Aapi Elonen

Pelitilan synkronointi verkkomoninpeleissä

Tietojenkäsittelytieteen kandidaatintutkielma

6. toukokuuta 2024

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Aapi Elonen

Yhteystiedot: aapi@tuta.io

Ohjaaja: Tytti Saksa

Työn nimi: Pelitilan synkronointi verkkomoninpeleissä

Title in English: Gamestate synchronization in multiplayer online games

Työ: Kandidaatintutkielma

Sivumäärä: 23+0

Tiivistelmä: Pelitilan synkronointi verkkomoninpeleissä on jatkuvasti kehittyvä uusi tutkimusalue, joka yhdistelee tietoliikennetekniikkaa, ohjelmistotekniikkaa ja teoreettista tietojenkäsittelytiedettä. Tietoliikenteen epäluotettavan luonteen ja monimutkaisten modernien pelimekaniikkojen vuoksi pelitilan synkronoinnista voi muodostua vaikea ongelma. Tutkielmassa käsitellään pelitilan välittämiseen liittyviä optimointimenetelmiä sekä viiveen kompensointimenetelmiä. Tulokset osoittavat että hyvin suunniteltu sovelluskerroksen protokolla ja sopiva suhde pelitilojen yhtenäisyyden ja viiveen välillä mahdollistavat kaikentyyppisten verkkomoninpelien toteuttamisen heikentämättä pelattavuutta.

Avainsanat: pelitila, autoritaarinen palvelin, viiveen kompensointi, determinismi

Abstract: Gamestate synchronization in multiplayer online games (MOGs) has gained increasing attention from the research community in the last two decades. Finding a good balance between gamestate consistency and latency is essential in developing a successful MOG, but can be a daunting process due to the unreliable nature of the network layer and the complexity of modern game mechanics. Discussed in this thesis are methods to optimize and relay gamestate updates, and to conceal the latency perceptible by individual players in client-server architecture MOGs. Results show that by careful planning of the application layer protocol and gamestate synchronization, it is possible to implement highly latency sensitive MOGs without impairing playability.

Keywords: gamestate, authoritative server, latency compensation, determinism

Kuviot

Sisällys

1 Johdanto

Pelitilan synkronoinnilla verkkomoninpeleissä tarkoitetaan menetelmiä, joilla peli pyritään esittämään samanlaisena keskenään vuorovaikutuksessa oleville pelaajille. Toisin kuin klassisissa lautapeleissä ja korttipeleissä, verkkomoninpeleissä pelitila voi päivittyä kymmeniä kertoja sekunnissa, eikä pelitilaa yleensä ole järkevää tai mahdollista lähettää verkon yli jokaisen pelitilan muutoksen yhteydessä. Yhtenäinen pelitila on kuitenkin välttämättömyys pelin säilymiseksi reiluna ja sääntöjen mukaisena.

Verkkomoninpelit yleistyivät kun vuonna 1993 vertaisverkkoteknologiaa hyödyntävä *Doom* osoitti, että sen pelaaminen verkon yli reaaliajassa oli mahdollista. Doomin suurta suosiota kuvaa se, että sen julkaisua seuraavina päivinä esimerkiksi Intelin, Yhdysvaltojen hallinnon ja joidenkin yliopistojen rakennusten aliverkot ruuhkautuivat (Kushner 2003, s. 129-130). Pelaajien välisestä vuorovaikutuksesta verkkomoninpeleissä on tullut entistä monimutkaisempaa, joten moderneihin verkkomoninpeleihin tarvitaan jatkuvasti paremmin optimoituja pelitilan synkronointimenetelmiä.

Pelitilan synkronointia käsitellään tässä tutkielmassa keskitetyn palvelimen arkkitehtuurissa, jossa keskitetty palvelin on *autoritaarinen*, eli se ylläpitää täydellistä kopiota pelitilasta ja välittää tilapäivityksiä pelaajille tarpeen mukaan. Palvelimen toteutuksessa on otettava huomioon pelaajien ja palvelimen välisistä etäisyyksistä johtuvat viive-erot ja virheelliset tai kadonneet viestit. Verkkoyhteyksien epäluotettavan luonteen, pelimekaniikkojen ja viiveerojen vuoksi pelitilan synkronointi luo ainutlaatuisia haasteita verkkomoninpelien kehitykseen, joihin tässä tutkielmassa pyritään löytämään ratkaisuja.

Luvussa 2 tutkitaan kuljetuskerroksen sekä sovelluskerroksen tiedonsiirtoprotokollia ja teknologisten rajoitteiden vaikutusta pelitilan välitykseen. Luvussa 3 keskitytään autoritaarisen palvelimen optimointimenetelmiin. Luvussa 4 käsitellään pelaajien päätelaitteissa ilmenevää puutteellisen tai viivästyneen tiedon kompensointia. Lopuksi yhteenvedossa tiivistetään tutkimuksen tulokset ja pohditaan sen mahdollista vaikutusta myöhemmin ilmestyviin moninpeleihin.

2 Pelitilan välitys

Pelin verkkokomponentilta edellytetään pelityypistä ja mekaniikoista riippuen monia ominaisuuksia. Oliveiran ja Hendersonin (Oliveira ja Henderson 2003) kyselyssä pelattavuuden kannalta tärkeimpiä verkkokomponenttiin liittyviä ominaisuuksia olivat pelitilojen yhtenäisyys, lyhyt viive ja häiriöttömyys. Muita pelitilan välitykseen liittyviä tarpeita ovat mm. luotettava tiedonsiirto, sopivan alhainen kaistanleveys, turvallinen tiedonsiirto ja skaalautuvuus. Ihanteellisesti pelaaminen verkon yli vaikuttaa samalta kuin lähiverkossa riippumatta pelaajien ja keskitetyn palvelimen maantieteellisistä etäisyyksistä (Ferretti 2014). Keskitetyn palvelimen vähimmäisvaatimuksena voidaan pitää asianmukaisesti yhtenäistettyjä pelitiloja, koska vaikka pelissä esiintyykin viivettä, sitä voi silti pelata. Lisäksi viivettä on mahdollista häivyttää jossain määrin pelaajien päätelaitteissa, kuten luvussa 4 esitetyillä viiveen kompensointimenetelmillä.

Verkkokomponentin suunnittelu pelin varhaisessa kehitysvaiheessa kannattaa, koska verkkoominaisuuksien lisäämisestä liian myöhään voi helposti tulla ylipääsemättömän monimutkaista (Gregory 2014, s. 50). Yksi keino varmistaa pelitilan jakamisen kannalta tärkeiden tietojen helppo saatavuus on tapahtumankäsittelijä, jonka kautta kaikki syötteet kulkevat. Verkkokomponentti saa tietoja syötteistä tapahtumankäsittelijältä ja lähettää ne eteenpäin. Verkkokomponenttia toteuttaessa pitää tietää ennalta, millä tavalla se kommunikoi palvelimen tai pelaajien kanssa. Tässä luvussa käsitellään ensin yleisimmät verkkoarkkitehtuurit ja sen jälkeen keskitytään yksityiskohtaisemmin keskitetyn palvelimen arkkitehtuuriin.

2.1 Verkkoarkkitehtuurit

Ferretti (2014, s. 175–196) käsittelee artikkelissaan yleisiä moninpelien verkkokomponenttien ongelmia ja ratkaisuja, sekä niitä yhdistäviä verkkoarkkitehtuureja. Ferretti 2014 jakaa verkkoarkkitehtuurit kolmeen luokkaan:

- Keskitetyn palvelimen arkkitehtuurit
- Vertaisverkkoarkkitehtuurit
- Hajautetut arkkitehtuurit

Keskitetyn palvelimen arkkitehtuuri on yleisin verkkomoninpeleissä käytetty arkkitehtuuri (Singhal 1996, luku 2.1.5; Ferretti 2014, luku 7.1; Roccetti, Ferretti ja Palazzi 2008; Jiang, Safaei ja Boustead 2005). Keskitetyn palvelimen merkittävin huono puoli on sen ylikuormittumisesta tai vikaantumisesta johtuva häiriötilanne, joka estää pelitilan välityksen ja pelaajien välisen kommunikaation (Ferretti 2014, luku 7.1). Hyvänä puolena voidaan pitää sen kykyä toimia pelimaailman *auktoriteettina*; koska kaikki peliin liittyvät tilapäivitykset kulkevat keskitetyn palvelimen kautta, palvelimen on mahdollista varmistaa että kaikki pelaajien lähettämät syötteet ovat pelin protokollan mukaisia. Lisäksi keskitetyssä palvelimessa voidaan ehkäistä huijaamista ja sen toteuttaminen on suhteellisen yksinkertaista. Monimutkaisemmissa peleissä palvelimen toteutuksesta voi tulla hyvin vaikeaa kun pelitilan koko kasvaa, eikä pelitilojen välitys ilman optimointimenetelmiä ole mahdollista. Olennainen edellytys keskitetylle palvelimelle on että pelitilojen välitykseen käytössä oleva kaistanleveys ei ylity, mutta pelitilan rekonstruointi pelaajien päätelaitteissa on edelleen mahdollista.

Vertaisverkkoarkkitehtuurissa kaikki verkon osapuolet toimivat yhdenvertaisina palvelimen tavoin ja pelaajat lähettävät tilapäivityksiä suoraan toisille pelaajille. Vertaisverkkomalli ei sovellu hierarkian puutteen vuoksi hyvin suurempiin peleihin (Smed ja Hakonen 2017). Huonosti optimoitu vertaisverkkoa hyödyntävä moninpeli voi nopeasti ylikuormittaa verkon viestien määrän eksponentiaalisen kasvun seurauksena, jos pelaajat lähettävät tilapäivityksiä kaikille pelin osallisille. Vertaisverkkoarkkitehtuurin synkronointimenetelmiä voidaan hyödyntää osittain myös keskitetyn palvelimen arkkitehtuurissa.

Hajautettu arkkitehtuuri voidaan käsittää kahden edellisen arkkitehtuurin yhdistelmänä. Käytössä on useampi palvelin, jotka ovat keskenään yhdenvertaisia. Pelaajat puolestaan kommunikoivat yhden tai useamman autoritaarisen palvelimen kanssa.

2.2 Teknologiset rajoitukset

Laitteisto ja maantieteelliset etäisyydet asettavat verkkomoninpeleille fyysiset rajoitukset, joiden mukaan niiden on toimittava. Simuloitavien olioiden suuri lukumäärä vaatii jatkuvasti paremmin optimoituja menetelmiä (Hastings, Mesit ja Guha 2005) koska kaistanleveys, viive ja laskennallinen teho rajoittavat edelleen monimutkaisempia simulaatioita (Singhal

1996, luku 1.2). Lisäksi palvelimen toteutuksesta riippumatta lähetetty paketti ei välttämättä aina päädy tarkoitetulle päätelaitteelle esimerkiksi siinä tapauksessa, että jokin reitin varrella sijaitseva reititin on ylikuormittunut eikä kykene käsittelemään pakettia.

Moninpelien verkkoliikenne erottuu selvästi muusta verkkoliikenteestä, koska pelaajien lähettämät paketit ovat yleensä pieniä ja niitä lähetetään lyhyin, tasaisin väliajoin (Armitage ja Stewart 2004; Feng ym. 2005). Yleensä pelaajien lähettämät paketit pyritään pitämään tarpeeksi pieninä, jotta palvelimen vastaanottamat ja lähettämät tilapäivitykset eivät ylikuormita palvelinta ja käytettävissä oleva kaistanleveys ei ylity (Lee ym. 2003). Kaiserin (Kaiser ym. 2009) mukaan yksittäiseltä pelaajalta vaadittava kaistanleveys tyypillisessä verkkomoninpelissä on vain muutaman kilobitin kymmenyksen sekunnissa. Pelaajien lukumäärän kasvaessa kaistanleveys voi kuitenkin aiheuttaa pullonkaulan palvelimessa, koska palvelimen pitää pystyä käsittelemään jokaisen pelaajan syötteet. Jiang, Safaei ja Boustead (2005) huomioivat keskitetyn palvelimen huonoja puolia skaalautuvuuden kannalta; sen lisäksi että keskitetty palvelin vikaantuessaan estää kaiken pelitilan jakamisen, palvelimen riittämätön suorituskyky ja kaistanleveys voivat aiheuttaa pullonkauloja ja näiden ongelmien ratkaiseminen on mahdollista vain verkkoarkkitehtuuria vaihtamalla.

2.3 Luotettava ja epäluotettava tiedonsiirto

Verkkosovellukset noudattavat keskinäisen tiedonvälityksen mahdollistamiseksi tietoliikenneprotokollia (Smed ja Hakonen 2017). Verkkomoninpelit käyttävät tyypillisesti joko OSI-viitemallin kuljetuskerroksen TCP- (engl. *Transmission Control Protocol*) tai UDP- (engl. *User Datagram Protocol*) protokollaa (Smed ja Hakonen 2017, luku 11.1.2; Feng ym. 2005). Oman kuljetuskerroksen protokollan toteuttaminen on teoriassa mahdollista, mutta käytännössä vaikeaa käyttöjärjestelmien asettamien rajoitusten vuoksi (Heuschkel ym. 2017).

TCP-protokollassa on mekanismeja, jotka mahdollistavat onnistuneesti lähetettyjen pakettien kuittaukseen perustuvan luotettavan tiedonsiirron. TCP-protokolla soveltuu peleihin, joissa pelitilan yhtenäisyys on pelattavuuden kannalta viivettä merkityksellisempää. UDP-protokollaa kutsutaan epäluotettavaksi, koska se ei sisällä TCP-protokollaa vastaavia sisäänrakennettuja kuittausmekanismeja. UDP-protokollaa käytetään pääasiassa peleissä, joil-

ta vaaditaan lyhyttä palvelimen ja pelaajien välistä viivettä (Feng ym. 2005). Epäluotettavuutta voidaan pitää myös UDP-protokollan hyvänä puolena, sillä epäonnistuneesti lähetettyjä paketteja ei tarvitse lähettää uudelleen jos ne lakkaavat olemasta pelin simulaation kannalta merkityksellisiä ja UDP-protokollassa ei ole samanlaisia viivettä aiheuttavia yleiskustannuksia kuin TCP-protokollassa. Raaen ja Grønli (2014) ovat kartoittaneet tieteellisissä tutkimuksissa löydettyjä viiveen kynnysarvoja, joiden on osoitettu alentavan verkkomoninpelien pelattavuutta: joissakin tapauksissa jo 50 ms viive vaikuttaa pelattavuuteen, joten viiveen merkitys on järkevää ottaa huomioon myös tietoliikenneprotokollan valinnassa.

2.4 Sovelluskerroksen protokollat

Pelinkehittäjän on suunniteltava kuljetuskerroksen protokollan päälle sovelluskerroksen protokolla, joka vastaa viime kädessä pelin sääntöjen mukaisesta pelitilan synkronoinnista. Siinä tapauksessa, että kuljetuskerroksella käytetään epäluotettavaa protokollaa kuten UDP:tä, sovelluskerroksen protokollaan on mahdollista toteuttaa mekanismi täyden tai osittaisen tiedonsiirron luotettavuuden varmistamiseksi sekä mekanismi, jolla varmistetaan että tilapäivitykset tulevat käsitellyiksi oikeassa järjestyksessä. Pakettien käsittelyjärjestys voi olla pelistä ja tilanteesta riippuen esimerkiksi FIFO (engl. *First In First Out*), kausaalinen, täydellinen, tai kausaalinen ja täydellinen järjestys (Ferretti 2014).

Tiedon pysyminen yhtenäisenä ja oikeassa järjestyksessä ei ole itsestäänselvyys, koska mikä tahansa reititin voi matkan varrella muuttaa paketin kulkemaa reittiä. Sovellustason protokollalla tiedon muuttumattomuus ja oikea järjestys on mahdollista varmistaa paketin otsikkotietoihin liitettävillä tiedoilla kuten tarkistussummalla ja järjestysnumerolla. Järjestysnumeron avulla väärässä järjestyksessä vastaanotettu paketti voidaan tallentaa väliaikaisesti myöhempää käsittelyä varten kunnes vanhempi, pienemmän järjestysnumeron sisältävä paketti on käsitelty onnistuneesti.

2.5 Käytännön toteutus

Vuosituhannen vaihteen suositut FPS-verkkomoninpelit (engl. *first-person shooter*) kuten *QuakeWorld* (1996), *Quake III Arena* (1999) ja *Counter-Strike* (2000) suunniteltiin käyttä-

mään autoritaarista palvelinta keskitetyn palvelimen arkkitehtuurissa (Bernier 2001). Pienempiin osiin paloiteltuja tilapäivityksiä kutsutaan näissä peleissä yleisesti komennoiksi. QuakeWorldin sovellustason protokollan toiminta on pohjimmiltaan yksinkertainen, mutta sitä voidaan pitää aikansa edelläkävijänä, koska se oli suunniteltu toimimaan myös hitaammilla yhteyksillä (Armitage, Claypool ja Branch 2006) ja komentojen välittämiseen pelaajan ja palvelimen välillä oli mahdollista käyttää samanaikaisesti sekä luotettavaa, että epäluotettavaa tietoliikennekanavaa. Samaa tekniikkaa voidaan hyödyntää edelleen uusissa peleissä ja siksi on syytä tutkia tarkemmin pelin C-lähdekoodia, joka vastaa pelitilan välittämisestä.

QuakeWorldin luotettava tiedonsiirtokanava toteutettiin lisäämällä sovellustason protokollan otsikkotietoihin viimeisimmän lähetetyn paketin- ja viimeisimmän palvelimelta vastaanotetun vahvistusviestin järjestysnumerot. Lisäksi yksi bitti varattiin merkitsemään luotettavaa pakettia. Protokollassa luotettavat komennot lähetetään niin monta kertaa uudelleen, että niitä vastaava vahvistusviesti on vastaanotettu (Sanglard 2009). Epäluotettavat komennot puolestaan lähetetään vain kerran. Otsikkotietojen jälkeen pakettiin liitetään järjestyksessä ensiksi luotettavat komennot ja sitten epäluotettavat komennot. QuakeWorldin lähdekoodissa (Carmack 1999) tiedonsiirtokanava on määritelty tiedoston `net_chan.c` tietueessa `netchan_t` ja paketit generoidaan funktiossa `Netchan_Transmit`:

```
223 void Netchan_Transmit (netchan_t *chan, int length, byte *data)
```

Funktiolle annetaan parametrina `length` tavun mittainen merkkijono `data`, joka sisältää epäluotettavan kanavan yli lähetettävät komennot. Jos luotettavia komentoja on vielä vahvistamatta, ne ovat merkkijonopuskurissa `chan->reliable_buf`. Lähettävä paketti generoidaan tietueeseen `send` alkaen sovelluskerroksen protokollan otsikkotiedoista:

```
262     w1 = chan->outgoing_sequence | (send_reliable<<31);
263     w2 = chan->incoming_sequence | (chan->incoming_reliable_sequence<<31);
264
265     chan->outgoing_sequence++;
266
267     MSG_WriteLong (&send, w1);
268     MSG_WriteLong (&send, w2);
269
270     // send the qport if we are a client
```

```
271     #ifndef SERVERONLY
272     MSG_WriteShort (&send, cls.qport);
273     #endif
```

`w1` ja `w2` ovat 32-bittisiä etumerkittömiä kokonaislukuja. Lähetettyjen ja vastaanotettujen pakettien järjestysnumeroista merkityksellisin bitti (2^{31}) on varattu merkitsemään luotettavaa tiedonsiirtoa, joten suurin mahdollinen järjestysnumero on $2^{31} - 1$. Jos pakettia ollaan lähettämässä asiakkaan päätelaiteelta, otsikkotietojen lopuksi liitetään porttinumero, jota käytetään pelaajan tunnistamisessa.

```
275     // copy the reliable message to the packet first
276     if (send_reliable)
277     {
278         SZ_Write (&send, chan->reliable_buf, chan->reliable_length);
279         chan->last_reliable_sequence = chan->outgoing_sequence;
280     }
281
282     // add the unreliable part if space is available
283     if (send.maxsize - send.cursize >= length)
284         SZ_Write (&send, data, length);
```

Otsikkotietojen jälkeen kopioidaan pelitilaa muuttavat komennot tietueeseen `send`. Jos luotettavia komentoja on vielä vahvistamatta, totuusehto `send_reliable` on tosi ja pakettiin liitetään ensiksi komennot merkkijonopuskurista `reliable_buf`. Seuraavaksi jos paketissa on tilaa epäluotettaville komennoille, ne kopioidaan `send` tietueeseen merkkijonosta `data`.

```
295     NET_SendPacket (send.cursize, send.data, chan->remote_address);
```

Komentojen liittämisen jälkeen valmis paketti välitetään funktiolle `NET_SendPacket`, joka lähettää tiedot verkon yli UDP-soketin välityksellä.

3 Palvelimen optimointi

Keskitetyn palvelimen optimointi on tasapainottelua viiveen hallinnan ja tiedon oikeellisuuden välillä ja simulaatioilla on niihin liittyen erilaisia vaatimuksia (Savery 2014). Teoreettisesti katsoen pelitilan täydellinen oikeellisuus tarkoittaa sitä, että pelaajat näkevät identtisen kopion palvelimen pelitilasta samaan aikaan niin, että viivettä ei ole mahdollista havaita (Ravindran, Sabbir ja Ravindran 2008). Todellisuudessa pelaajien havaitsemat pelitilat ovat aina viiveestä johtuen hieman poikkeavia, koska tilapäivitysten välittäminen pelaajille tapahtuu viive-eroista johtuen eri aikaan eikä viiveen täydellinen poistaminen ole mahdollista (Liu, Xu ja Claypool 2022).

Ihmisen osallisuus simulaatiossa heikentää pelitilan oikeellisuutta ja pelitilaa on siitä syystä välttämätöntä simuloida pienellä viiveellä. Toisaalta pelin säännöt voivat rajata pelaajien toimintamahdollisuuksia ja mahdollistaa simulaation paremman ennustettavuuden. Viive on tärkeää ottaa huomioon erityisesti nopeissa moninpeleissä, joissa ihmisellä on suuri vaikutus simulaation kulkuun (Armitage 2003; Cronin ym. 2002). Simulaation ennustettavuuteen pelaajan näkökulmasta palataan neljännessä luvussa. Seuraavissa alaluvuissa käsitellään palvelimen optimointimenetelmiä, joilla pyritään saavuttamaan reiluuden ja pelattavuuden kannalta sopiva suhde pelitilan yhtenäisyyden ja viiveen välillä.

3.1 Merkityksellinen tieto

Yksi johdonmukaisimmista keinoista optimoida pelitilan synkronointia on rajata tilapäivitykset merkitykselliseen tietoon, eli siihen pelitilan osa-alueeseen joka voi jollain tapaa vaikuttaa simulaatioon. Smed ja Hakonen (2017, s. 277) luokittelevat tilapäivitykset deterministisiin ja epädeterministisiin tilapäivityksiin. Deterministisiksi tilapäivityksiksi kutsutaan niitä tilapäivityksiä jotka eivät ole lähtöisin ihmiseltä ja epädeterministisiksi niitä jotka ovat. Ihmisen tuottamia tilapäivityksiä kutsutaan epädeterministisiksi, koska ihmisen toiminnan ennustaminen on käytännöllisesti katsoen mahdotonta. Ilman ihmisen tuottamaa satunnaisuutta pelitilaa on mahdollista simuloida aina täsmälleen samalla tavalla seuraavaan epädeterministiseen tilapäivitykseen asti. Jos pelaajalla on käytössään sama informaatio kuin palvelimella,

pelitilan replikoimiseksi sille tarvitsee lähettää vain epädeterministiset tilapäivitykset.

Deterministisyydestä on paljon hyötyä etenkin reaaliaikaisissa strategiapeleissä joissa voi olla ajoittain suuria määriä simuloitavia olioita. Sen sijaan että jokaisen tilapäivityksen yhteydessä lähetettäisiin jokaisen yksittäisen olion tilapäivitykset, mukaan lukien niiden kineettiset ominaisuudet, pelaajan syötteet yleensä riittävät. Pelaajan syötteestä palvelin voi replikoida asiakassovelluksen kanssa yhtenäisen pelitilan deterministisesti mikäli simulointi on toteutettu huolellisesti, eikä siinä ole käytetty esimerkiksi epädeterministisiä liukulukuja (Fiedler 2011).

Pelitilan vanhentuneisuudella (engl. *obsolescence*) tarkoitetaan tilapäivityksiä, jotka ovat lakanneet olemasta kiinnostavia tai merkityksellisiä pelin simuloinnin kannalta. Palvelimen on tässä tapauksessa usein järkevää jättää tilapäivitys käsittelemättä, priorisoiden enemmän merkityksellisiä tilapäivityksiä. Tilapäivitysten ohittaminen nopeuttaa muiden tilapäivitysten käsittelyä, mutta saattaa aiheuttaa poikkeamia palvelimen ja pelaajien synkronoituihin pelitiloihin (Ferretti ja Roccetti 2005).

3.2 Pakkausmenetelmät

Tilapäivitys voidaan pakata esimerkiksi LZW-algoritmien perheeseen (Welch 1984) kuuluvalla häviöttömällä pakkausalgoritmilla tai peliä varten kehitetyllä pakkausmenetelmällä. Tilapäivityksien osittain ennalta määrätyt sisällöt mahdollistavat *deltakoodauksen*, jolla voidaan rajoittaa lähetettävät tilapäivitykset pelitilan muutoksiin, eli millä tavalla edellisen täydellisen pelitilan kopion oliot ovat muuttuneet suhteessa uuteen pelitilaan (Armitage, Claypool ja Branch 2006). Pelitilan muutoksia sisältäviä tilapäivityksiä kutsutaan deltatiloiksi kreikkalaisen Δ-kirjaimen mukaan. Deltatilat voivat esimerkiksi sisältää tietoja pelaajan sijainnista, orientaatiosta ja inventaariosta. Tallentamalla viimeisimmät deltatilat muistiin voidaan lisäksi ehkäistä verkkoyhteyden häiriöitä ekstrapoloimalla tilapäivityksiä käyttäen apuna edellisiä deltatiloja (Stefyn, Cricenti ja Branch 2011).

Olioiden ryhmittely voi pienentää tilapäivityksen kokoa erityisesti suuremman kokoluokan simulaatioissa, koska yksittäisten olioiden sijaan tilapäivityksiä tarvitsee lähettää vain ryhmästä johon ne kuuluvat. Keskeisenä haasteena olioiden yhdistelemisessä on määrittää mitkä

oliot voivat toimia järkevästi ryhmänä (Singhal 1996, luku 2.2.2). Hastings, Mesit ja Guha (2005) ovat esittäneet avaruudellisiin tiivisteisiin (engl. *spatial hashing*) perustuvia ryhmittelymenetelmiä, jotka tiivistävät kaksi- tai kolmiulotteisen tilan yksiulotteiseksi hajautustauluksi.

3.3 Kausaaliset suhteet

Yksinkertaisessa sovelluskerroksen protokollan toteutuksessa tilapäivitykset käsitellään siinä järjestyksessä kuin ne on lähetetty. Täydellisen järjestyksen vaatiminen voi kuitenkin aiheuttaa turhaan viiveettä, jos jokin kausaalisesti riippumaton paketti saapuu käsiteltäväksi ennen seuraavaa simulaation kannalta merkityksellistä pakettia (Roccetti, Ferretti ja Palazzi 2008).

Pelimaailman simulaatiossa oliot eivät välttämättä ole vuorovaikutuksessa keskenään ja silloin ne muodostavat oman kausaalisen ketjunsa, joka on mahdollista käsitellä erillään muista ketjuista. Tapahtumien kausaalisia suhteita hajautetuissa virtuaalimaailmoissa on tutkittu laajemmin esimerkiksi Zhou ym. 2007 artikkelissa.

Tässä luvussa esitettiin muutamia palvelimen optimointiin käytettäviä menetelmiä. Tilapäivitysten rajaaminen merkitykselliseen tietoon, pakkausalgoritmit ja olioiden kausaalisten suhteiden huomioiminen ehkäisee pullonkaulan syntymistä keskitetyn palvelimen ympärille. Hyvin optimoitu palvelin mahdollistaa pelin jatkuvuuden ja pelitilan synkronoinnin tehokkaasti peliin asiaankuuluvalla tavalla.

4 Viiveen kompensointi

Viivettä pidetään erityisesti FPS-moninpelien merkittävimpänä pelattavuutta alentavana tekijänä (Savery 2014, s. 37–38; Briscoe ym. 2016; Liu, Xu ja Claypool 2022). Viiveen kompensoinnilla pyritään häivyttämään pelaajien havaitsemaa viivettä ja poistamaan pelaajien välisistä viive-eroista johtuvia ongelmia. Hyvin suunniteltu asiakassovellus on responsiivinen ja voi parhaimmassa tapauksessa luoda illuusion viiveettömästä tietoliikenneyhteydestä. Viiveen kompensointia on tutkittu paljon; Liu, Xu ja Claypool (2022) ovat tehneet viiveen kompensointimenetelmistä kartoituksen josta käy hyvin selväksi, että pelien yleinen kaupallisuus ja suljettu lähdekoodi ei ole estänyt aiheen tutkimusta.

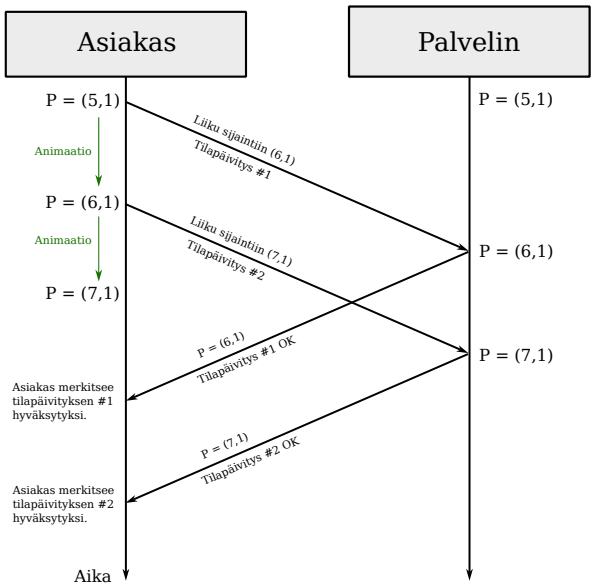
Viiveen kompensointimenetelmät on mahdollista jakaa optimistisiin ja konservatiivisiin menetelmiin (Cronin ym. 2002; Jiang, Safaei ja Boustead 2005; Roccetti, Ferretti ja Palazzi 2008). Optimistiset menetelmät perustuvat oletuksiin simulaation deterministisyydestä ja mahdollisten palvelimen ja asiakassovellusten välisten ristiriitojen selvittämiseen. Konservatiiviset menetelmät sen sijaan hyödyntävät pelin hidasta luonnetta, ja varmistavat pelitilan synkronoinnin viiveen kustannuksella. Konservatiivisia menetelmiä kutsutaan myös synkronisointimenetelmiksi, sillä ne varmistavat että pelitila pysyy jatkuvasti synkronoituna asiakassovellusten kesken.

Tässä tutkielmassa optimistisia menetelmiä kutsutaan toisen vakiintuneen käytännön mukaan ennustaviksi menetelmiksi (Savery 2014; Armitage, Claypool ja Branch 2006; Liu, Xu ja Claypool 2022). Konservatiivisia menetelmiä on käsitelty osana keinotekoista viivettä käsittelevää alalukua 4.3. Aiheesta on rajattu pois menetelmät joiden pääasiallinen tavoite on helpottaa pelaamista, esimerkiksi muuttamalla pelimaailman attribuutteja suhteessa viiveeseen, koska vaikkakin ne kompensoivat viivettä, ne eivät liity varsinaisesti pelitilan synkronointiin.

4.1 Ennustavat menetelmät

Ennustavat menetelmät voidaan jakaa edelleen kahteen luokkaan sen mukaan kompensoivatko ne pelaajaan vai johonkin ulkopuoliseen olioon kohdistuvaa viivettä (Armitage, Claypool

ja Branch 2006, luku 6.2). Vaikka pelaaja näkee oman hahmonsa lokaalissa simulaatiossa reaaliajassa, tilapäivitykset täytyy välittää autoritaariselle palvelimelle, joka hyväksyy tai hylkää pelaajan lähettämän syötteen. Pelaajan ohjaamaan olioon liittyvää ennustamista kutsutaan myös asiakaspuolen ennustamiseksi (engl. *client-side prediction*), koska se perustuu pelitilan simuloimiseen asiakassovelluksessa ennen kuin autoritaarinen palvelin on hyväksynyt tilapäivityksen (Bernier 2001; Liu, Xu ja Claypool 2022). Menetelmä hyödyntää pelin deterministisyyttä, sillä asiakassovelluksen lokaalisti simuloima pelitila päätyy *useimmiten* samaan tilaan autoritaarisen palvelimen kanssa. Ristiriitoja voi syntyä esimerkiksi jos jokin pelaajalle näkymätön tapahtuma on vaikuttanut pelaajan hahmoon ja estänyt pelaajaa toimimasta toivotulla tavalla (Savery 2014; Bernier 2001; Armitage, Claypool ja Branch 2006).



Kuvio 1. Esimerkki asiakaspuolen ennustamisesta

Tieto pelaajan ulkopuolisista olioista välittyy tilapäivitysten rajallisesta lukumäärästä johtuen asiakassovellukselle jaksottaisesti, eikä simuloitavien olioiden liike ole palvelimella yhtä tasaista ja jatkuvaa kuin pelaajien päätelaitteissa. Jaksottaisten tilapäivitysten väliin jäävä tyhjiö on mahdollista täyttää interpoloimalla tai ekstrapoloimalla puuttuvia tiloja. Interpoloimalla voidaan ennustaa olioiden tiloja perustuen viimeisimpiin palvelimelta vastaanotettuihin tilapäivityksiin. Ekstrapoloimalla voidaan laskea olioiden tiloja tulevaisuudessa olettaen, että liike jatkuu samanlaisena (Liu, Xu ja Claypool 2022).

Laskelmasuunnistus (engl. *dead reckoning*) on paljon käytetty ekstrapolointimenetelmä, jota

on historiallisesti käytetty merenkulussa laivojen koordinaattien laskemiseen (Savery 2014, s. 24). Laskelmasuunnistuksella voidaan laskea jonkin olion sijainti P (esimerkiksi) sen aloitussijainnin P_0, nopeuden V_0, ja kiihtyvyyden A_0 perusteella ajan Δt kuluttua (Shi, Corriveau ja Agar 2014; Murphy 2011):

$$P = P_0 + V_0\Delta t + \frac{1}{2}A_0\Delta t^2. \tag{4.1}$$

Laskelmasuunnistus johtaa helposti virheelliseen sijaintiin P, jos liikkeeseen vaikuttavat voimat eivät ole vakioita. Virheellisiä sijainteja on kuitenkin mahdollista korjata sitä mukaa kun reaaliaikaisia sijaintitietoja on saatavilla. Murphy (2011) ehdottaa että kaavalla 4.1 saadut virheelliset sijainnit korjataan niin kutsutulla projektiivisella nopeuksien yhdistämistekniikalla (engl. *projective velocity blending*), jolla lasketaan uusista aloitussijainneista ja nopeusvektoreista kaksi uutta pistettä ja sitten lineaarisesti interpoloidaan niistä uusi väliarvo.

Käytetyn suunnistustavan ei ole välttämätöntä perustua mitattaviin fysikaalisiin suureisiin, vaan suunnistukseen voidaan käyttää myös jotain pelille ominaista polkua. Esimerkiksi Agar, Corriveau ja Shi (2012) ovat esitelleet itse kehittelemäänsä verkkomoninpeliin suunnitellun algoritmin, jonka toiminta perustuu pelaajilta kerättyyn dataan ja toimintamalleihin. Lisäksi Pantel ja Wolf (2002) ovat käsitelleet erilaisia laskelmasuunnistusmalleja osana urheilu-, toiminta- ja rallipelejä.

4.2 Virtuaaliajan manipulointi

Moninpelit joissa oliot voivat muuttaa nopeasti suuntaa vaativat monimutkaisempia viiveen kompensointimenetelmiä. Usein käytetty menetelmä perustuu pelitilojen tallentamiseen ja taaksepäin kelaamiseen. Bernier (2001) kutsuu usein viitatussa artikkelissa tätä menetelmää viiveen kompensoinniksi, vaikka nykypäivänä viiveen kompensointi on muodostunut tarkoittamaan yleisemmin kaikkia viivettä häivyttäviä menetelmiä. Keskitetyn palvelimen tallentamat pelitilat mahdollistavat asiakassovellusten väliset viive-erot sekä asiakkaan ja palvelimen välisten ristiriitojen selvittämisen. Palvelin käsittelee kunkin asiakassovelluksen välittämät syötteet menneisyydessä suhteessa viiveeseen ja mahdollistaa näin myös korkeaviiveisten asiakkaiden syötteiden oikeudenmukaisen käsittelyn (Armitage, Claypool ja Branch

2006).

Viiveen vaikutus pelitilan oikeellisuuteen on havaittavissa hyvin FPS-peleissä, joissa vastustajan ampuminen vaatii tarkkaa ajoitusta ja tähtäämistä (Jiang, Safaei ja Boustead 2005; Armitage, Claypool ja Branch 2006; Liu, Xu ja Claypool 2022). Jos viivettä ei ole huomioitu palvelimen toteutuksessa, pelaajan täytyisi tähdätä sinne, missä vastustaja on sillä hetkellä kun palvelin käsittelee ampumista koskevan tilapäivityksen.

4.3 Keinotekoinen viive

Täydellisen pelitilan yhtenäisyyden vaativissa peleissä, kuten shakissa tai korttipeleissä, viiveellä ei ole pelattavuuden kannalta yhtä keskeinen rooli. Näissä peleissä on mahdollista varmistaa pelitilan yhtenäisyys keinotekoista viivettä lisäämällä, koska lisätty viive ei ole helposti havaittavissa (Savery 2014).

Yksinkertaisin keinotekoista viivettä lisäävä menetelmä on tunnettu *lockstep*-protokollana, jossa pelitilaa voidaan simuloida eteenpäin vasta kun syötteille on saatu vahvistusviesti jokaiselta peliin liittyneeltä asiakkaalta (Lee ym. 2003). Lockstep-protokollasta on kehitetty myös muistipaikkoihin perustuva synkronointimenetelmä (engl. *bucket synchronization*), jossa jokaiselle muistipaikalle on asetettu aikaikkuna jonka umpeuduttua siihen tallennetut syötteet tulevat käsitellyiksi (Bettner ja Terrano 2001; Savery 2014). Muistipaikkoihin perustuvaa synkronointimenetelmää voi pitää kompromissina viiveen ja pelitilan oikeellisuuden välillä.

Tässä luvussa mainitut viiveen kompensointimenetelmät ovat useammin sopivia tiettyihin pelityyppeihin, mutta niitä voi olla järkevää yhdistellä jos pelissä on yhdistelty pelimekaniikkoja useista pelityypeistä. Esimerkiksi laskelmasuunnistus sopii kaikkiin peleihin, joissa olioilla on fysikaalisiin suureisiin perustuvia liikeratoja. Samoin keinotekoista viivettä voidaan soveltaa pelin osa-alueisiin, joissa pelaajien välinen vuorovaikutus on vuoropohjaista.

5 Yhteenveto

Verkkomoninpelien monimutkaistumisen ja teknologisen kehityksen seurauksena pelitilan synkronoinnista on muodostunut 1990-luvun jälkeen uusi tutkimusalue, joka on saanut pelien kasvavan suosion vuoksi paljon huomiota. Monimutkaiset pelitilat ovat liian suuria välitettäväksi sellaisenaan verkkoyhteyden yli pelaajille, joten pelitilan synkronointiin on täytynyt kehittää tehokkaita optimointimenetelmiä. Lisäksi täysin viiveetön pelitilan välitys keskitetyn palvelimen arkkitehtuurissa ei ole fyysisten rajoitusten vuoksi mahdollista, joten pelitilan synkronoinnissa tavoitellaan sopivaa suhdetta pelitilojen yhtenäisyyden ja viiveen välillä.

Viive on edelleen keskeisin verkkomoninpelejä rajoittava tekijä, mutta viiveen kompensointimenetelmät ovat mahdollistaneet verkkomoninpelien pelaamisen myös hitaammilla yhteyksillä. Ennustavat menetelmät voivat aiheuttaa ristiriitoja autoritaarisen palvelimen ja asiakassovelluksen pelitilojen välille, mutta ristiriitojen selvittäminen on keskitetyn palvelimen avulla aina mahdollista. Viiveen kompensointimenetelmiä koskevaa tutkimusta on paljon ja menetelmien monipuolisuudesta voi tehdä johtopäätöksen, että pelitilan synkronointi on mahdollista toteuttaa kaikentyyppisiin verkkomoninpeleihin riippumatta pelien yksilöllisistä pelimekaanikoista.

Tutkielmassa koottiin yhteen yleisimpien pelitilan välitykseen, palvelimen optimointiin ja viiveen kompensointiin liittyvien menetelmien vahvuuksia ja heikkouksia. Tutkimuksesta käy ilmi että pelitilan synkronointiin on mahdollista kehittää loputtomasti uusia menetelmiä, jotka pohjautuvat edelliseen tutkimukseen, pelien yksilöllisiin ominaisuuksiin ja pelaajien toimintamalleihin. Aihetta hyvin tunteva voi nähdä tutkimusalueen kehityksen myös kaupallisissa suljetun lähdekoodin peleissä, mutta tavalliselle pelaajalle hyvin toteutettu pelitilan synkronointi ei yleensä ilmene muuten kuin pelin virheettömyytenä.

Lähteet

Agar, Jacob, Jean-Pierre Corriveau ja Wei Shi. 2012. “Play patterns for path prediction in multiplayer online games”. Teoksessa *7th International Conference on Communications and Networking in China,* 12–17. IEEE.

Armitage, G. 2003. “An experimental estimation of latency sensitivity in multiplayer Quake 3”. Teoksessa *The 11th IEEE International Conference on Networks, 2003. ICON2003.* 137–141. https://doi.org/10.1109/ICON.2003.1266180.

Armitage, Grenville, Mark Claypool ja Philip Branch. 2006. *Networking and online games: understanding and engineering multiplayer Internet games.* John Wiley & Sons.

Armitage, Grenville ja Lawrence Stewart. 2004. “Limitations of using real-world, public servers to estimate jitter tolerance of first person shooter games”. Teoksessa *Proceedings of the 2004 ACM SIGCHI International Conference on Advances in computer entertainment technology,* 257–262.

Bernier, Yahn W. 2001. “Latency compensating methods in client/server in-game protocol design and optimization”. Teoksessa *Game Developers Conference,* nide 98033. 425.

Bettner, Paul ja Mark Terrano. 2001. “1500 archers on a 28.8: Network programming in Age of Empires and beyond”. Teoksessa *GDC,* 2:30. 2001.

Briscoe, Bob, Anna Brunstrom, Andreas Petlund, David Hayes, David Ros, Ing-Jyh Tsang, Stein Gjessing, Gorry Fairhurst, Carsten Griwodz ja Michael Welzl. 2016. “Reducing Internet Latency: A Survey of Techniques and Their Merits”. *IEEE Communications Surveys & Tutorials* 18 (3): 2149–2196. https://doi.org/10.1109/COMST.2014.2375213.

Carmack, John. 1999. “Quake engine GPL release”. Viitattu 2. huhtikuuta 2024. https://github.com/id-Software/Quake.

Cronin, Eric, Burton Filstrup, Anthony R Kurc ja Sugih Jamin. 2002. “An efficient synchronization mechanism for mirrored game architectures”. Teoksessa *Proceedings of the 1st workshop on Network and system support for games,* 67–73.

Feng, Wu-chang, F. Chang, Wu-chi Feng ja J. Walpole. 2005. “A traffic characterization of popular on-line games”. *IEEE/ACM Transactions on Networking* 13 (3): 488–500. https://doi.org/10.1109/TNET.2005.850221.

Ferretti, Stefano. 2014. “Synchronization in multiplayer online games”. Teoksessa *Handbook of Digital Games,* 1st, 175–196. Wiley-IEEE Press. ISBN: 1118328035.

Ferretti, Stefano ja Marco Roccetti. 2005. “Fast delivery of game events with an optimistic synchronization mechanism in massive multiplayer online games”. Teoksessa *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology,* 405–412.

Fiedler, Glenn. 2011. “Floating point determinism”. Viitattu 7. huhtikuuta 2024. https://gafferongames.com/post/floating_point_determinism.

Gregory, Jason. 2014. *Game Engine Architecture, Second Edition.* 2nd. USA: A. K. Peters, Ltd. ISBN: 1466560010.

Hastings, Erin J, Jaruwan Mesit ja Ratan K Guha. 2005. “Optimization of large-scale, real-time simulations by spatial hashing”. Teoksessa *Proc. 2005 Summer Computer Simulation Conference,* 37:9–17. 4. Citeseer.

Heuschkel, Jens, Tobias Hofmann, Thorsten Hollstein ja Joel Kuepper. 2017. *Introduction to Raw-sockets.* Tekninen raportti TUD-CS-2017-0111. Technische Universität Darmstadt, toukokuu. Viitattu 9. huhtikuuta 2024. https://tuprints.ulb.tu-darmstadt.de/6243/1/TR-18.pdf.

Jiang, Xinbo, Farzad Safaei ja Paul Boustead. 2005. “Latency and scalability: a survey of issues and techniques for supporting networked games”. Teoksessa *2005 13th IEEE International Conference on Networks Jointly held with the 2005 IEEE 7th Malaysia International Conf on Communic,* nide 1, 6–pp. IEEE.

Kaiser, Arnaud, Dario Maggiorini, Nadjib Achir ja Khaled Boussetta. 2009. “On the Objective Evaluation of Real-Time Networked Games”. Teoksessa *GLOBECOM 2009 - 2009 IEEE Global Telecommunications Conference,* 1–5. https://doi.org/10.1109/GLOCOM.2009.5426032.

Kushner, David. 2003. *Masters of Doom: How Two Guys Created an Empire and Transformed Pop Culture.* USA: Random House Inc. ISBN: 1588362892.

Lee, Ho, Eric Kozlowski, Scott Lenker ja Sugih Jamin. 2003. “Multiplayer Game Cheating Prevention with Pipelined Lockstep Protocol”. Teoksessa *Entertainment Computing: Technologies and Application,* toimittanut Ryohei Nakatsu ja Junichi Hoshino, 31–39. Boston, MA: Springer US. ISBN: 978-0-387-35660-0. https://doi.org/10.1007/978-0-387-35660-0_4. https://doi.org/10.1007/978-0-387-35660-0_4.

Liu, Shengmei, Xiaokun Xu ja Mark Claypool. 2022. “A Survey and Taxonomy of Latency Compensation Techniques for Network Computer Games”. *ACM Comput. Surv.* (New York, NY, USA) 54 (11s). ISSN: 0360-0300. https://doi.org/10.1145/3519023. https://doi.org/10.1145/3519023.

Murphy, Curtiss. 2011. “Believable Dead Reckoning for Networked Games”. Teoksessa *Game Engine Gems 2,* 1st, 307–328. USA: A. K. Peters, Ltd. ISBN: 1568814372.

Oliveira, Manuel ja Tristan Henderson. 2003. “What online gamers really think of the Internet?” Teoksessa *Proceedings of the 2nd workshop on Network and system support for games,* 185–193.

Pantel, Lothar ja Lars C. Wolf. 2002. “On the suitability of dead reckoning schemes for games”. Teoksessa *Proceedings of the 1st Workshop on Network and System Support for Games,* 79–84. NetGames ’02. Braunschweig, Germany: Association for Computing Machinery. ISBN: 1581134932. https://doi.org/10.1145/566500.566512. https://doi.org/10.1145/566500.566512.

Raaen, Kjetil ja Tor-Morten Grønli. 2014. “Latency thresholds for usability in games: A survey”. Teoksessa *Norsk IKT-konferanse for forskning og utdanning.*

Ravindran, Kaliappa, Ali Sabbir ja Balachandran Ravindran. 2008. “Impact of Network Loss/Delay Characteristics on Consistency Control in Real-Time Multi-Player Games”. Teoksessa *2008 5th IEEE Consumer Communications and Networking Conference,* 1128–1133. https://doi.org/10.1109/ccnc08.2007.254.

Roccetti, Marco, Stefano Ferretti ja Claudio E Palazzi. 2008. “The brave new world of multiplayer online games: Synchronization issues with smart solutions”. Teoksessa *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC),* 587–592. IEEE.

Sanglard, Fabien. 2009. “Quake Engine code review”. Viitattu 28. maaliskuuta 2024. https://fabiensanglard.net/quakeSource/quakeSourceNetWork.php.

Savery, Cheryl. 2014. “Consistency maintenance in networked games”. Tohtorinväitöskirja, Queen’s University at Kingston, Ontario, Canada.

Shi, Wei, Jean-Pierre Corriveau ja Jacob Agar. 2014. “Dead reckoning using play patterns in a simple 2D multiplayer online game”. *International Journal of Computer Games Technology* 2014:5–5.

Singhal, Sandeep Kishan. 1996. “Effective remote modeling in large-scale distributed simulation and visualization environments”. Tohtorinväitöskirja, Stanford University.

Smed, Jouni ja Harri Hakonen. 2017. *Algorithms and Networking for Computer Games.* 2nd. Wiley Publishing. ISBN: 1119259762.

Stefyn, D., A.L. Cricenti ja P.A. Branch. 2011. *Quake III Arena Game Structures.* Tekninen raportti 110209A. Melbourne, Australia: Centre for Advanced Internet Architectures, Swinburne University of Technology, helmikuu. Viitattu 9. huhtikuuta 2024. http://caia.swin.edu.au/reports/110209A/CAIA-TR-110209A.pdf.

Welch. 1984. “A Technique for High-Performance Data Compression”. *Computer* 17 (6): 8–19. https://doi.org/10.1109/MC.1984.1659158.

Zhou, Suiping, Wentong Cai, Stephen J Turner, Bu-Sung Lee ja Junhu Wei. 2007. “Critical causal order of events in distributed virtual environments”. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)* 3 (3): 15–es.