta itse pelimekaniikkaan millään tavalla. Nämä visuaaliset tehosteet luodaan yleensä jonkin toisen replikoidun toiminnon yhteydessä. Replikoidut objektit voidaan jakaa kahteen tyyppiin, aina replikoituihin ja vain aluksi replikoituihin. Aina replikoituja peliobjekteja ovat yleisesti kaikki ne pelissä olevat objektit, joiden liikettä ei voida ennustaa, tai ne ovat merkittävä osa pelimekaniikkaa. Pelin pelaajat ovat tällaisia aina replikoituja objekteja. Vain aluksi replikoituja objekteja voi esimerkiksi olla pelimaailmassa lentävä raketti. Esimerkkinä raketin kulkurata voidaan ennustaa kaikilla pelaajilla ja palvelimella tarkasti, jolloin se voidaan replikoida vain kerran, kun se luodaan pelimaailmaan. Replikoinnin jälkeen objekti elää ja toimii sekä palvelimen että pelaajan pelimaailmassa, kunnes se osuu johonkin ja räjähtää, jonka jälkeen se poistetaan pelimaailmasta. (Nalezynski & Polge & Sweeney, 2010)

4.3 Pelaajasyöte ja pelaajan toiminnot

Pelaajan komennot/tiedot päivitetään palvelimelle yleensä joko mahdollisimman nopeasti tapahtuman jälkeen tai jollakin kiinteällä aikavälillä esimerkiksi 33,3333… ms, mikä tarkoittaisi noin 30 kertaa sekunnissa. Lähetettäessä pelaajan komentoja/tietoja kiinteällä aikavälillä, tulee jokaiseen pakettiin kasata useita komentoja/tietoja viimeksi lähetetyn paketin ja lähetet-

tävän paketin väliltä. Mikäli peli on rakennettu siten, että komennot lähetetään mahdollisimman nopeasti komennon tapahtumisen jälkeen, tämä ratkaisu vaatii myös pelaajan internet-yhteydeltä enemmän verrattuna kiinteällä aikavälillä lähettyihin paketteihin. (Valve developer community b, 2011)

Itse pelaajatoimintojen toteuttamiseen on olemassa kolmella eri vaihtoehtoa. Voimme antaa pelaajalle täyden vallan pelihahmonsa suorittamiin toimintoihin, voimme varmistaa jokaisen toteutettavan toiminnon palvelimella tai voimme antaa pelaajalle mahdollisuuden toteuttaa toimintonsa omassa pelimaailmassaan heti, mutta silti tarkistaa palvelimelle, voiko toimintoa suorittaa. Näistä kolmesta vaihtoehdosta viimeinen on yleisesti käytössä oleva ja suositeltava tapa, jotta pelikokemus säilyy nautinnollisena, mutta valta pelin tapahtumista on palvelimella ja huijaaminen on hankalampaa. (Valve developer community b, 2011)

Täyden vallan antaminen pelaajalle kaikkien toimintojen suorittamiseen johtaisi tilanteeseen, jossa pelaaja voisi pelissä huijaamalla esimerkiksi juosta nopeammin kuin muut. Tämä olisi mahdollista, jos pelaaja pääsee muokkaamaan pelin logiikkaa ulkopuolisen ohjelman kautta. Tämä ei tietenkään ole toivottua, koska muiden pelaajien pelikokemus kärsii. Mikäli varmistaisimme jokaisen toiminnon palvelimella, joutuisimme tilanteeseen, jossa pelin pelaaminen ei olisi enää sulavaa, koska jokaisen askeleen ottamiseen kuluisi sama aika kuin paketin lähettämisellä palvelimelle ja palvelimen vastauksen vastaanottamiseen. Tämä ratkaisu eliminoisi mahdollisuuden huijata pelaajan toimintojen suorittamisessa, mutta pelikokemus kärsisi. (Valve developer community b, 2011 & Gambetta, 2010)

Kolmas ja suositeltava tapa on antaa pelaajalle valta toimintojen suorittamiseen omassa pelissään, mutta antaa palvelimelle valta tarkistaa kaikki pelaajan suorittamat toiminnot. Käytännössä tämä tarkoittaa sitä, että kun pelaaja toteuttaa omassa pelissään jonkin toiminnon, tieto siitä välitetään palvelimelle. Palvelimen saatua tiedon toiminnosta se tarkistaa, voiko pelaaja suorittaa tämän toiminnon, ja jos voi, pelaajan toiminto suoritetaan myös palvelimella. Tilanteessa jossa pelaaja ei voikaan suorittaa toimintoa, palvelin lähettää tästä tiedon takaisin pelaajalle, pelaajan tila päivitetään oikeaksi ja mahdollisesti suoritettava toiminto peruutetaan. (Valve developer community b, 2011 & Gambetta, 2010)

4.4 Pelaajasyötteen ennustaminen

Pelaajasyötteen ennustaminen liittyy olennaisesti pelaajatoimintojen ja pelaajan liikkeen käsittelyyn, kun pelaajalle on annettu valta suorittaa toiminnot omassa pelissään ja palvelin varmistaa toimintojen laillisuuden. Pelaajaennustuksen tarkoituksena on poistaa viive, joka tapahtuu, kun paketti lähetetään palvelimelle ja palvelin vastaa palaajalle.

Esimerkiksi tilanteessa, jossa ennustamista ei toteuteta, pelaaja liikkuu eteenpäin ja verkkoliikenteen viive on noin 100 millisekuntia, pelaaja lähtee liikkumaan eteenpäin pelimaailmassa, jolloin pelaajasyöte tallennetaan ja se välitetään palvelimelle. Palvelin liikuttaa pelaajaa pelimaailmassa ja välittää tämän tilan muutoksen kaikille pelaajille. Tämä tarkoittaa sitä, että liikkunut pelaaja näkisi oman liikkeensä noin 100 millisekunnin viiveellä ja sama viive toistuisi jokaisessa pelaajan suorittamassa toiminnossa. Viive pelaajan toiminnoissa aiheuttaa epäluonnollisen tunteen, pelin pelaaminen vaikeutuu ja kaiken kaikkiaan pelikokemus kärsii. (Valve developer community b, 2011)

Pelaajan ennustaminen on tapa poistaa tämä viive ja antaa pelaajalle tunne toimintojen välittömästä toteuttamisesta. Sen sijaan, että odottaisimme palvelimelta vastausta jokaisen toiminnon kohdalla, pelaaja ennustaa näppäinsyötteen perusteella, mitä tulevaisuudessa tulee tapahtumaan. Tämä tarkoittaa sitä, että pelaajan on toteutettava ennustus täsmälleen samalla tavalla, kuin palvelin toteuttaa pelaajan suorittaman toiminnon. Pelaajan toteuttaessa ennustuksen, se päivittää pelaajan tilan välittömästi ennustuksen mukaisesti, tässä vaiheessa palvelin näkee pelaajan vielä sen vanhassa tilassa. (Valve developer community b, 2011)

Pelaajan saadessa palvelimelta tiedon omasta tilastaan, joka perustuu pelaajan välittämään näppäinsyötteeseen, pelaaja vertaa omaa ennustustaan palvelimelta saatuun tietoon. Mikäli saaduissa tiedoissa on eroavaisuuksia, on pelaajalla tapahtunut virhe ennustamisessa. Tämä tarkoittaa sitä, että pelaajan tulee korjata ennustuksessa tapahtunut virhe, koska palvelimella on aina päätösvalta kaikessa. Riippuen tapahtuneesta virheestä ennustamisessa, voi tapahtunut virhe olla suurikin. Pelaajan välitön virheen korjaaminen, saattaisi muuttaa radikaalisti objektien ja pelaajan paikka tai tila. Paikan muutoksessa tämä tarkoittaisi sitä, että objektit hyppäisivät ennustetusta paikasta toiseen yhtäkkiä. Virhe paikassa tai tilassa voidaan korjataan hiljalleen esimerkiksi puolen sekunnin tai sekunnin aikana, näin voidaan parhaassa tapauksessa virheen tapahtuminen piilottaa pelaajalta kokonaan. (Valve developer community b, 2011)

Pelaajan ennustaminen toimii vain, jos pelaaja tietää täsmälleen samat pelin säännöt, mekaniikat ja objektien tiedot kuin pelin palvelin. Tämä ei aina pidä paikkansa, sillä yleensä palvelimella on enemmän tietoa kuin yksittäisellä pelaajalla, koska pelaaja näkee yleensä vain osan pelimaailmasta. Joten pelaajan ennustaminen toimii vain paikallisella pelaajalla ja hänen toteuttamillaan toiminnoilla. Muiden pelaajien ja niiden toimintojen ennustaminen paikallisella pelaajalla on lähes mahdotonta varsinkin FPS-peleissä. (Valve developer community b, 2011)

4.5 Dead Reckoning

Dead reckoning on yksi tapa ennustaa peliobjektien paikan muutosta ja pienentää lähetettävän tiedon määrää. Perimmäinen ongelma, miksi tätä halutaan toteuttaa, piilee internetissä ja siinä, että pakettien välittämiseen internetin kautta kuluu jonkin verran aikaan, riippuen yhteyden nopeudesta ja välimatkasta lähetys kohteeseen. Verkkoviive ei ole ongelma, kun pelataan hitaita vuoropohjaisia verkkopelejä, viiveestä tule ongelma, kun pelataan nopeatempoisia pelejä. (Aronsonm, 1997)

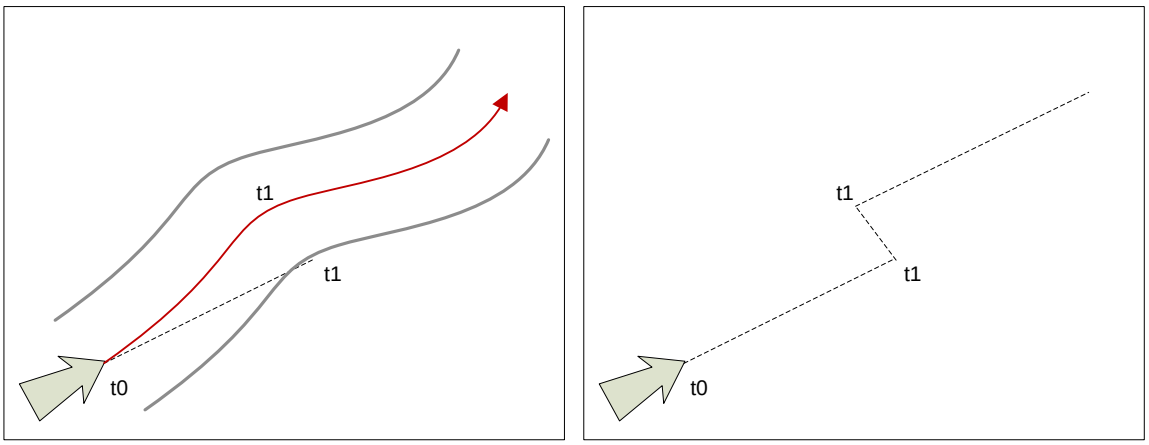
Dead reckoning perustuu siihen, että sekä palvelin että pelaaja käyttävät samoja ennalta määritettyjä sääntöjä/algoritmeja, joilla haluttuja objekteja liikutetaan maailmassa. Pelaajien saatua tiedon jostakin dead reckoning algoritmia käyttävästä objektista, jokainen tiedon saanut pelaaja osapuoli alkaa liikuttaa tätä objektia sovittujen sääntöjen mukaan. Niin kauan kuin objektin liikkeessä ei tapahdu muutosta, kaikki pelaaja osapuolet näkevät objektin liikkuvan sulavasti ja yhtenäisesti. Tämä tarkoittaa myös sitä, että objektista ei tarvitse välittää uutta tietoa pelaajille ja verkkokaistaa voidaan käyttää muihin tarkoituksiin. (Aronsonm, 1997)

Tietenkään kaikkia objekteja ei voida ennustaa koko ajan. Esimerkiksi pelaajaobjekteja ei voida ennustaa koko pelin ajan, koska nämä objektit muuttavat suuntaansa riippuen pelaajan ohjauksessa tapahtuvista muutoksista. Palvelimen saatua tiedon muutoksesta, se laskee objektin liikkeen uudestaan ja jatkaa sen liikuttamista sääntöjen mukaan. Samaan aikaan palvelin vertaa uutta liikettä siihen ennustukseen, joka lähetettiin muille pelaajille. Objektin uuden liikeradan erotessa muiden pelaajien ennustuksesta ennalta määritetyllä tavalla, palvelin lähettää objektin nykyisen liikeradan muille pelaajille. (Aronsonm, 1997)

Alla olevassa kuvassa 2 nähdään esimerkki dead reckoning algoritmistä käytännössä. Objektia ohjaavan pelaajan liike näkyy kuvassa yhtenäisenä punaisena viivana. Harmaat viivat pu-

naisen viivan ympärillä kuvaavat ennalta määritettyä kynnystä ennustuksessa ennen kuin uusi tieto lähetetään. Muiden pelaajien ennustus näkyy kuvassa katkoviivalla. Palvelin lähettää objektin tiedot kaikille muille pelaajille ajan hetkellä t0, tämän jälkeen pelaaja muuttaa objektin liikettä hieman. Hetkellä t1 näkyyn että objektin oikea liikerata eroaa ennustuksesta määritetyn kynnyksen verran. Tällöin palvelin lähettää objektin uudet tiedot kaikille pelaajille ja kun tieto saapuu pelaajille, objektin tiedot päivitetään ja dead reckoning algoritmi suoritetaan näiden uusien tietojen pohjalta. (Aronsonm, 1997)

Kuva 2. Dead Reckoning

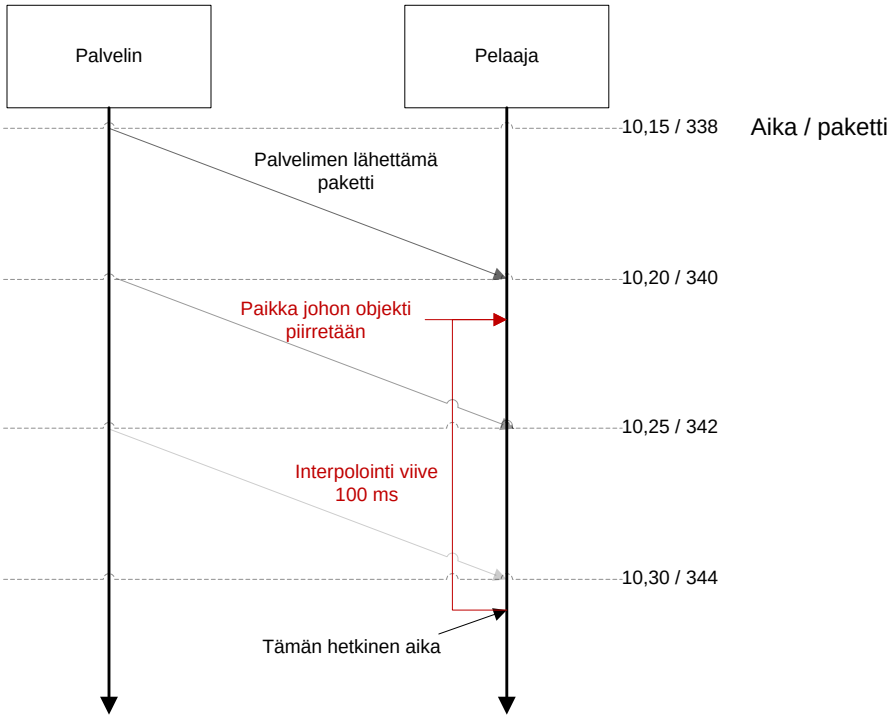


Kuvasta 2 on helppo huomata, että jos pelissä käytetään suuria kynnysarvoja, tuloksena voi olla hyvinkin hyppivä liikerata kaikilla muilla pelaajilla. Toisaalta pienellä kynnysarvolla lähetettävän tiedon määrä kasvaa, kun ennustusta ei voida suorittaa kovinkaan pitkään. On kuitenkin tapa, jolla voidaan käyttää kohtuullisia kynnysarvoja ja silti pitää objektin liike sulavana. Tämä on mahdollista siten, että luomme niin sanotusta haamun ennustetusta objektista ja pelissä pelaajalle näkyvä objekti pyrkii pysyttelemään samassa paikan, kuin sen haamu. Aina kun uusi tieto objektista saapuu, haamu siirretään oikeaan kohtaan saapuneen tiedon mukaan ja ennustusta jatketaan. Haamun paikan päivittämisen jälkeen, voimme liikuttaa näkyvän objektin haamun päälle ennalta määritetyn ajan sisällä, esimerkiksi puolen sekunnin kuluessa siitä, kun haamu siirrettiin. Tällä tavalla voimme ylläpitää yhtenäisen liikkeen ennustetulla objektilla, eikä se hypin paikasta toiseen yhtäkkiä uuden objekti tiedon saapuessa. Tämä kuitenkin vaatii enemmän laskentatehoa jokaiselta pelaajalta, joten on syytä miettiä pelikohtaisesti, millä tavalla dead reckoning algoritmi tulisi toteuttaa. (Aronsonm, 1997)

4.6 Kappaleen Interpolointi (entity interpolation)

Kappaleen interpolointi on tapa piilottaa verkkoviive ja liikuttaa pelin hahmoja tai objekteja ilman ennustamista. Kappaleen interpolointitekniikkaa käytetään yleensä peleissä, joissa objektien liikkumista on hankala ennustaa tai se on täysin mahdotonta. Esimerkiksi nopeatempoisissa FPS-peleissä pelaaja voi vaihtaa liikkumisen suuntaa yhtäkkiä, eikä tätä voida ennustaa millään tavalla. FPS-peleissä pelaaja voi vastaanottaa objektien ja muiden pelaajien tiedot esimerkiksi 30 kertaa sekunnissa. Tietojen päivitysrytmistä johtuen pelihahmojen liike voi näyttää hyvin takkuilevalta ja pätkivältä, jos hahmot siirrettäisiin suoraa uuteen paikkaan maailmassa, tätä ongelmaa voi edelleen pahentaa mahdolliset hukatut paketit. Ongelma ratkaistaan siirtämällä objektia ajassa hieman taaksepäin, hetkeen, jossa objektin paikka voidaan interpoloida kahden viimeisimmän tiedossa olevan paikan välillä. Käytännössä tämä tarkoittaa sitä, että objektien tai pelaajien paikat piirretään muutaman millisekunnin viiveellä. (Valve developer community b, 2011 & Gambetta, 2010)

Kuva 3. Kappaleen interpolointi



Kuvassa 3 on esimerkki kappaleen interpoloinnista, kun se esitetään aikajanan avulla. Viimeisin tieto objektista on saatu pelaajan ajanhetkellä 10,30, paketti 344. Pelaajan päivittäessä uuden pelinäkymän, tämä tapahtuisi ajanhetkellä 10,32. Tässä tapauksessa interpolaatioviive on

100 millisekuntia, piirrettään objekti paikkaan, jossa se oli ajanhetkellä 10,22. Uusi paikka laskettaisiin pakettien 340 ja 342 paikkojen väliltä. Esimerkissä interpolaatioviive on 100 millisekunttia, interpolaatio toimisi, vaikka paketti 342 olisi hukattu. Paikka voidaan edelleen interpoloida pakettien 340 ja 344 tietojen perusteella. Esimerkintapauksessa laskukaava Unity3D pelimoottorissa olisi:

Vector3.Lerp(p340, p342, (10,32 - 0,1 - 10,2) / (10,25 - 10,2))

Kappaleen interpoloinnissa on kuitenkin ongelma, joka riippuu määritetystä interpolaatioviiveestä. Esimerkissä (kuva 3) ongelma tulee vastaan, jos kaksi peräkkäistä pakettia hukataan, esimerkiksi paketit 342 ja 344. Tällöin interpolaatiota ei voida toteuttaa, koska ei tiedetä, missä objekti on 100 millisekunnin päästä. Tässä tilanteessa voimme yrittää ennustaa sen paikan edellisten paikkojen perusteella, eli voimme ekstrapoloida objektin paikkaa. Ekstrapolointia ei kuitenkaan voida suorittaa kuin muutaman millisekunnin päähän, koska mitä pitemmälle paikkaa ekstrapoloidaan, sitä epävarmemmaksi paikan ennustaminen muuttuu.

Kappaleen interpolointi aiheuttaa siis jokaiselle pelaajalle, interpolaatioviiveestä riippuen, viivettä objektien paikoissa. Tämä tarkoittaisi FPS-pelissä sitä, että pelaajien tulisi tähdätä ennakkoon, jotta he osuisivat toisiin pelaajiin. Tämä ongelma voidaan poistaa viivehyvityksellä, koska palvelin tietää tästä viiveestä. Palvelin voi hyvittää tämän viiveen aseen ampumisessa ja sen osumien tarkistuksen yhteydessä. (Valve developer community b, 2011 & Gambetta, 2010)

4.7 Viivehyvitys

Pelaaja ampuu FPS-pelissä aseellansa ajanhetkellä 10.0. Tämä tieto pakataan ja lähetetään palvelimelle seuraavassa paketissa. Paketti lähetetään ja se on liikkeellä palvelinta kohti, palvelin jatkaa pelimaailman simuloimista ja ampumisen kohde siirtyy eri paikkaan maailmassa. Paketti ja tieto ampumisesta saapuvat palvelimelle ajanhetkellä 10.1, jolloin palvelin tarkastaa ampumisen ja kertoo, että pelaaja ei osunut kohteeseen, vaikka pelaaja oli tähdännyt suoraa kohteeseen ja hänen olisi pitänyt osua. Viivehyvityksellä pyritään korjaamaan tämä ongelma, jossa palvelin saa ampumistiedon muutaman millisekunninviiveellä ja kohde onkin siirtynyt eri paikkaan. (Valve developer community b, 2011)

Viivehyvityksen toimimiseksi palvelimen täytyy pitää historiatietoa peliobjektien sijainneista. Palvelimen saatua tiedon toiminnosta, se laskee arvion siitä, milloin tämä toiminto on suoritettu seuraavan kaavan mukaan:

Arvioitu aika = Tämänhetkinen aika - pakettiviive - pelaajan näkymä

Palvelimen arvioitua ajan, jolloin komento suoritettiin, siirtää se kaikkia muita pelaajaobjekteja taaksepäin arvioituun hetkeen. Tämän jälkeen palvelin suorittaa toiminnon ja suorittaa siitä seuraavat toimenpiteet. Toimintojen ja mahdollisten toimenpiteiden suorittamisen jälkeen, palvelin siirtää objektit takaisin niiden alkuperäisiin paikkoihin. (Valve developer community b, 2011)

Näistä toimenpiteistä huolimatta pelaajan ja palvelimen osumatarkistukset eivät koskaan ole täsmälleen samat. Muutaman millisekunninkin ero voi aiheuttaa eriävät tulokset pelaajan ja palvelimen tarkistuksissa, kun kyseessä on nopeasti liikkuvat objektit. Nämä virheet ovat riippuvaisia laskennallisista arvioista objektien paikoissa, palvelimen päivitysnopeudesta ja tarkistettavien objektien liikenopeuksista. Virheitä voidaan vähentää kasvattamalla palvelimen päivitysnopeutta, mutta samalla itse palvelin tarvitsee enemmän laskentatehoa, muistia ja verkkokaistaa. (Valve developer community b, 2011)

Voimme esittää kysymyksen, miksi näitä tarkistuksia ei voitaisi suorittaa pelaajalla paljon tarkemmin, jos tarkistuksien toteuttaminen palvelimella on epätarkkaa? Tällöin pelaaja kertoisi vain palvelimelle, mitä tapahtui. Ongelma on se, että palvelin ei voi luottaa pelaajaan pelimekaniikan kannalta tärkeissä toimenpiteissä ja tarkistuksissa. Pelaajalla olisi mahdollisuus huijata muokkaamalla palvelimelle lähetettyjä viestejä ja täten pilata pelikokemuksen kaikilta muilta pelaajilta. (Valve developer community b, 2011)

Pakettien viiveen takia pelissä voi joskus esiintyä epäloogisia tapahtumia. Esimerkiksi pelaaja A voi osua pelaajaan B, vaikkakin pelaaja B onkin jo seinän takana. Tällaiset tapahtumat johtuvat palvelimella tehdystä viivehyvityksistä, joissa palvelin arvioi pelaajan B olleen näkyvillä, kun pelaaja A ampui. Todellisuudessa tätä ongelmaa ei välttämättä edes huomata, koska lähetetyt paketit liikkuvat pelaajalta palvelimelle todella nopeasti ja peli voi jo itsessään olla hyvinkin hektinen ja nopeatempoinen. (Valve developer community b, 2011)

4.8 Deltapakkaus

Deltapakkauksen tarkoituksena on vähentää palvelimen ja pelaajan välisen liikenteen määrää. Yksinkertaisuudessaan deltapakkauksessa pelaajalle ei lähetetä kaikkea tietoa uudestaan. Sen sijaan pelaajalle lähetetään tiedot, jotka ovat muuttuneet pelimaailmassa pelaajan viimeksi saaneiden tietojen jälkeen. Aina ei kuitenkaan voida kaikkea tietoa lähettää deltapakattuna. Esimerkiksi jos välistä häviää useita paketteja, muutaman sekunnin ajalta, ei deltapakattu tieto ole enää luotettavaa. Tämän takia pelaajille saatetaan lähettää tietyin väliajoin pakkaamaton paketti, joka sisältää kaiken tiedon objektista, eikä pelkkiä muutoksia sen tilassa. Tämä onnistumiseksi, on palvelimen pidettävä kirjaa siitä, mitä tietoa pelaajille on viimeksi lähetetty. Tämän takia jokaiseen palvelimen ja pelaajan välillä lähetettyyn pakettiin liitetään tunnistenumero. Tunnisteen avulla voidaan seurata verkkoliikennettä, mikä paketti on viimeksi saapunut pelaajalle, jos jokin paketti ei ole saapunut perille ja myös jos jokin paketti saapuu uudemman paketin jälkeen. (Valve developer community b, 2011)

4.9 Objektien priorisoiminen

Objektien priorisointia käytetään vähentämään palvelimelta pelaajille lähetettävän tiedon määrää. Objektin priorisoinnin ajatuksena on lähettää pelaajalle vain niiden objektien ja tapahtumien tiedot, jotka ovat pelaajalle tärkeitä kullakin pelin hetkellä. Tämä tarkoittaa sitä, että pelaajalle ei lähetetä tietoa toisista pelaajista, jotka ovat pelaajalle näkymättömissä seinän tai jonkin muun suuren esteen takana. Pelaajalle ei myöskään välitetä tietoa näkymättömissä olevista ammuksista tai muista liikkuvista objekteista, jotka eivät vaikuta pelaajaan millään tavalla hänen sen hetkisessä pelitilanteessaan. Sen sijaan pelaajalle välitetään tiedot objekteista, jotka ovat merkittäviä hänelle: esimerkiksi muut pelaajat, joihin hänellä on näköyhteys, objektit ja ammukset, jotka voivat vaikuttaa pelaajan tilaan jollakin tavalla, sekä objektit, jotka mahdollisesti voivat vaikuttaa näköyhteydessä oleviin muihin pelaajiin. (Aldridge, 2011)

Objektipriorisointi toteutetaan palvelimella aina ennen uuden paketin lähettämistä pelaajille, ja se toimii käytännössä seuraavalla tavalla:

1. Pelimaailman tilanne päivitetään.

2. Arvioidaan jokaisen pelimaailman replikoidun objektin tärkeys jokaista pelissä olevaa pelaajaa kohden.

3. Objektit järjestetään kullekin pelaajalle omaan tärkeysjärjestykseen arvioitujen tärkeyksien mukaisesti.

4. Jokaiselle pelaajalle kootaan lähetettävään pakettiin tiedot tärkeysjärjestyksen ja mahdollisten muiden ehtojen mukaisesti.

Priorisoinnissa avaintekijöitä ovat pelaajien näkökentät ja näköyhteys eli mitä objekteja pelaajat voivat nähdä sen hetkisestä paikastaan ja näiden nähtävissä olevien objektien etäisyys pelaajaan. Muita avaintekijöitä pelistä riippuen voivat olla esimerkiksi objektin vaarallisuusaste pelaajaan nähden, milloin objektin tiedot on viimeksi välitetty pelaajalle ja kuinka usein objekti tiedot tulisi lähettää. Priorisoinnissa käytettävät tekijät voivat vaihdella hyvinkin paljon eri pelityyppien välillä, ja joissakin peleissä avaintekijöinä voivat olla aivan eri asiat. Yleisesti tärkeimpiä tekijöitä priorisoinnissa ovat objektien näkyvyys ja niiden etäisyys pelaajaan. (Nalezynski & Polge & Sweeney, 2010 & Aldridge, 2011)

Eri pelimoottorisessa on käytössä erilaisia tapoja, joilla objektien priorisointi on toteutettu. Pelimoottoreista Valven Srouce-pelimoottori käyttää näkyvyysalueita objektien priorisoinnissa. Näkyvyysalueilla tarkoitetaan alueita, joilla pelimaailma on jaettu erikokoisiin osiin, jokainen näistä alueista kertoo, mitä pelaaja voi nähdä, kun hän on jonkin alueen sisällä. Näkyvyysalueet ovat aina liitoksissa toisiinsa, ja liitosten avulla tiedetään, mitkä muut alueet pelaajan on mahdollista nähdä eri alueilta. Tätä tietoa voidaan käyttää hyväksi objekteja priorisoidessa, eli pelaajalle voidaan välittää vain tieto objekteista, jotka sijaitsevat joko nykyisessä näkyvyysalueella tai siihen liitoksissa olevilla alueilla. (Valve developer community a, 2011 & Valve developer community c, 2011)

Epic Games:n Unreal Engine käyttää pelkästään näköyhteyslaskentaa, ja sen apuna käytetään monia muita eri tekijöitä objektien tärkeyden määrittämiseen. Näköyhteyslaskennassa pelimaailmaa ei ole jaettu erillisiin osiin, vaan sitä käsitellään yhtenä yhtenäisenä alueena. Näkyvyyslaskennassa objekteista lasketaan suora säde toisiin objekteihin, joista jokainen säde tarkistetaan osuuko se johonkin muuhun objektiin matkan varrella. Tässä on kuitenkin ongelma suurien objektien kanssa. Laskettu säde voi osua matkan varrella johonkin toiseen objektiin, mutta kohdeobjekti on kuitenkin näkyvillä, vaikka laskenta väittää, ettei se olisi. Unreal En-

gine käyttää tässä apuna jonkinlaista heuristiikka, josta ei ole saatavilla julkista dokumentaatiota kirjoitushetkellä. (Nalezynski & Polge & Sweeney, 2010)

5 PROTOTYYPPI PROJEKTI

Opinnäytetyön käytännön projektina toteutettiin server-client moninpelin prototyyppi, jonka päälle rakennettiin hyvin yksinkertainen FPS-moninpelin. Tämä tarkoittaa sitä, että prototyyppi on graafisesti hyvin alkeellinen, mutta sen tarkoituksena on testata ja demonstroida tässä opinnäytetyössä käsiteltyjä moninpelitekniikoita. FPS-moninpeli valittiin toteutukseksi siitä syystä, että tämän tyyppisissä peleissä tietoa joudutaan välittämään suuria määriä lyhyessä ajassa.

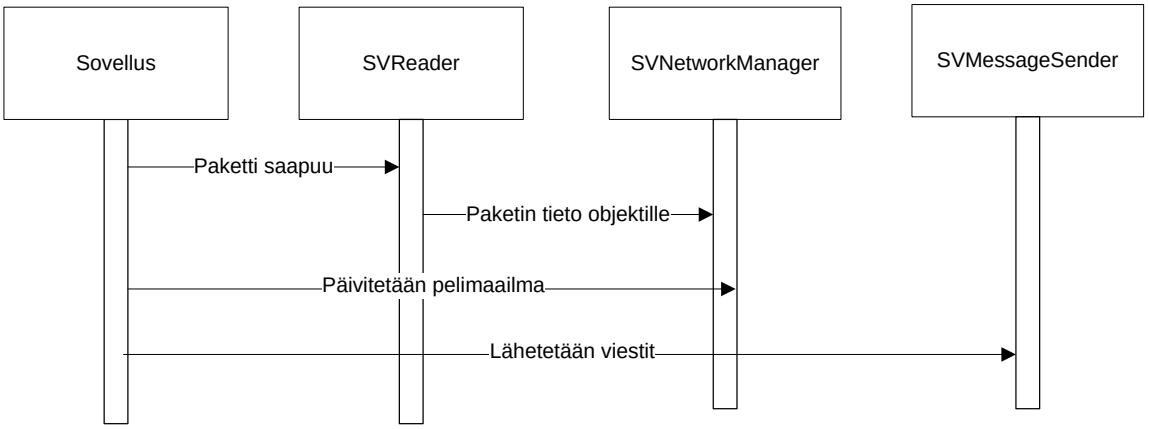
Tavoitteena oli toteuttaa toimiva moninpelin prototyyppi, jota voitaisiin jatkokehittää opinnäytetyöprojektin valmistumisen jälkeen. Henkilökohtaisella tasolla tavoitteena oli oppia ja tutustua käytännössä yleisimpiin moninpelitekniikoihin ja siihen, millaisia ongelmia ja haasteita niiden kehitys sisältää. Prototyyppi toteutettiin kesän 2012 aikana käyttäen C# -ohjelmointikieltä, Unity3D-pelimoottoria ja lidgren network gen3 -verkkokirjaston 26.5.2012 julkaistua versiota.

5.1 Arkkitehtuuri

Prototyypin arkkitehtuuri perustuu server-client -malliin, projektin toteutuksessa palvelin on aina erillinen sovellus. Tämä tarkoittaa sitä, että pelaaja ei voi toimia niin sanotusti palvelimena ja pelata samaan aikaan samalla käynnissä olevalla sovelluksella. Prototyypissä kaikki peliä koskeva verkkoliikenne kulkee palvelimen läpi, jossa se toteuttaa erinäisiä toimenpiteitä, riippuen pelin tapahtumista. Pelaajan kannalta merkittävistä pelin tapahtumista tieto välitetään pelaajalle, joka tiedon perusteella päivittää oman pelimaailmansa tilan.

Prototyyppitoteutuksessa on kolme pääkomponenttia: SVNetworkManager, SVReader ja SVMessageSender. Nämä kolme komponenttia toteuttavat verkkopelin toiminnallisuuden. Lisäksi jokaisella replikoitavalla objektilla on SVNetObject-komponentti, jonka avulla määritetään, millä tavalla objekti replikoidaan muille pelaajille, Dead Reckoning vai Entity Interpolation, ja kuinka tärkeä objekti on priorisointia toteutettaessa. Toteutuksessa jokainen replikoitu objekti käsittelee niitä koskevat viestit itsenäisesti. Tämä on toteutettu siten että, SVReader-komponentti lukee kaikki sisään tulevat viestit ja välittää nämä viestit eteenpäin niille objekteille, joita viestit koskevat.

Kuva 4. Päivitys logiikka.



Kuvassa 4 on kuvattu ohjelman toiminnallisuutta jokaisella päivityksellä. Jokaiselle päivityksellä SVReader lukee sovellukselle lähetetyt paketit ja välittää niissä olevat tiedot SVNetworkManager komponentin kautta niille replikoiduille objekteille, joille tieto on osoitettu. Tämän jälkeen kaikkien pelissä olevien objektien päivitykset toteutetaan normaalisti, jonka aikana SVNetObject-komponentti käsittelee saapuneet tiedot ja tekee tarvittavat päivitykset objektin tilassa. Kaikkien objektien päivityksien suorittamisen jälkeen, SVMessageSender lähettää palvelimelle tai pelaajille seuraavat paketit. Esitellystä logiikasta poikkeavia erikoistapauksia ovat pelaajien peliin liittymiset ja pelaajien poistumiset. Näissä tapauksissa SVReader toteuttaa tarvittavat toiminnot välittömästi, kun se saa tiedon tapahtumista.

5.2 Projektin kulku ja toteutetut ominaisuudet

Projekti aloitettiin pienimuotoisilla testeillä, joilla varmistettiin käytetyn verkkokirjaston toimivuus Unity3D:n ilmaisen version kanssa. Näiden testien jälkeen rakennettiin pohja verkkotoiminnallisuudelle. Ensimmäiset varsinaiset viestit palvelimen ja pelaajan välillä saatiin toimimaan jo toisena projektipäivänä, jonka jälkeen toteutettiin perusverkkopelitoiminnallisuus, kuten pelaajan liittyminen ja pelaajien poistuminen pelistä. Tässä vaiheessa projektia SVReader oli toteutettu toimimaan erillisessä säikeessä, jotta kaikki saapuvat viestit päivitettäisiin objekteille, heti kun ne saapuvat. Säikeistys kuitenkin tuotti ongelmia, koska säie lopetti toimintansa kesken ohjelman ajamisen tuntemattomasta syystä. Näiden ongelmien aikana huomattiin Unity3D-pelimoottorissa olevan Script Executon Order-ominaisuus, jolla voidaan vaikuttaa pelimoottorin komponenttien toimintajärjestykseen. Säikeistys ongelman poistamiseksi, siirryttiin käyttämään Script Executon Order-ominaisuutta, jotta voitiin olla

varmoja siitä, että kaikki saapuneet viestit käsitellään ennen muiden objektien päivittämistä. Näiden ongelmien jälkeen prototyypissä oli toiminnallisuus pelaajan liittymiselle ja pelaajan poistumiselle.

Seuraavaksi toteutettiin toiminnallisuus objektien luomiselle ja poistamiselle. Nämä ominaisuudet liitettiin pelaajan liittymiseen ja poistumiseen siten, että pelaajalla on 3D-malli sekä palvelimella että pelaajalla. Samaan aikaan toteutettiin normaalit FPS-kontrollit, jotta pelaaja voi liikkua pelimaailmassa. Kontrollien valmistuessa ja toimiessa pelaajan päässä, aloitettiin pelaajasyötteen replikoinnin toteuttaminen. Pelaajaobjektien liikkuessa hyvin sekä paikalliselle pelaajalla ja palvelimella, alettiin toteuttaa kappaleen interpolointimekaniikkaa, jota käytetään pelaajan paikan replikoinnissa muille pelaajille. Tässä vaiheessa projektin aloituksesta oli kulunut noin 2 viikkoa. Kuluneesta ajasta suurin osa oli kulunut perusominaisuuksien rakentamiseen, kuten pakettien lähettämiseen, SVNetObject- ja SVNetworkManager-luokan rakentamiseen.

Näiden jälkeen aloitettiin Dead Reckoning toiminnallisuuden toteuttaminen käyttäen yksinkertaista fysiikkaobjektia. Dead Reckoning-ominaisuuden ensimmäinen versio saatiin toteutettua, arkkitehtuurin toteutuksessa ilmeni suurempi ongelma. Ongelma liittyi Dead Reckoning- ja kappaleen interpolointimekaniikkojen valintaan SVNetObject-komponentissa. Muutaman päivän mietinnän jälkeen koodiin tehtiin tarvittavia korjauksia, jotta ongelma saatiin poistettua ja jottei samantapaisia ongelmia tulisi eteen projektin edetessä. Ongelman ratkaisemisen jälkeen toteutettiin ominaisuus, jolla pelaaja voi heittää kranaatin pelimaailmassa. Tämä toiminto toteutettiin, jotta sen avulla voitiin testata Dead Reckoning-ominaisuuden toimintaa. Samalla voitiin toteuttaa peliobjekti, joka replikoidaan vain kerran sen elinkaaren aikana. Samaan aikaan kranaattiin toteutettiin myös visuaalinen partikkeliefekti, joka replikoitiin vain pelaajilla.

Samaan aikaan toteutettiin testi, jossa palvelin saataisiin pyörimään ilman grafiikan piirtämistä, jolloin palvelin käyttäisi vähemmän resursseja. Ominaisuus on mahdollista toteuttaa Unity3D:n -batchmode toimintoa käyttäen. Tässä kuitenkin tuli vastaan Unity3D-pelimoottorin ilmaisen version rajoitukset, sillä batchmode toiminto on käytettävissä vain Unity3D Pro-versiossa. Näin ollen batchmode ominaisuus jäi toteuttamatta prototyyppiä rakentaessa.

Seuraavana suunniteltiin ja toteutettiin yksittäisten tapahtumien replikoiminen. Tapahtumien replikoimista käytetään esimerkiksi pelaajan ampumisen yhteydessä, jolloin viesti tapahtu-

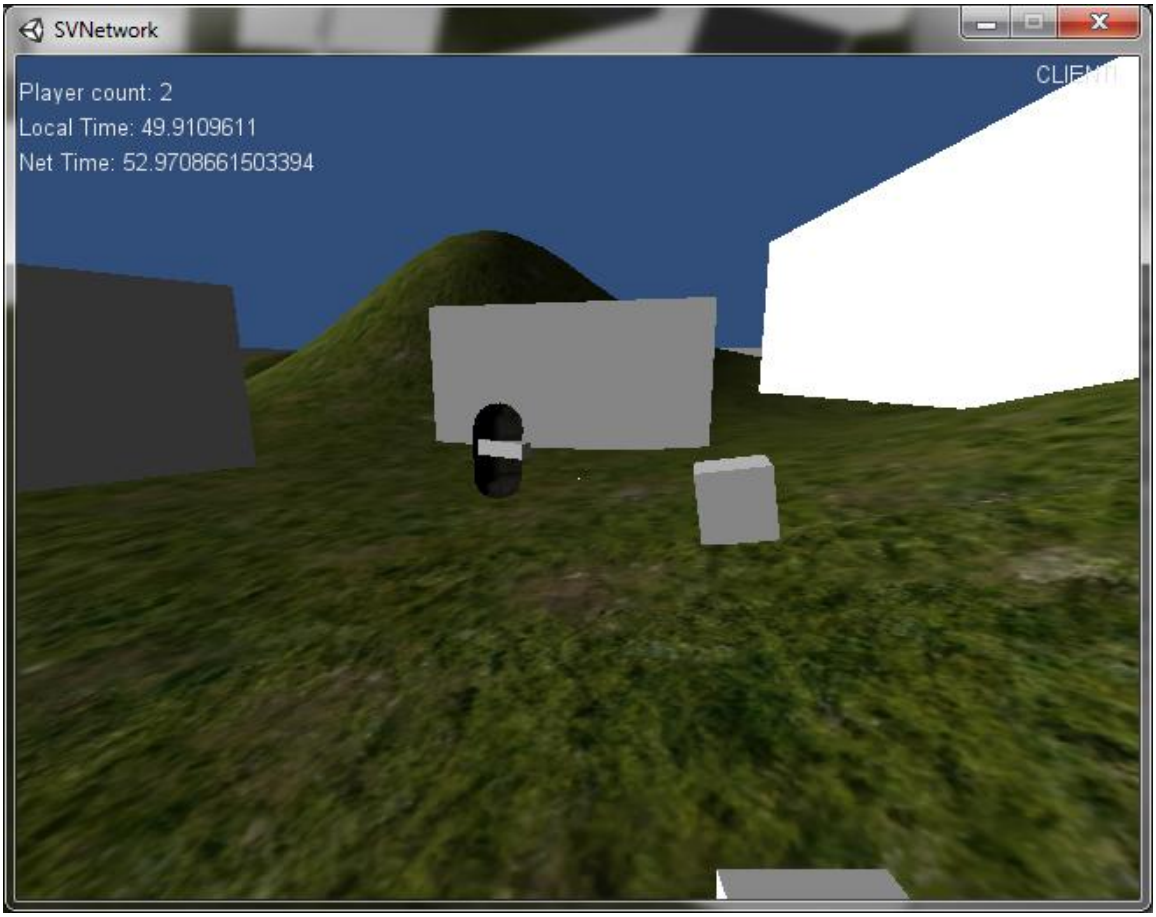
masta välitetään kaikille pelaajille ilman interpolointiviivettä. Tapahtumien replikointi ominaisuuden valmistuttua, toteutettiin hitscan-aseita ja niiden toiminnallisuutta. Hitscan-aseissa pelaajan aseesta luodaan suora viiva tähdättyyn suuntaan, tämä tarkoittaa sitä, että osumat tapahtumat välittömästi, eikä aikaa kulu panoksen lentämisessä. Tässä vaiheessa hitscan-aseet toimivat pelkästään pelaajan päässä, siitä syystä, että niiden toiminnallisuus oli helpoin testata paikallisella pelaajalla. Hitscan-aseiden toimiessa paikallisella pelaajalla, niiden toiminnallisuus siirrettiin palvelimelle. Siirron yhteydessä hitscan-aseiden toiminnallisuuteen lisättiin palvelimen viivehyvitys. Tämän jälkeen lisättiin projektiiliase, jossa pelimaailmaan luodaan fyysinen panos, joka kulkee annetulla nopeudella pelimaailmassa. Myös projektiiliaseeseen lisättiin palvelimen viivehyvitys, sillä prototyypin pelaajaobjektit käyttävät kappaleen interpolointimekaniikkaa. Mikäli viivehyvitystä ei olisi lisätty projektiiliaseeseen, aseen osumatarkistukset eivät olisi toimineet. Pelaajan ampumat panokset on tähdätty paikkaan, jossa toinen pelaaja oli hetki sitten, interpolointimekaniikasta johtuen yksikään näistä pelaajan ampumista panoksista ei olisi osunut kohteeseen.

Priorisointiominaisuutta lisätessä suurimpana haasteena oli se, miten käytännössä ominaisuus toteutetaan. Materiaalia aiheesta löytyy kyllä, mutta mekaniikka on kuvattu hyvin karkeasti. Tämä vaikeutti hieman ominaisuuden toteuttamista, koska varsinainen koodin mekaniikka tuli suunnitella kokonaisuudessaan. Toteutusta suunnitellessa suurimmaksi avuksi osoittautui David Aldridgen I Shot You First! - Gameplay Networking in Halo: Reach -luento, jonka hän esitti vuoden 2011 Game Developers Conference -tapahtumassa. Tämän lisäksi suunnittelussa hieman apua antoi itse luennon pitäjältä. Kävimme muutaman sähköpostiviestin mittaisen keskustelun, jonka aikana saatiin hieman lisätietoa siitä, miten Halo: Reach -pelissä priorisointi on toteutettu.

Suunnittelun jälkeen rakennettiin pohja priorisointiominaisuudelle, siten että vain itse priorisointialgoritmi oli toteuttamatta. Priorisointialgoritmitoteutukseni pohjautuu hyvin pitkälle Halo: Reach -pelin toimintamalliin. Toteutuksessani objektien priorisointiin vaikuttavat etäisyys, aika viimeisestä lähetyksestä, objektin prioriteetti ja objektin tärkeys. Toteutuksessa ei oteta huomioon, näkeekö pelaaja kyseisen objektin, koska tämä vaatisi useita raycasttarkistuksia yhtä objektia kohti. Raycasttarkistuksien määrän kasvaessa, samalla kasvavat palvelimen suorituskyvyn vaatimukset. Algoritmin valmistuttua ja sen toimivuuden tarkistamisen jälkeen, toteutettiin pakettien luominen algoritmin laskemia priorisointiarvoja käyttäen. Tämä onnistui ongelmitta, koska olemassa olevaa pakettien luontilogiikkaa tuli muokata hyvin vähän.

Priorisointiominaisuuden toteutuksen jälkeen projekti oli suurimmilta ominaisuuksiltaan valmis. Tämän jälkeen projektiin lisättiin muutamia visuaalisia tehosteita, jotta kaikkia ominaisuuksia olisi helpompi testata tulevassa, hieman suuremmassa testissä. Ohjelmointiprojektissa kului kokonaisuudessaan noin 1½ kuukautta (6 työviikkoa), mukaan lukien ohjelmiston suunnittelemiseen kuluneen ajan.

Kuva 5. Kuvakaappaus prototyypistä.



5.3 Käytännön testaus

Projektin aikana toteutettiin käytännön testausta aina, kun uusi ominaisuus saatiin lisättyä. Nämä testaukset suoritettiin usealla pelaajasovelluksella, jotka olivat käynnissä samalla tietokoneella kuin itse pelin palvelin. Projektin ollessa loppusuoralla ja vain muutama visuaalinen ominaisuus oli toteuttamatta, toteutettiin käytännön testi internetin välityksellä. Testiin osallistui vain kaksi henkilöä, koska kyseessä oli ensimmäinen internetin välityksellä toteutettu

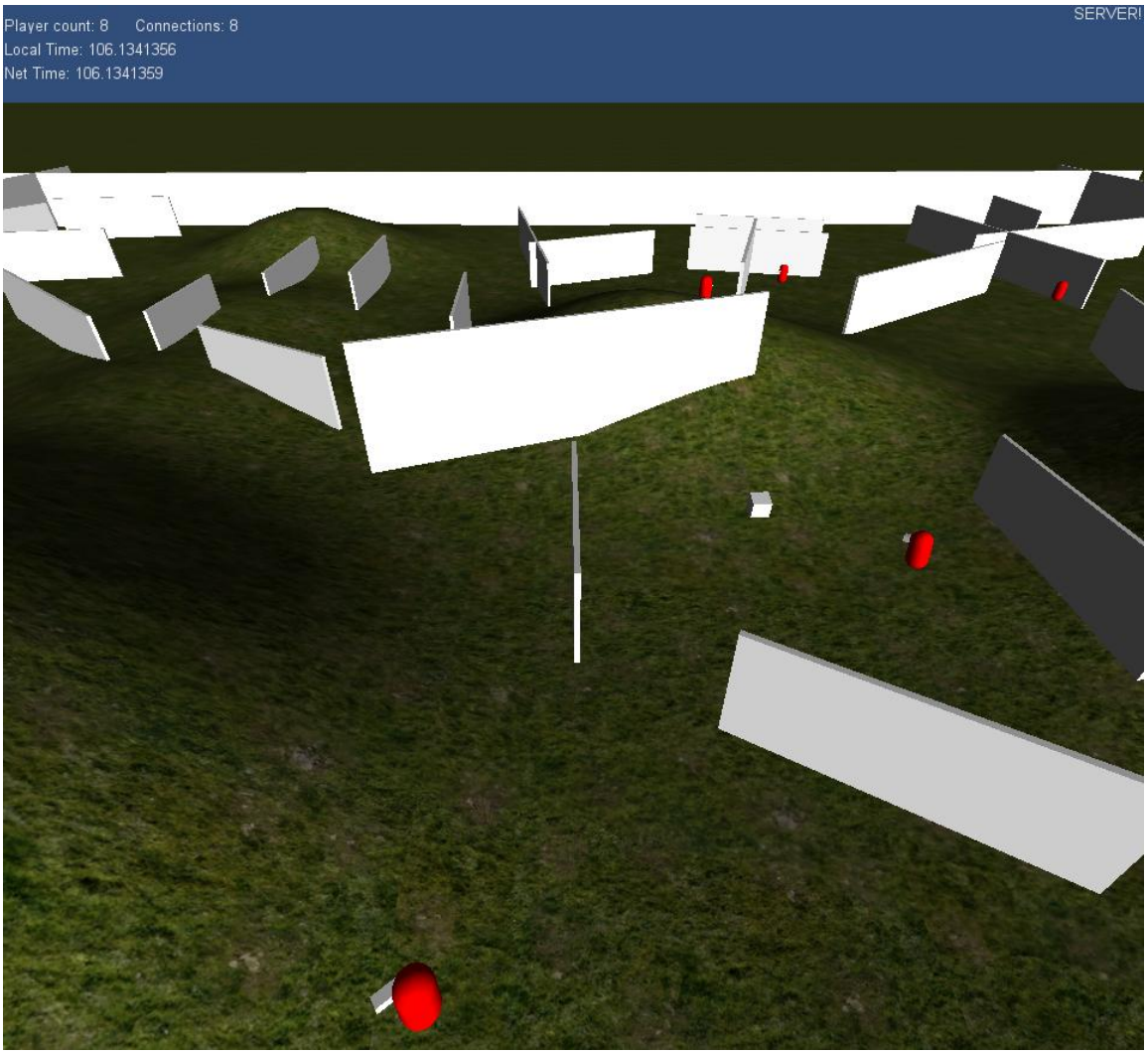
testi. Testissä ei huomattu mitään erikoista ja prototyyppi toimi kaikin puolin hyvin. Näiden testien jälkeen järjestettiin suurempi testi, johon osallistui noin kahdeksan henkilöä. Suuremman testin testaajat värvättiin Facebookin välityksellä ja prototyyppisovelluksen jakelu testaajille suoritettiin Dropbox-palvelua käyttäen.

Suuremmassa testissä ilmeni useitakin eri ongelmia, joista osa saatiin korjattua heti testin jälkeen. Nämä uudet korjaukset testattiin vielä viiden hengen porukalla. Suuremman testin pahimmaksi ongelmaksi ilmeni viiveen määrä ja se lisääntyi sitä mukaan, kun pelaajia tuli enemmän. Ongelma voitaisiin varmaankin korjata muutamilla arkkitehtuuri- ja koodimuutoksilla. Muutamia pienempiä ongelmia ilmeni aseiden osumatarkistuksessa ja palvelimen viivehyvityksen toteuttamassa paikka-arvioinnissa, joka joissain tilanteissa palautti väärän arvon. Vakavampi ongelma ilmeni vasta testin loppupuolella, jolloin pelaajat alkoivat tippua maailman läpi. Ongelma liittyi edelliseen ongelmaan, kun pelaajaan osuttiin mutta paikka-arvioinnissa palautettiin väärä arvo. Ongelma saatiin korjattua, kun palvelimen viivehyvitykseen lisättiin muutama paikan tarkistus. Suurimmaksi ongelmaksi testissä ilmeni pelaajien liikkeen nykiminen, joka johtuu joko palvelimen puolella olevasta synkronisointibugista, priorisointiominaisuudesta, pakettihävikistä tai jostain muusta tällä hetkellä tuntemattomasta tekijästä.

Pienten korjauksien ja muutosten jälkeen prototyyppi toimi jo hieman paremmin. Tässä vaiheessa toteutusta saatiin testattua viiden henkilön voimin, jolloin liikkuminen oli jo suhteellisen sulavaa. Pelaajien liikkumisessa ilmeni edelleen synkronisointiongelmia. Tässä vaiheessa projektia synkronisointiongelmien etsiminen ja niiden korjaamien olisi vaatinut huomattavasti enemmän aikaa, kuin mitä oli käytettävissä. Näiden ongelmien korjaaminen on jatkokehitysosuuden asiaa.

Suuremmasta testissä positiivinen yllätys oli, että prototyyppitoteutus saatiin toimimaan myös yhden testihenkilön Mac OS X Snow Leopard -käyttöjärjestelmässä ilman minkäänlaisia koodimuutoksia. Unity3D itsessään toimii Mac OS X -käyttöjärjestelmässä ongelmitta. Suurin huoli toimivuuden kannalta oli käytetty verkkokirjasto, mutta tämä huoli osoittautui turhaksi. Testeissä kävi myös ilmi, että prototyypin toteutusta olisi hyvä testata suuremmalla porukalla jo aikaisessa vaiheessa. Tällöin osa ongelmista voitaisiin korjata jo aikaisessa vaiheessa, mutta tämä vaatisi enemmän resursseja ja aikaa kuin opinnäytetyöhön on varattu.

Kuva 6. Kuvakaappaus suuremmasta testistä.



5.4 Jatkokehitysajatukset

Alussa ajatuksena oli toteuttaa prototyyppi, jota käyttäen kenen tahansa olisi helppo ja yksinkertainen toteuttaa moninpeli. Tämä ajatus kuitenkin osoittautui hyvin hankalaksi toteuttaa ja todella paljon aikaa vieväksi, joten sitä ei pystytty toteuttamaan ajatellulla tasolla opinnäytetyöhön varatun ajan puitteissa. Tästä syystä projektin tuotosta ei julkaista vapaana lähdekoodina.

Jatkokehityksen osalta projekti on järkevää aloittaa alusta, koska tietoa on nyt enemmän käytännön toteuttamisesta ja ongelmista. Ohjelmiston kooditasolla palvelimen ja pelaajan koodit erotettaisiin toisistaan täysin omiin tiedostoihin, toisin kuin prototyypissä, jossa palvelimen ja

pelaajan logiikkaa on samoissa kooditiedostoissa. Tällainen koodierottelu helpottaisi ylläpitoa ja luettavuutta projektin kasvaessa. Näiden muutosten lisäksi toteuttaisin muutamia seurantatyökaluja, jotka olisivat avuksi verkkopelimekaniikan virheiden etsimiseen. Projektin aikana huomattiin, että itse verkkopelimekaniikan virheenkorjaus voi olla todella hankalaa, varsinkin kun lähetettyjen ja vastaanotettuja pakettien liikennettä ja sisältöä ei voida tarkasti seurata. Samalla olisi myös hyvä kehittää työkalu, jolla voitaisiin seurata palvelimen toimintaa. Palvelimen toimintojen seuraamisella tarkoitetaan lähinnä verkkokaistan käyttöä sekä priorisointialgoritmin tuloksia. Tällaiset työkalut voivat helpottaa työtä todella paljon siinä vaiheessa, kun pieniä virheitä aletaan etsiä tai toteutukseen tehdään jonkinlaisia optimointeja.

6 POHDINTA

Tulevaisuuden moninpeliprojekteissa en ole järkevää toteuttaa pohjaa, jossa olisi kaikenlaisia verkkopelimekaniikkoja valmiina ja jota voisi käyttää mahdollisimman monessa pelissä. Sen sijaan kannattaisi toteuttaa yksinkertaistetun pohjan, jota voisi käyttää verkkopelimekaniikan rakentamisen pohjana. Tällaisessa pohjassa olisivat vain oleelliset toiminnot, esimerkiksi peliin liittyminen, siitä lähteminen, pelaajalistat, pelin sisäinen tekstikeskustelu yms. yleisiä toimintoja, joita on lähes kaikissa moninpeleissä. Tämän pohjan päälle voisi rakentaa ja suunnitella jokaiselle pelille tarpeelliset verkkopelimekaniikat. Projektin aikana yksittäisen verkkopelimekaanikkojen toteuttamiseen ei kulunut valtavasti aikaan, sen sijaan moninpeliperustan rakentamiseen aikaa kului huomattavasti enemmän. Moninpelipohjan rakentamisen jälkeen, erilaisten moninpelimekaniikkojen rakentaminen onnistui suhteellisen vaivatonta. Samalla tällaiseen pohjaan olisi hyvä rakentaa mahdollisuudet erilaisille seuranta- ja mittausohjelmille, jotka auttavat moninpelikoodin optimoinnissa ja virhetilanteissa.

Ohjelmointiprojektin tulos on hyvä, vaikkakin se ei toiminut suuremmilla pelaajamärillä toivotusti ja sen ohjelmakoodi ei ole niin selkeää kuin tavoiteltiin. Projektissa ei käyttänyt valmiina olevia moninpelikirjastoja tai pelimoottorissa olevia moninpelitoimintoja. Ohjelmointiprojektin osoitti, kuinka paljon työtä ja testausta todellisuudessa vaaditaan, jotta moninpeli saadaan toimimaan niinkin hyvin, kuin suuren budjetin konsoli- ja pc-pelitoteutuksissa. Vaikka moninpelitekniikat eivät sinänsä ole kovinkaan monimutkaisia tai hankalia, niiden toiminnallisuuden testaamisessa ja optimoinnissa voi kulua todella paljon aikaa. Toisinaan toimivuuden takaamiseksi joudutaan tekemään muutoksia pelin perusmekaniikkoihin, kuten kävi ilmi David Aldridgen I Shot You First! - Gameplay Networking in Halo: Reach -luennosta.

Kokonaisuutena tämä opinnäytetyöprosessi on ollut todella opettava ja tulevaisuuden kannalta hyödyllinen. Projekti on tuonut tekijälleen todella paljon uutta tietoa moninpelien toteutuksesta. Vaikkakin opinnäytetyössä esitellyt moninpelitekniikat liittyvät suurimmalta osin FPS-verkkopelien toteutukseen, voidaan näitä tekniikoita soveltaa moniin muihin verkkopeleihin.

LÄHTEET

Aldridge, D. 2011. I Shot You First! - Gameplay Networking in Halo: Reach
Saatavilla:
http://www.gdcvault.com/play/1014345/I-Shot-You-First-Networking
http://downloads.bungie.net/presentations/David_Aldridge_Programming_Gameplay_Networking_Halo_final_pub_without_video.pptx
Luettu 12.1.2012

Aronsonm J. 1997. Dead Reckoning: Latency Hiding for Networked Games
Saatavilla:
http://www.gamasutra.com/view/feature/3230/dead_reckoning_latency_hiding_for_.php
Luettu 15.2.2012

Fiendler, G. 2008. UDP vs. TCP
Saatavilla: http://gafferongames.com/networking-for-game-programmers/udp-vs-tcp/
Luettu 15.3.2012

Fiedler, G. 2010. What every programmer needs to know about game networking
Saatavilla: http://gafferongames.com/networking-for-game-programmers/what-every-programmer-needs-to-know-about-game-networking/
Luettu 15.2.2012

Frohnmayer, M & Gift, T. 2000. The TRIBES Engine Networking Model
Saatavilla: http://www.pingz.com/wordpress/wp-content/uploads/2009/11/tribes_networking_model.pdf
Luettu 15.2.2012

Gambetta, A. 2010. Fast-paced multiplayer (part I)
Saatavilla: http://www.gabrielgambetta.com/?p=11
Luettu 15.2.2012

Gambetta, A. 2010. Fast-paced multiplayer (part II)
Saatavilla: http://www.gabrielgambetta.com/?p=22
Luettu 15.2.2012

Gambetta, A. 2010. Fast-paced multiplayer (part III)
Saatavilla: http://www.gabrielgambetta.com/?p=63
Luettu 15.2.2012

Bernier, Y. 2011. Latency Compensating Methods in Client/Server In-game Protocol Design and Optimization.
Saatavilla:
https://developer.valvesoftware.com/wiki/Latency_Compensating_Methods_in_Client/Server_In-game_Protocol_Design_and_Optimization
Luettu: 15.2.2012

Lincrft, P. 1999. The Internet Sucks: Or, What I Learned Coding X-Wing vs. TIE Fighter
Saatavilla:
http://www.gamasutra.com/view/feature/3374/the_internet_sucks_or_what_i_.php
Luettu 15.2.2012

Nalezynski, R.& Polge, S. & Sweeney, T. 2010. Unreal Networking Architecture
Saatavilla: http://udn.epicgames.com/Three/NetworkingOverview.html
Luettu 24.5.2012

Spurling, A. 2004. QoS Issues for Multiplayer Gaming
Saatavilla: http://users.cs.cf.ac.uk/O.F.Rana/data-comms/gaming.pdf
Luettu 15.2.2012

Valve developer community a. 2011. PVS
Saatavilla: https://developer.valvesoftware.com/wiki/PVS
Luettu 24.5.2012

Valve developer community b. 2011. Source Multiplayer Networking
Saatavilla: https://developer.valvesoftware.com/wiki/Source_Multiplayer_Networking
Luettu 15.2.2012

Valve developer community c. 2011. Visibility optimization
Saatavilla: https://developer.valvesoftware.com/wiki/Visibility_optimization
Luettu 24.5.2012