OULUN AMMATTIKORKEAKOULU

Tuomas Keränen

MONINPELIN TOTEUTUS PHOTONILLA

MONINPELIN TOTEUTUS PHOTONILLA

Tuomas Keränen
Opinnäytetyö
Kevät 2016
Tietotekniikan koulutusohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan koulutusohjelma, ohjelmistokehityksen sv

Tekijä: Tuomas Keränen
Opinnäytetyön nimi: Moninpelin toteutus Photonilla
Työn ohjaaja: Lasse Haverinen
Työn valmistumislukukausi ja -vuosi: Kevät 2016 Sivumäärä: 34

Työn tavoitteena oli toteuttaa kaksi erillistä moninpeliä. Ensimmäinen toteutus oli jaetun näytön moninpeli ja toinen toteutus verkkomoninpeli.

Kehityksessä käytettiin Unity 3D -kehitysympäristöä. Jaetun näytön toteutuksessa näyttötila jaettiin kahteen osaan näyttämään molemmille pelaajille omaa pelihahmoa. Jaetun näytön toteutuksessa ei käytetty mitään Unityn ulkopuolisia työkaluja vaan kaikki toteutettiin Unityn perusominaisuuksilla. Verkkomoninpelissä Unityyn ladattiin Photon Unity Networking -lisäosa, jonka avulla verkkopelaamisen ominaisuudet saatiin toteutettua. Molemmissa toteutuksissa käytettiin C#-ohjelmointikieltä.

Molempien toteutuksien tavoitteet saatiin täytettyä, vaikkakaan kumpaakaan peliä ei saatu julkaisukelpoiseksi. Lopputulos, erityisesti verkkomoninpelistä, toimii hyvänä pohjana jatkokehitykselle.

Asiasanat: Unity, Photon, verkkopelit, pelikehitys

SISÄLLYS

SANASTO

Collider	Collider määrittää objektin rajat fyysisiä törmäyksiä varten
Coroutine	Funktio, joka voi pysäyttää suorituksensa määritetyn mittaiseksi ajaksi
Master Server	Pääpalvelin, suurikokoinen palvelinkokonaisuus joka huolehtii pelaajien yhdistämisen pelipalvelimelle
Pelipalvelin	Pienempi palvelin, jonka kautta pelin dataliikenne tapahtuu
PhotonEngine	Saksalaisen Exit Games -yhtiön tarjoama lisäosa verkkopelien toteutukseen
Photon Cloud	Photonin pilvipalveluna toimiva palvelinkokonaisuus, joka koostuu viidestä suuresta pääpalvelimesta
Prefab	Peliobjekteista koostuva myöhemmin uudelleenkäytettävä elementti
Renderöinti	Data muunnetaan näytölle esitettäväksi sopivaan muotoon
Transform	Unityssä oleva komponentti, joka sisältää esimerkiksi peliobjektin paikkatiedon

1 JOHDANTO

Tämän opinnäytetyön tarkoituksena oli tutustua moninpelin tuottamiseen Unity-pelimoottorilla. Alkuperäisesti tarkoituksena oli tehdä vain verkkopohjaista tekniikkaa käyttävä peli, mutta työn edetessä aihe hieman laajeni ja mukaan tuli paikallisesti pelattavan moninpelin toteutus.

Pelimoottoriksi valittiin Unity, koska henkilökohtaisesti minulla oli siitä työtä aloittaessa eniten kokemusta. Unity toimii erittäin hyvin sekä aloittelevan että kokeneemman pelikehittäjän työkaluna. Unityn hyviä puolia ovat muun muassa alustariippumattomuus, usean koodikielen tuki ja helppo laajennettavuus.

Paikallisesti pelattava moninpeli tehtiin jaetun näytön toteutuksena, koska se sopi parhaiten peli-idean kanssa yhteen. Verkkopeliin työkaluksi valittiin Unityn lisäksi PhotonEngine, erityisesti Photon Unity Networking, koska se toimii saumattomasti yhteen juuri Unityn kanssa. PhotonEngine tarjoaa pelinkehittäjälle paljon valmiita verkkokomponentteja sekä valmiin palvelinrakenteen, jolloin kehittäjien ei tarvitse keskittyä palvelimien ongelmiin.

Tavoitteena työssä oli oppia mahdollisimman monipuolisesti moninpelien toteuttamisesta. Erityisiä kiinnostuksen kohteita oli dataliikenne palvelimen ja pelaajien välillä.

2 MONINPELIMAHDOLLISUUDET

2.1 Moninpelaamisen mahdollistavat tekniikat

Moninpelaamisella käsitetään kaikki useamman kuin yhden henkilön pelaaminen. Jotta videopeli on monen ihmisen pelattava, täytyy sitä tehtäessä käyttää jotain tekniikkaa, joka mahdollistaa usean henkilön yhtäaikaisen osallistumisen. Tässä työssä käytetään kahta eri tekniikkaa, jaetun ruudun tekniikkaa sekä verkkopohjaista ratkaisua.

2.2 Jaetun näytön moninpeli

Jaetun näytön moninpelitilaa on käytetty peliteollisuudessa jo vuosikymmenten ajan. Esimerkiksi yksi ensimmäisistä kaupallisista peleistä, Pong (Atari, 1972), käytti jaettua näyttöä moninpelaamisen mahdollistamiseen. Myöhemmin jaetun näytön käyttö peleissä on ollut vaihtelevan suosittua. Tietokonepeleissä jaetun näytön tila ei ole nykyään enää niin suosittu ominaisuus kuin aikaisemmin tai konsolipeleissä. Konsolipelien valmistajatkin alkavat luopua tästä ominaisuudesta siirtymällä internetin välityksellä tapahtuvaan moninpelaamiseen.

Jaetun näytön moninpelissä käytössä oleva pelinäyttö jaetaan useamman pelaajan käyttöön. Tämä mahdollistaa sen, että useampi henkilö voi osallistua samaan peliin yhdellä laitteella. Pelikehityksessä jaetun näytön pelaaminen aiheuttaa normaaliin yhden pelaajan peliin verrattuna lisää toimenpiteitä. Avainte-kijänä tässä on näyttötilan jakaminen useamman pelaajan käyttöön. Tämä voidaan tehdä useammalla eri tavalla.

Pelin näyttöalue voidaan jakaa useampaan osaan, tämä toteutetaan antamalla jokaiselle pelihahmolle oma kamera-objekti. Kamera renderöi sen näkymässä olevat, pelissä käytettävät peliobjektit näytölle. Jotta useamman pelaajan näkymät saadaan mahdutettua yhdelle näytölle, täytyy jokaisen kameran viewporttia eli aluetta, minkä kokoisena kameran renderöimä kuva näkyy pelinäytöllä,

muuttaa. Tätä keinoa käytettäessä tulee ottaa huomioon se, että jokaisen pelaajan pelikuvan renderöinti kuluttaa aina enemmän resursseja. Siksi neljän pelaajan pelissä ei voi esimerkiksi käyttää kaikkia graafisia erikoistehosteita, joita yksin- tai kaksinpelissä voisi käyttää. Hyvänä puolena tätä keinoa käytettäessä on se, että pelaajien ei tarvitse sijoittua pelimaailmassa lähelle toisiaan, koska jokaisella hahmolla on käytössään oma kameraobjekti (kuva 1). (1.)

KUVA 1. Neljän pelaajan jaetun ruudun moninpeli pelissä Mario Kart 8 (2)

Toinen vaihtoehto on käyttää pelissä yhtä kameraa, joka näyttää kaikkien pelaajien pelikuvan yhdessä viewport-ikkunassa. Tämä keino vähentää renderöinnin määrää, mutta peli tulee suunnitella niin, että jokainen pelaaja mahtuu yhden kameran kuvaan (kuva 2).

KUVA 2. Neljä pelaajaa, yksi kamera. ”Magicka 2” (3)

Jaetun ruudun moninpeli tuo mukavan lisämausteen pelaamiseen. Pelaajien ollessa samassa tilassa voi pelatessa nähdä toisten reaktiot pelin tapahtumiin ja sosiaalinen kanssakäyminen on helppoa. Toisen pelaajan näkymän vakoilu saattaa pelistä riippuen joko olla hauskaa tai haitata pelikokemusta, mutta se kuuluu asiaan.

2.3 Verkkopohjainen moninpelaaminen

Verkkopohjainen moninpelaaminen eli verkkopelaaminen on internetin tai paikallisen LAN-verkon välityksellä tapahtuvaa pelaamista. Verkkopelaaminen mahdollistaa monen ihmisen osallistumisen yhteen peliin useammalla laitteella, eikä pelaajien tarvitse olla fyysisesti lähekkäin toisiaan.

Verkkopelaaminen on nykyään suosittua, jopa niin suosittua että siitä on muodostunut ammattitason urheilulaji. E-urheilu eli elektroninen urheilu kerää miljoonayleisön internetin välityksellä seurattaviin suoriin lähetyksiin (4).

2.3.1 Palvelinmalleja

Verkkopelejä pelatessa on käytössä aina jonkinlainen palvelin, serveri. Palvelimet ovat tyypiltään joko erillisiä, peliprosessin kanssa ajettavia tai peer-to-peer-tyyppisiä.

Erillinen palvelin (englanniksi dedicated server) on pelin ulkopuolella toimiva palvelin. Yleensä ne ovat yritysten ylläpitämissä datakeskuksissa, joissa palvelinkäyttöön erikseen tarkoitetuilla tietokoneilla ajetaan suuria määriä palvelimia. Pelaajien laitteiden ulkopuolella olevan palvelimen käyttö säästää yleensä verkkoviiveen kasvamiselta. Erillisten palvelimien ylläpito maksaa, joten ne ovat yleensä vuokrattavia tai ostettavia omaan käyttöön. Tästä syystä peleissä yleensä tarjotaan mahdollisuus ajaa omaa palvelinta pelin ohella. (5.)

Peliprosessin ohella ajettavat palvelimet ovat maksuttomia. Ne toimivat samalla periaatteella kuin erilliset palvelimet, mutta niitä ajetaan jonkun pelaajan koneella peliprosessin ohella. Ongelmaksi itse ajettavalla palvelimella muodostuu verkkoviiveen kasvaminen, koska yksityisihmisille myytävissä nettiliittymissä on harvoin tarjolla niin suuria verkkonopeuksia, kuin palvelimen ajamiseen olisi tarve. Tästä johtuen pelaajien määrä tämän tyyppisellä palvelimella on yleensä rajoitettu. Hyvänä puolena voidaan todeta ilmaisuus, sekä erillisen laitteiston puuttuminen. (5.)

Peer-to-Peer-tyyppinen palvelinratkaisu poikkeaa muista siten, että verkon yli lähetettävä tieto kulkee suoraan pelaajien välillä, eikä välissä ole mitään palvelinta. Huonona puolena nähdään heti, se että jokainen pelaaja lähettää datan jokaiselle pelissä olevalle pelaajalle, mikä rajoittaa pelaajien maksimimäärää. Tämän lisäksi viive pelaajien välillä aiheuttaa ongelmia pelitilanteiden pitämisessä synkronoituina. (5.)

2.3.2 Verkkoviive ja sen haittavaikutukset

Verkkopohjaisessa moninpelaamisessa yksi perustavanlaatuinen haaste on verkkoviive. Verkkoviive eli latenssi tarkoittaa aikaa, joka kuluu yksittäisellä paketilla siirtyä lähettäjältä verkon yli vastaanottajalle ja saapua vastaanottamiskuittauksena takaisin lähettäjälle. Pienempi verkkoviive on aina parempi kuin suuri, koska liian iso viive tarkoittaa esimerkiksi pelien kohdalla paikkatietojen siirtymisen viivästymistä, mikä näkyy pelissä viiveellä tapahtuvana liikkumisena. Erittäin hyväksi verkkoviiveeksi voidaan laskea kaikki alle 50 millisekunnin viiveet. Viiveen ollessa 90 millisekunnin paikkeilla saattaa verkkopelejä pelatessa esiintyä virheitä. Verkkoviiveen ollessa 150 tai yli alkaa peleissä ilmetä suurempia ongelmia ja muutkin verkkopohjaiset palvelut kärsivät. (6.) Verkkoviiveeseen vaikuttavat muun muassa käytössä oleva internetyhteys ja palvelimen fyysinen etäisyys käyttäjästä.

Verkkoviiveen vaikutus moninpeleihin riippuu hieman pelin tyylistä. Esimerkiksi nopeatempoiset ensimmäisen persoonan ammuntapelit (FPS-pelit, First Person Shooter) ja MOBA-pelit (Multiplayer Online Battle Arena) kärsivät suuresta verkkoviiveestä, kun taas hitaampitempoisempiin verkkopeleihin viive ei juurikaan vaikuta.

Otetaan esimerkiksi yksi suosituimmista FPS-peleistä, Counter Strike: Global Offensive. Pelin kehittänyt Valve on kertonut omilla sivustoillaan, mitä erilaisia keinoja on käytetty verkkoviiveen haittavaikutuksien eliminoimiseksi pelissään. Counter Strike -pelissä käytetään erillisiä palvelimia, eikä dataliikenne tapahdu suoraan pelaajien välillä. Palvelin simuloi peliä tietyn ajan välein. Tästä tapahtumasta käytetään termiä ”tick”, suomeksi käännettynä ”pelipäivitys”. Jokaisella pelipäivityksellä palvelin vastaanottaa pelaajien tekemät komennot ja päivittää pelimaailman jokaiselle pelaajalle nykyiseen tilaan. Counter Strike -pelissä pelipäivityksien välinen aika on vakiona 15 millisekuntia eli yhden sekunnin aikana tapahtuu 66,66 pelipäivitystä. Mitä pienempi pelipäivityksien väli eli mitä enemmän pelipäivityksiä tapahtuu sekunnissa, sitä tarkemmin pelaajien liikkeet ja pe-

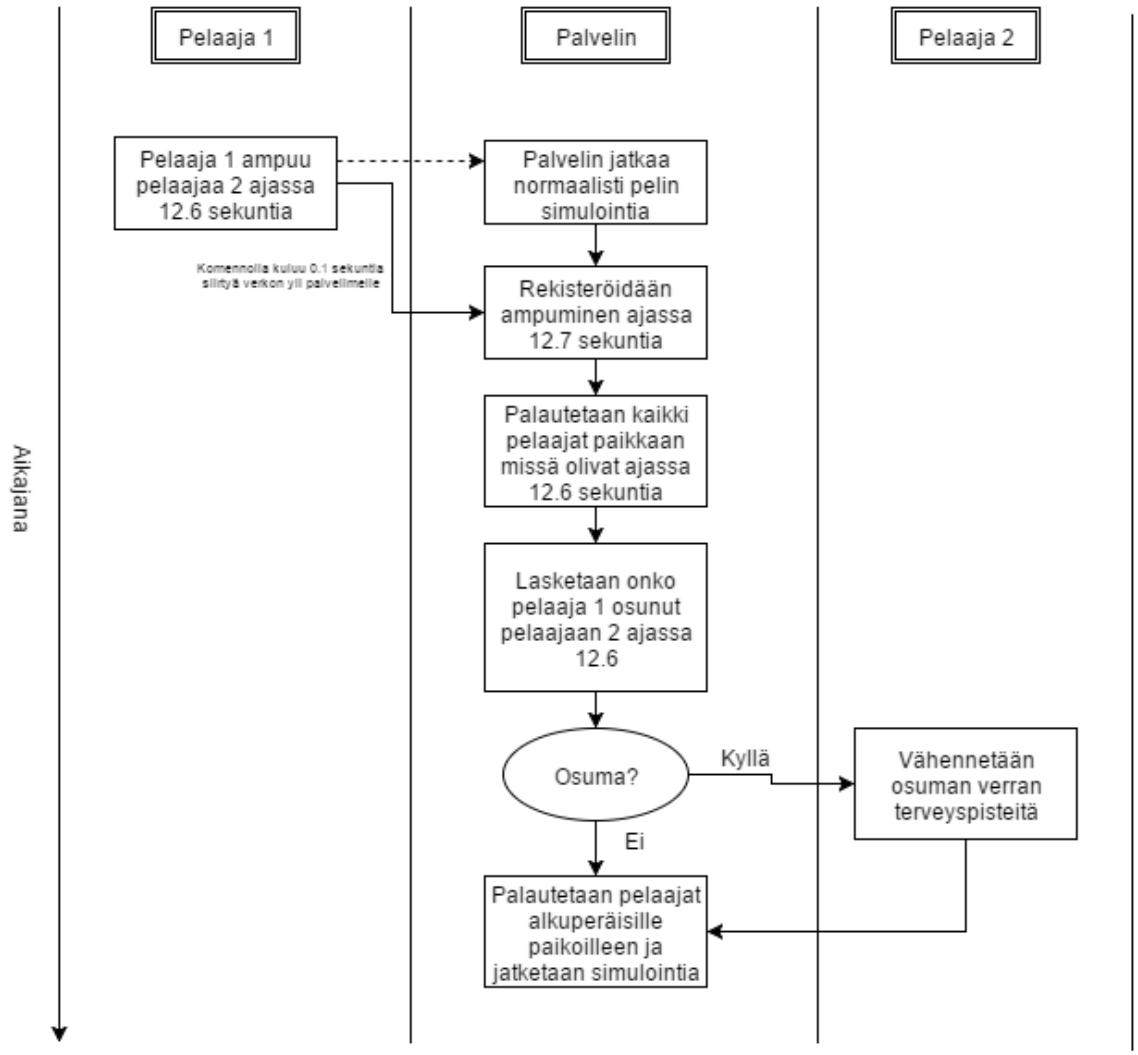
lin tapahtumat simuloituvat pelaajille. Pelipäivityksien määrää ei voida kuitenkaan kasvattaa liian suureksi, koska mitä enemmän niitä tapahtuu, sitä enemmän peli vaatii suoritintehoa ja verkon kaistaleveyttä sekä palvelimelta, että pelaajan laitteelta. (7.)

Pelaajilla saattaa olla käytössään erittäin suuresti eroavia verkon kaistaleveyksiä. Tämä vaikuttaa siihen, kuinka paljon palvelin voi lähettää yksittäiselle pelaajalle dataa, koska jos yhden pelaajan yhteys on 6 kilotavua sekunnissa, ei hän voi ottaa vastaan suuria datamääriä ilman datahävikkiä. Tähän ongelmaan on keksitty ratkaisu vaatimalla jokaiselta pelaajalta päivitystaso-arvo (”update-rate”), joka määrittää, kuinka monta kertaa pelaaja voi vastaanottaa pelimaailman tilanteen sekunnissa (vakiona Counter Strike -pelissä 20). Tämä arvo ei voi olla kuitenkaan suurempi kuin palvelimen pelipäivityksien määrä sekunnissa. Counter Strike -pelissä verkon yli kulkeva data delta-kompressoidaan eli pakataan, jotta saadaan vähennettyä verkon rasitusta. Tämä tarkoittaa käytännössä sitä, ettei palvelin joudu jokaisella päivityskerralla lähettämään koko pelin tilannetta, vaan ainoastaan pelissä tapahtuneet muutokset edelliseen tilanteeseen verrattuna. (7.)

Palvelin siis lähettää vakioarvoilla 20 tilannepäivitystä sekunnissa pelaajalle. Jos peliobjektit päivitettäisiin siihen tilanteeseen, missä ne näkyvät palvelimella, olisivat peliobjektien animaatiot ja liikkuminen pelaajan näytöllä pätkiviä. Ongelman ratkaisuksi on käytetty interpolaatiota. Pelaajan vastaanottaessa 20 tilannepäivitystä sekunnissa eli päivityksen 50 millisekunnin välein voidaan pelaajan renderöintiajankohtaa taaksepäin siirtämällä (interpolaatioaika) saada peliobjektien liikehdintä sulavaksi. Jos interpolaatioaika on 100 millisekuntia, pelaaja voi renderöidä kahden palvelimelta saadun tilannepäivityksen pohjalta peliobjektien liikehdinnän oikein. Interpolaatio näkyy pelissä 100 millisekunnin viiveenä. (7.)

Peliin tulee siis monen asian summana paljon viivettä, joka aiheuttaa ongelmatilanteita, kun yritetään osua aseella toiseen pelaajaan. Counter Strike -pelissä ratkaisuksi on käytetty viiveen kompensointia. Esimerkiksi pelaajat ampuvat toi-

siaan kohti ajassa 12,6 sekuntia pelaajan oman pelin aikaa. Data tästä tapahtumasta lähetetään palvelimelle ja samalla, kun datapaketti on siirtymässä verkon yli, palvelin jatkaa pelimaailman simulointia. Tässä ajassa pelaajat saattavat liikkua eri sijainteihin, kuin missä olivat hetki sitten. Paketti saapuu palvelimelle ajassa 12,7. Palvelin ei tunnista pelaajan osumaa toiseen pelaajaan, vaikka tähtäys olisikin ollut täydellinen. Viiveen kompensoinnin järjestelmä pitää tallessa jokaisen pelaajan paikkatiedot yhden sekunnin ajalta. Palvelimen vastaanottaessa pelaajalta ampumiskäskyn, siirretään kaikki pelaajat niille paikoille missä he olivat ampumishetkellä (lisäksi ottaen huomioon verkkoviiveen ja interpolaation). Näin palvelin voi tarkistaa, onko osuma oikeasti tapahtunut, ja palauttaa sen jälkeen pelaajat alkuperäisille paikoilleen. (Kuva 3.) (7.)



KUVA 3. Aikajana ampumistapahtumasta pelissä Counter Strike

2.4 Photon

Photon on saksalaisen Exit Gamesin vuonna 2003 julkaisema alustariippumaton työkalu moninpelien kehittämiseen. Photonia on mahdollista käyttää tällä hetkellä noin 15 kehitysympäristössä, joihin kuuluu esimerkiksi Unity, iOS ja Android. Photon ei siis ole itsessään pelien kehittämistä mahdollistava pelimoottori, vaan lisäosa mikä on tehty käytettäväksi suosituimmilla pelinkehitystyökaluilla. (8.)

2.4.1 Keskeiset ominaisuudet

Photon on erittäin hyvä kokonaisuus verkkopelien kehittämiseen. Exit Games on tehnyt paljon töitä saadakseen verkkopelien kehittämisen kynnyksen pienemmäksi uusille sekä kokeneemmille pelinkehittäjille. Verkkopelien kehittämisessä tarvitaan kuitenkin ymmärrystä verkon toiminnasta sekä käytettävästä tekniikasta, mikä saattaa olla liian iso pala haukattavaksi pelinkehitystiimeille. Photonissa verkkoteknistä osaamista ja paljon työtä vaativat komennot on yksinkertaistettu niin, että hieman kokemattomampikin kehittäjä osaa käyttää niitä. Toisaalta asian huonona puolena voidaan nähdä se, että kehittäjät eivät välttämättä tiedä mitä Photon pohjimmillaan tekee.

Hyvää Photonissa on helppo käyttöönotto. Työkalun kotisivuilta löytyy erikseen jokaiselle kehitysympäristölle räätälöity paketti ohjelmaesimerkkeineen ja dokumentaatioineen. Omakohtainen kokemus rajoittuu Photon Unity Networking -versioon, joka oli äärimmäisen helppo ottaa käyttöön, mutta uskon muillakin kehitysalustoilla asioiden hoituvan helposti. Jatkokehityksen kannalta Photon tarjoaa hyvät puitteet, koska lähdekoodi on täysin tarjolla ja sitä voi muokata vastaamaan omia tarpeita.

Aiemmin mainittu verkkoviive on otettu huomioon Photonia kehitettäessä. Viiveen minimoimiseksi on kehitetty Photon Cloud, joka on viidellä pääpalvelimella (Master Server) toimiva pilvipalvelukokonaisuus. Pääpalvelimet on sijoitettu keskeisiin paikkoihin lähelle suuria käyttäjäkuntia (kuva 4).

KUVA 4. Photon pääpalvelimet kartalla esitettynä (9)

Pääpalvelimen valinta vaikuttaa siis olennaisesti verkkoviiveeseen, joka vaikuttaa taas pelikokemukseen. Kuten aiemmin todettiin, mitä pienempi viive, sitä sujuvampaa on pelaaminen. Pääpalvelimen valintaan on mahdollista käyttää muutamaa eri tekniikkaa. Yksi keino on antaa Photonin itsensä etsiä pienimmän viiveen saavuttava pääpalvelin. Toisena keinona peliin voidaan kehittää jokaiselle viidestä alueesta oma erillinen versio pelistä, joka yhdistää automaattisesti alueen pääpalvelimeen. Kolmantena vaihtoehtona peliin tehdään valikko, josta pelaaja saa itse valita pääpalvelimen johon otetaan yhteys. Näiden vaihtoehtojen lisäksi on mahdollista laittaa kaikki pelaajat yhdistämään yhteen pääpalvelimeen. Tätä vaihtoehtoa käytettäessä tulee muistaa, että verkkoviive kasvaa pääpalvelimesta kauempana olevilla pelaajilla. Tästä ei pitäisi koitua suurta haittaa, jos pelissä ei ole paljoa reaaliaikaisesti päivittyviä elementtejä.

2.4.2 Photonin eri versiot ja hinnoittelu

Photonista on tarjolla erilaisia versioita, jotka on räätälöity palvelemaan asiakkaiden erilaisia tarpeita. Tällä hetkellä tarjolla on neljä erilaista pakettia: Realtime, Turnbased, Bolt ja Photon Unity Networking. Näiden lisäksi on tarjolla vielä erillinen Photon Chat, joka sisältää valmiin paketin viestipohjaiseen keskustelemiseen pelin aikana.

Photon Realtime on tehty erityisesti pelihuoneisiin pohjautuvia pelejä varten. Pelihuone, 2-60 pelaajan kokoinen yhtä pelisessiota varten luotu palvelin, on yleisin ratkaisu jos pelin kaikkien pelaajien ei haluta olevan yksittäisessä pelisessiossa. Realtime tarjoaa myös valmiita työkaluja matchmakingiin, eli pelisessioiden järjestämiseen tietynlaisilla parametreilla, esimerkiksi pelaajien taitotason mukaan. Alustariippumattomuus on yksi Photonin myyntivalteista ja sisältyy oletettavasti myös Realtimeen. Alustariippumattomuus tarkoittaa käytännössä sitä, että esimerkiksi Android- ja iOS-pelaajat voivat osallistua samoihin pelisessioihin. Realtime käyttää viestintään Photon Cloudia, jota käsitellään myöhemmin hieman tarkemmin.

Photon Turnbased sisältää kaiken minkä Realtime-versiokin, mutta sen lisäksi tarjoaa vielä mahdollisuuden vaihtaa synkronisen ja asynkronisen päivittämisen välillä. Synkroninen päivittäminen tarkoittaa tietyn syklin välein tapahtuvaa päivittymistä ja asynkroninen päivittäminen tapahtuu taas synkronointisignaalien tahdissa. Tämä helpottaa vuoropohjaisissa peleissä, koska vuorojen pituudet eivät ole välttämättä aina samanmittaisia.

Bolt Engine on ruotsalaisen Fredrik Holmströmin kehittämä Unityn lisäosa verkkopelien kehittämiseen. Holmström liittyi virallisesti osaksi Photonin kehitystiimiä 15.5.2015, jolloin Bolt Engine siirtyi osaksi Photon-tuoteperhettä ja nimeksi vaihtui samalla Photon Bolt (10). Photon Bolt on tällä hetkellä beeta-vaiheessa oleva paketti Unity-pelimoottorille. Bolt eroaa teknisesti muista Photon-paketeista ja sisältää myös paljon ominaisuuksia, mitä muista versioita ei löydy. Boltissa ei muun muassa käytetä Photon Cloudia pelien väliseen viestimiseen, vaan viestintä tapahtuu pelin ohella ajettavan palvelimen kautta. Muita pelikehitykseen vaikuttavia eroja muihin versioihin nähden ovat muun muassa Unityn Mecanim-animaatioiden automaattinen päivittyminen verkon yli kaikille pelaajille, peliobjektien paikkatiedot sisältävien transform-komponenttien automaattinen päivittyminen kaikille pelaajille ja liikkeiden varmentamisen palvelimella. Liikkeiden varmentaminen palvelimella on asetelma, missä pelaaja lähettää pe-

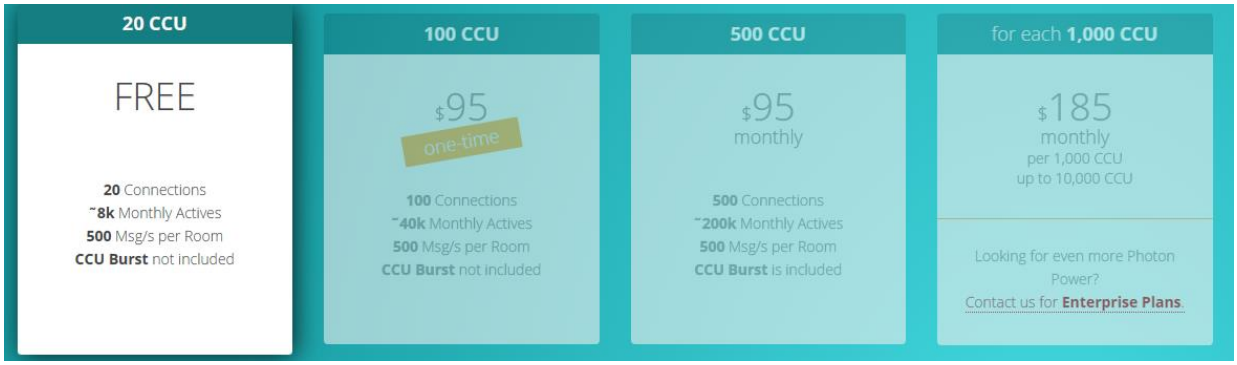
lidataa palvelimelle, joka varmentaa datan aidoksi ja lähettää sen edelleen jokaiselle käyttäjälle. Tämä tarkoittaa sitä, että jos käyttäjä yrittää esimerkiksi huijata pelissä liikkumalla nopeampaa kuin olisi tarkoitus, palvelin rajoittaa pelaajan liikkeen normaalinopeuksiseksi, eikä huijaaminen näin onnistu.

Photon Unity Networking, myöhemmin lyhennettynä PUN, on nimensä mukaisesti Unityyn tehty lisäosa. Paketti on ladattavissa Unityn sisäisestä kaupasta Unity Asset Storesta. Unity sisältää itsessään verkkopelien kehitykseen tarkoitetun Unity Networkingin, mutta pelikehittäjän tulee sitä käytettäessä osata itse tehdä kaikki verkkoliikennettä koskeva koodaus ja palvelinlogiikka. Unity Networkingissä palvelin toimii yhden pelaajan laitteella, kun taas PUN:ssä käytetään erillisiä palvelimia. Tämän ansiosta pelisessio ei pääty, jos peliä isännöivä pelaaja päättää poistua pelistä. PUN:stä on olemassa ilmainen, sekä maksullinen versio. Ilmaisessa versiossa alustana on ainoastaan PC, kun taas maksullisessa versiossa voidaan pelejä tehdä myös mobiilialustoille.

Photon Chat on erillinen lisäpalvelu viestipohjaista keskustelua varten. Chat sisältää valmiiksi muun muassa valmiit komponentit keskusteluhuoneiden luomiseen ja viestien tallentamisen viestipuskuriin.

Kaikkien muiden Photon-tuotteiden hinnoittelu, paitsi Photon Boltin, pohjautuu yhtä aikaa palvelua käyttävien pelaajien määrään (CCU, Concurrent Users), eikä esimerkiksi kaiken kaikkiaan olevien pelaajien määrään. Kaikista paitsi Bolt-versiosta löytyy myös ilmainen vaihtoehto, joka on tarkoitettu lähinnä pelin kehitysvaiheeseen ja testaukseen. Ilmaisessa versiossa yhtäaikaisien käyttäjien määrä on rajoitettu 20:een. Maksullisista vaihtoehdoista ainoa kiinteällä hinnalla oleva on 100 yhtäaikaista käyttäjää salliva, 95 dollaria maksava vaihtoehto. Loput vaihtoehdot ovat kuukausimaksullisia, joiden hinta määräytyy yhtäaikaisten käyttäjämäärien mukaan. Koska kehitysvaiheessa olevan pelin tulevien pelaajien määrää on vaikea ennustaa, Photon mahdollistaa maksusuunnitelman vaihdon milloin vain. Kalleimmassa, tuhansien käyttäjien mukaan skaalautuvassa paketissa on myös tarjolla ominaisuus, joka mahdollistaa hetkellisesti maksimimäärän ylittävät käyttäjät. Muissa vaihtoehdoissa käyttäjämäärä on rajoitettu,

eikä ylityksiä sallita. Photon Bolt on mahdollista ostaa Unity Asset Storesta kiinteään hintaan 80 dollaria (kuva 5). (11.)

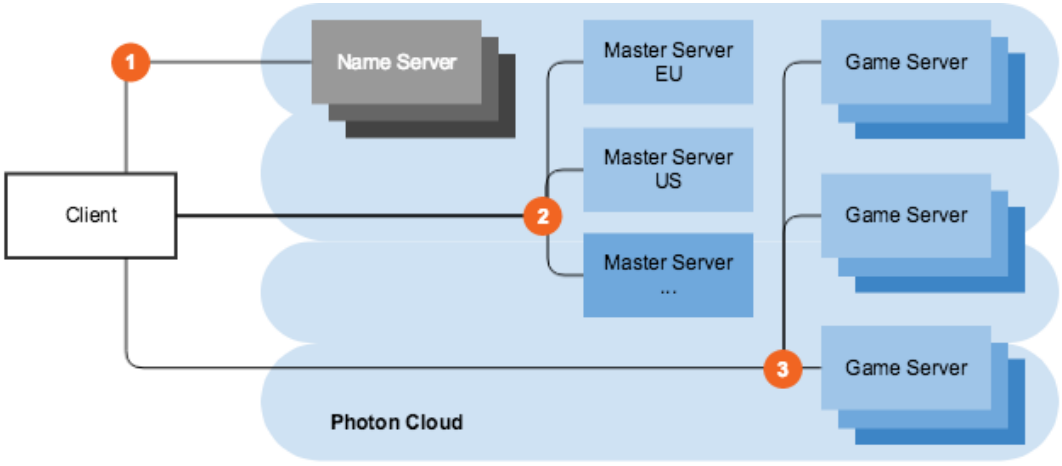


KUVA 5. Kuvankaappaus hinnoittelusta (11)

2.4.3 Photon palvelimet

Photon tarjoaa kaksi palvelinpuolen ratkaisua peleille, Photon Cloud sekä Photon Server, joista Photon Cloud on tehty Photon Serverin pohjalle.

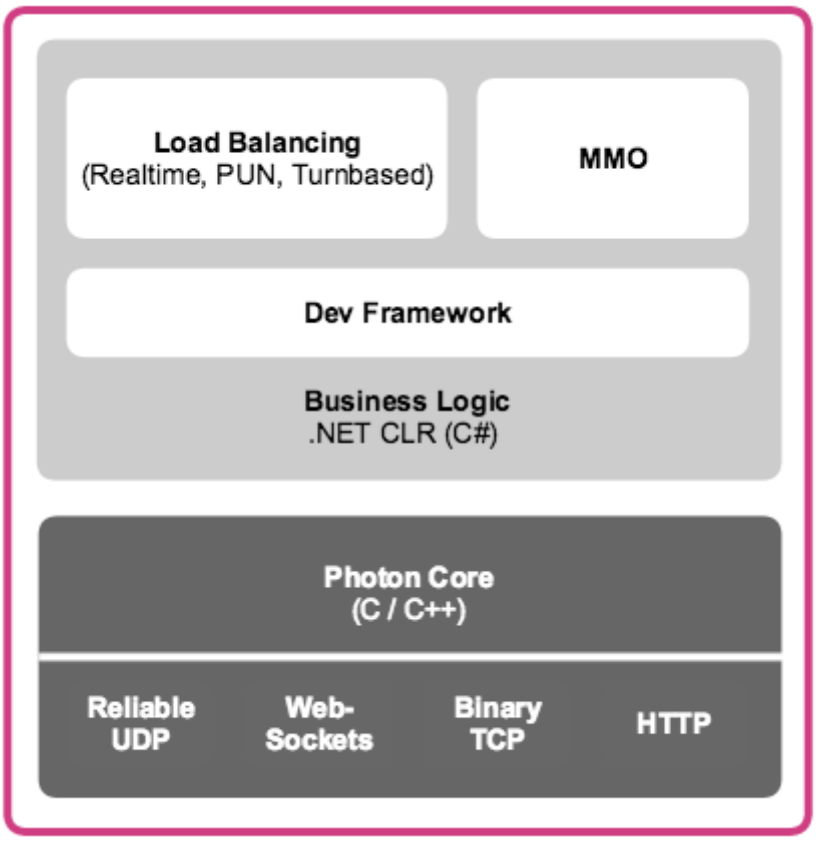
Photon Cloud on valmis palvelu, joka kuuluu kaikkiin versioihin paitsi Photon Boltiin automaattisesti mukaan. Cloud on käyttäjämäärien mukaan skaalautuva palvelinpilvi, jota ylläpitää Exit Games (kuva 6). Asiakkaan päätyessä käyttämään Cloudia, ei palvelinpuolelle tarvitse tehdä itse mitään koodauksia, vaan Exit Games huolehtii itse, että asiakkaiden käytössä on täysin valmis palvelu. Cloud sisältää kaikki tarpeelliset palvelinpuolen toiminnallisuudet, eikä kehittäjän tarvitse omistaa lainkaan palvelinkalustoa tai ostaa Photonin lisäksi muuta erillistä palvelua.



KUVA 6. Photon Cloud -rakennetta (12)

Photon Server on vaihtoehtoinen palvelu Photon Cloudin tilalle. Se tarjoaa työkalut ja ympäristön, joilla voi omalla laitteistolla ylläpitää moninpelaamiseen tarkoitettua palvelinta. Server on, kuten muutkin Photon-palvelut Boltia lukuun ottamatta alustariippumaton ratkaisu, joten yhdellä palvelimella voi asioida kaikilla tuetuilla laitealustoilla. Serverin hinnoittelu myötäilee pitkälti samaa linjaa kuin Photonin muutkin paketit. Erona kuukausimaksullisen järjestelmän lisäksi on kerralla 60 kuukaudeksi ostettava kiinteällä summalla hinnoiteltu paketti. Esimerkiksi 500 yhtäaikaisen käyttäjän kuukausimaksullinen paketti maksaa 25 dollaria kuukaudessa, kun taas vastaava kiinteäsummainen paketti on 500 dollaria kerralla maksettuna.

Photon Serverin ydin, Photon Core, on tehty C++-ohjelmointikielellä. Exit Games perustelee valintaa tehokkuussyillä (13). Photon Coren yhteydessä voidaan käyttää neljää eri internet protokollaa: UDP, Web Sockets, TCP tai HTTP. Arkkitehtuurissa Coren yläpuolella on C#-ohjelmointikielellä kirjoitettuja Business Logic -sääntöjä, jotka määräävät, miten dataa tulee käsitellä. Näitä sääntöjä noudattaen on tehty sovelluskehys (framework). Sovelluskehyksen yläpuolella ajetaan itse palvelimella tapahtuvaa pelilogiikkaa. Palvelimelle on mahdollista tehdä mukautettua palvelinlogiikkaa millä tahansa .NETiin pohjautuvalla koodikielellä, kuten C# (kuva 7).



KUVA 7. Photon Serverin korkean tason arkkitehtuuri (13)

3 TOTEUTUS

Tässä opinnäytetyössä tehtiin kaksi erilaista moninpelitoteutusta. Ensimmäisenä tehtiin jaetun näytön moninpeli, jonka jälkeen tehtiin verkkopeli käyttäen Photon Unity Networkingiä. Molemmat toteutukset on tehty käyttäen Unityä, sekä kummassakin on käytetty Visual Studio 2013 -ohjelmaa ja C#-ohjelmointikieltä.

3.1 Jaetun näytön toteutus

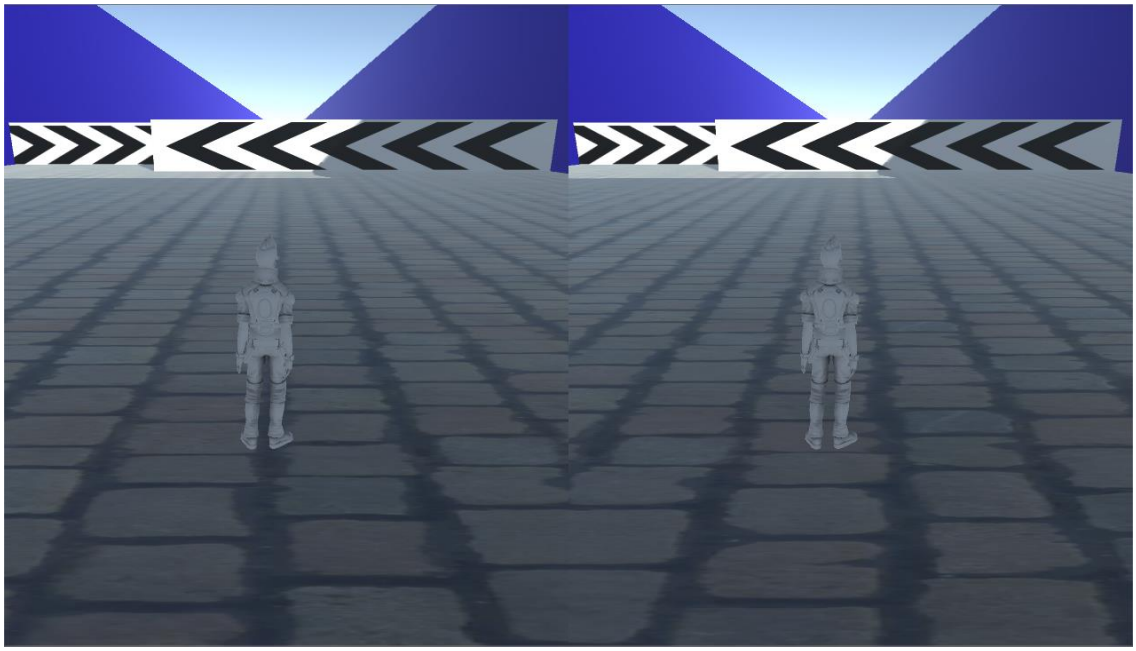
Jaetun näytön toteutus oli mieleiseni valinta, kun kyseessä oli yksittäisellä laitteella pelattava moninpeli. Käytin peliprototyypissä ratkaisua, jossa jokaiselle pelaajahahmolle annetaan oma kameraobjekti, koska tunsin sen olevan itselleni mukavin keino. Peliprototyypin toteutuksessa ei käytetty muita kuin Unityn perusominaisuuksia. Tarkoituksena oli vain kokeilla, kuinka yksinkertaista yhdellä laitteella pelattavan moninpelin toteutus voi olla. Jaetun näytön moninpelin ideaksi muodostui lopulta kahden pelaajan pelattava kilpajuoksupeli.

3.1.1 Peliympäristö

Ensimmäisenä peliin tehtiin ympäristö valmiiksi, jotta olisi paikka jossa voisi testata monen pelaajan toimintoja. Ympäristö on tehty pelkästään Unityssä valmiiksi olevista kuutio- ja taso (plane) -peliobjekteista. Aluksi peliin tehtiin pitkä kapea käytävä erilaisine esteineen, mutta tämä idea tuntui jotenkin turhan yksinkertaiselta. Peliin tarvittaisiin vähän lisää vaihtelevuutta, joten tehtiin aliohjelma, joka satunnaisesti generoi käytävälle peräjälkeen prefabeja, peliobjekteista koottuja uudelleen käytettäviä elementtejä. Tämä mahdollisti erilaisten, mahdollisesti suurienkin, ympäristöjen luomisen pienellä vaivalla. Peliympäristöjen ulkoasu jäi tekijän graafisesta osaamattomuudesta johtuen todella karuksi, mutta ottaen työn tavoitteet huomioon sen ei kuulunutkaan näyttää erityisen ”kauniilta”.

3.1.2 Kamerat

Koska peli on jaetun näytön moninpeli, täytyi molempien pelaajien näkymä saada mahtumaan yhdelle näytölle. Peliin lisättiin kaksi valmista pelihahmoa, molemmat omille juoksualueilleen. Näyttöalueen jakaminen näille kahdelle hahmolle tapahtui muutamalla hahmoille annettujen kameroiden viewporttien kokoa. Tämä tapahtuu yksinkertaisesti valitsemalla pelihahmon kameraobjekti, joka avaa kameran asetukset Unityn tarkastelunäkymään (inspector). Tarkastelunäkymästä kameroiden viewporttien leveydet muutettiin vastaamaan puolen näytön pinta-alaa ja toisen pelaajan viewportin alkamiskohdan x-koordinaatti arvoon 0,5 eli puoleen väliin näyttöä. Näin molempien pelaajien pelinäkymät mahtuvat näytölle (kuva 8).



KUVA 8. Pelinäkymä jaetun näytön pelissä

3.1.3 Pelinäppäimet

Jos kaksi pelaajaa pelaa samalla laitteella, tulee molemmille pelaajille määrittää erilliset pelissä käytettävät näppäimet. Yhdellä näppäimistöllä pelattaessa tulisi kahdelle pelaajalle aika ahdasta, joten ongelma ratkaistiin käyttämällä erillistä

ohjainta. Unityn projektiasetuksista Inputmanagerista käytiin muuttamassa molemmille pelaajille käytettävät näppäinkonfiguraatiot. Samat muutokset näppäimiin voidaan tehdä myös vakiona tulevasta pelin käynnistysohjelmasta, jos projekti käännetään Unityllä valmiiksi ilman editoria toimivaksi peliksi. Järkevän lopputuloksen kannalta näppäimet kannattaa kuitenkin määritellä ensin editorista sopiviksi, koska pelaajat voivat halutessaan muokata ne mieluiseksi käynnistäessään pelin.

3.1.4 Peliin vaikuttavia tapahtumia

Peliin tehtiin pelkän kilpajuoksun lisäksi muutama pelin kulkuun vaikuttava tapahtuma. Pelikentälle sijoitettiin erivärisiä kuutioita, joihin lisättiin kaikki eri tapahtumat sisältävä skripti. Kuution ja pelihahmon Colliderien eli peliobjektien fyysisten rajojen törmätessä skripti laukaisee kuution määrityksistä valitun tapahtuman, jolloin esimerkiksi osuneen pelaajan juoksunopeus kasvaa (kuva 9). Näitä tapahtumia peliin ei tehty projektin luonteen vuoksi kuin muutama, mutta ne vaikuttavat oleellisesti pelin kulkuun.

```
if (speed)
{
    if (other.attachedRigidbody.gameObject.name == "Player1")
    {
        speedBoost1 = true;
        thirdPersonCharacter1.m_AnimSpeedMultiplier = speedAmount;
        thirdPersonCharacter1.m_MoveSpeedMultiplier = speedAmount;
    }

    if(other.attachedRigidbody.gameObject.name == "Player2")
    {
        speedBoost2 = true;
        thirdPersonCharacter2.m_AnimSpeedMultiplier = speedAmount;
        thirdPersonCharacter2.m_MoveSpeedMultiplier = speedAmount;
    }
}
```

KUVA 9. Vauhtia kasvattava tapahtuma

3.2 Photonilla tehty toteutus

Tämän opinnäytetyön alkuperäisenä tarkoituksena oli perehtyä moninpelin tekemiseen Unityllä. Eri vaihtoehdoista lopulliseksi työkaluksi toteutukseen valitsin juuri Photonin, koska se vaikutti monipuolisimmalta, mutta samalla yksinkertaisimmalta ratkaisulta. Pelin ideaksi mietin ensin jaetun näytön toteutuksessa käytettyä juoksupeliä, mutta ohjaavan opettajan suosituksesta aiheeksi tuli FPS-peli.

3.2.1 Karttapohjan teko

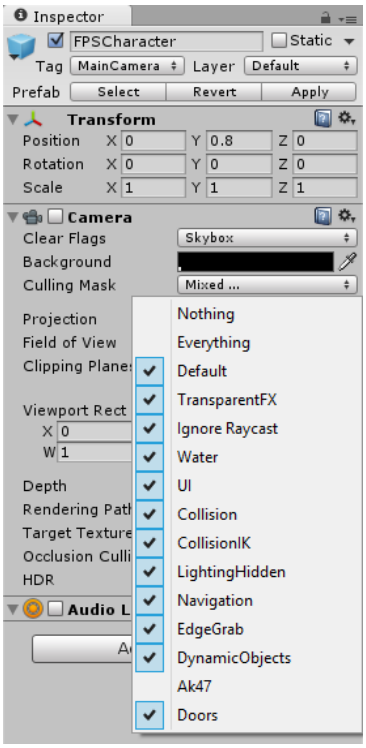
Moninpeliin otettiin karttapohjaksi Unity Asset Storesta, Unityn omasta sisältökaupasta löytyvä projektidemoon ”Unity Labs” sisältyvä kartta. Kartasta karsittiin suurin osa pois ja jäljelle jäi iso aukea huone, muutamia näköesteitä, korotettuja suojapaikkoja ja kapeaa käytävää. Ympäri karttaa sijoitettiin viisi tyhjää peliobjektia, joihin pelaajat voivat kuollessaan yksi kerrallaan uudelleensyntyä.

3.2.2 Pelihahmo

Pelissä käytettiin Unityn valmista ensimmäisen persoonan pelihahmoa, johon lisättiin näkymisen varmistamiseksi kapselin muotoinen 3d-objekti. Pelihahmolle tarvittiin jonkinlainen ase, koska kyse oli ampumispelistä. Unityn Asset Storesta löytyi AK47-rynnäkkökivääriä muistuttava valmis paketti, joka tuotiin peliin. Ase yhdistettiin pelihahmoon ja aseteltiin näkymään pelihahmolle kuuluvassa kamerassa. Aseen piipun päähän tehtiin vielä animaatio suuliekille, joka välähtää kun aseella ammutaan.

Pelissä pelaaja ei voi ampua ja juosta yhtä aikaa, joten aseeseen tehtiin havainnollistavaksi tekijäksi animaatio, joka laskee aseen piipun alas pelaajan juostessa. Animaatiota testatessa huomattiin aseen piipun menevän seinien ja esteiden sisään, jos pelihahmolla liikuttiin tarpeeksi lähelle näitä peliobjekteja. Tähän ratkaisuksi otettiin käyttöön toinen kamera, joka renderöi vain aseen eikä mitään muuta. Pelaajan aseelle tehtiin oma renderöintitaso, joka otettiin pois käytöstä hahmon pääkameran asetuksista (kuva 10). Vastavuoroisesti toisen

kameran asetuksista valittiin vain aseen oma renderöintitaso, jolloin aseen piippu ei enää näyttänyt leikkautuvan pelaajan ollessa lähellä toista peliobjektia.



KUVA 10. Pääkameran asetuksia

3.2.3 Photon Unity Networking

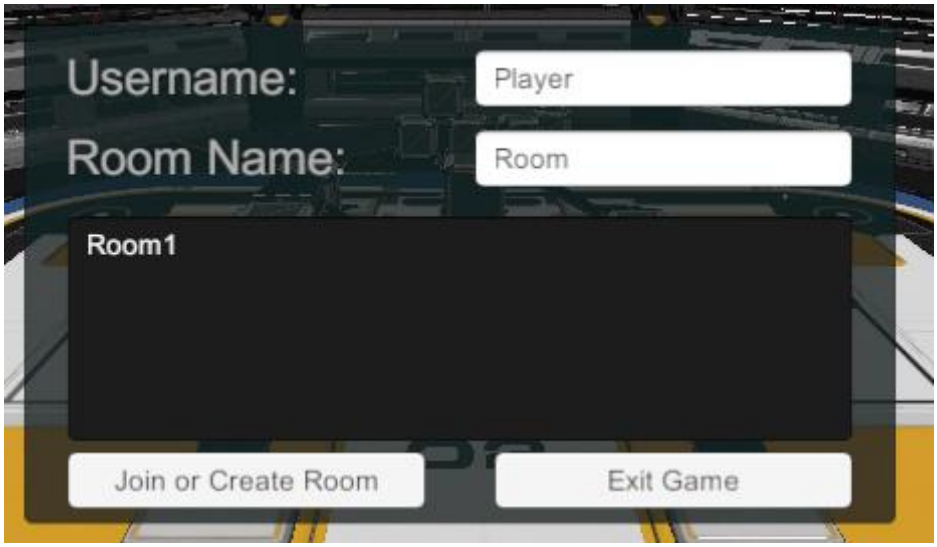
Photon Unity Networking (PUN) ladattiin projektiin Unity Asset Storesta. Välittömästi projektiin tuomisen jälkeen PUN voidaan joko yhdistää Photon Cloudiin tai käyttää omaa palvelinta. Tässä projektissa käytettiin Photon Cloudia, joten Photonin verkkosivujen kautta täytyi rekisteröityä kehittäjäkäyttäjäksi. Rekisteröitymisen jälkeen projekti tuli rekisteröidä oman käyttäjätunnuksen taakse, jolloin saatiin projektille uniikki AppId. AppId syötettiin Unityn PUN-asetuksiin, jolloin projekti käyttää omaan tunnukseen kytkettyä Photon Cloud -palvelinta.

3.2.4 PhotonView

Jotta pelihahmon saa päivitettyä Photonin avulla verkon yli, täytyi hahmoon lisätä PhotonView-komponentti. PhotonView on olennainen osa Photonin toimintaa, koska vain PhotonView-komponentissä voidaan määrittää mitkä lokaalit peliobjektit synkronoidaan verkon yli. Komponentin lisäyksen jälkeen pelihahmosta tehtiin prefab-peliobjekti ja se poistettiin pelimaailmasta, koska se tullaan myöhemmin lisäämään ohjelmallisesti peliin.

3.2.5 Aula

Peliin tehtiin mahdollisuus luoda omalla nimellä tehtyjä pelihuoneita. Tämä tarkoitti sitä, että ensin täytyi luoda aula, jonka kautta huoneita on mahdollista perustaa. Photonin ollessa käytössä, jokainen pelaaja yhdistää automaattisesti aulaan jos peli käynnistetään tai jostain pelistä poistutaan. Aulaan liityttäessä suoritetaan aina *OnJoinedLobby*-metodi, joka aukaisee aulan käyttöliittymän. Käyttöliittymästä tehtiin mahdollisimman yksinkertainen (kuva 11). Aulassa pelaajan on mahdollista määrittää oma pelissä näkyvä käyttäjänimi, sekä pelihuone johon pelaaja liittyy tai vaihtoehtoisesti luo, jos sen nimistä pelihuonetta ei ole vielä olemassa. Käyttöliittymässä on myös laatikko johon listautuu kaikki pelissä sillä hetkellä käynnissä olevat pelihuoneet.



KUVA 11. Aulan käyttöliittymä

3.2.6 Pelihuone ja pelihahmon tuonti peliin

Pelihuoneeseen liittymisen jälkeen Photon suorittaa automaattisesti metodin OnJoinedRoom. Tähän metodiin kirjoitettiin aluksi kaikki pelaajaan peliin tuomiseen tarvittava koodi, mutta uudelleen käytettävyyden takia niistä tehtiin myöhemmin omat metodit. OnJoinedRoom-metodissa kutsutaan StartSpawning-metodia, joka taas käynnistää coroutinen SpawnPlayer eli funktion, joka voi pysäyttää oman suorituksensa tietyksi ajaksi. Tämä on pelaajan uudelleensyntymisen kannalta tärkeä ominaisuus.

Jotta pelihahmon luominen synkronoituisi kaikille pelissä oleville pelaajille, ei voida siihen käyttää Unityn omaa Instantiate-funktiota, vaan käytetään Photonin PhotonNetwork.Instantiate-funktiota, jolloin peliobjekti luodaan karttaan jokaiselle pelaajalle.

3.2.7 Dataliikenne pelaajien välillä

Jotta pelaajien liikkeet saadaan näkymään jokaiselle pelissä olevalle pelaajalle, täytyi ne synkronoida jotenkin verkon yli. Yksinkertaisimmillaan tämä onnistuu pudottamalla pelaajan pelihahmon transform-komponentti, PhotonView-skriptin ObservedComponent-taulukkoon. Tähän taulukkoon lisätyt komponentit synkronoidaan automaattisesti verkon yli kaikille käyttäjille. Jos transform-komponentti lisättiin synkronoitavaksi, oli toisille pelaajille näkyvä liikehdintä pätkivää, joten tähän tarkoitukseen tehtiin skripti.

Photonissa on metodit nimeltään UpdateData ja OnPhotonSerializeView. OnPhotonSerializeView-metodissa määritetään mitä dataa lähetetään (esimerkiksi paikkatieto) ja vastaanotetaan, kun taas UpdateData-metodissa vastaanotettua dataa voidaan käyttää esimerkiksi päivittämään muiden pelaajien paikkatietoja sulavasti.

Testatessa peliä huomattiin yhden pelaajan liikkuessa kaikkien peliin liittyneiden pelaajien liikkuvan samalla. Tämä johtui siitä, että kaikilla pelihahmoilla on aina syntyessään liikkumiseen käytettävät skriptit käytössä. Tämä korjattiin ottamalla

pelihahmon prefabista pois käytöstä kaikki tärkeät elementit (kamerat, skriptit jne.) ja ne otettiin käyttöön ohjelmallisesti pelihahmon syntyessä. Näin jokaisen peliin liittyvään pelaajan pelihahmo on lokaalisti käytettävissä, mutta muiden pelissä olevien pelaajien pelissä se on pelkkä mallinukke, jonka liikkeet synkronoidaan verkon yli.

Koska kyseessä oli ammuskelupeli, tehtiin pelaajille mahdollisuus ampua tähtäimessään olevia kohteita. Jokaisella pelaajalla on tietty määrä terveyspisteitä, joita menetetään, jos joku toinen pelaaja onnistuu osumaan häneen. Terveyspisteiden ollessa nolla pelaaja kuolee ja syntyy uudelleen hetken päästä. Pelissä ampuminen tekee 25 pistettä vahinkoa ja pelaajalla on 100 terveyspistettä. Tässä tilanteessa muodostui pieni ongelma. Pelaajien tekemät vahinkopisteet eivät päivittyneet verkon yli, vaan kaksi pelaajaa saattoi molemmat tehdä 75 vahinkopistettä yhteen pelaajaan, ilman että osumia saanut pelaaja olisi kuollut. Tämä johtui siitä, että vahinkopisteitä ei synkronoitu lainkaan verkon yli. Ratkaisuksi tähän käytettiin RPC-kutsua (Remote Procedure Call).

RPC-kutsulla kutsuttava metodi on syntaksiltaan kuin normaali metodi, mutta se suoritetaan jokaisen pelaajan pelissä yhtäaikaisesti. Jotta metodia voidaan kutsua RPC-kutsulla, käytetään sitä ennen PunRPC-attribuuttia. Photonissa RPC-kutsu tehdään hakemalla PhotonView-komponentti, tässä tapauksessa osutusta pelaajasta, ja kutsumalla jotain siihen kuuluvaa PunRPC-attribuutilla merkattua metodia. Näin osumat rekisteröityvät oikein, mutta ongelmia esiintyi eri kohdassa. Yhden pelaajan kuollessa kaikki muut pelaajat yrittävät yhtä aikaa poistaa tämän peliobjektia pelistä, mutta vain yksi näistä onnistuu, koska muut yrittävät poistaa jo hävitettyä peliobjektia. Tässä ratkaisuksi käytettiin PhotonView-komponentin ominaisuutta isMine, jota voidaan käyttää vertailussa yhtenä elementtinä. Se palauttaa true-arvon, jos kyseessä oleva PhotonView-komponentin sisältävä peliobjekti on omassa käytössä. Näin ainoastaan ammutuksi tullut pelaaja poistaa itse itsensä verkosta ja syntyy uudelleen (kuva 12).

```
[PunRPC]
public void GetShot(float damage, string killerName)
{
    if (photonView.isMine)
    health -= damage;

    if (health <= 0 && photonView.isMine)
    {
        networkManager.GetComponent<NetworkManager>().AddMessage(PhotonNetwork.player.name + " got shot by " + killerName);

        if (Respawnme != null)
        {
            Respawnme(3f);
        }

        PhotonNetwork.Destroy(gameObject);
    }
}
```

```
    hit.transform.GetComponent<PhotonView>().RPC("GetShot", PhotonTargets.All, damage, PhotonNetwork.player.name);
```

KUVA 12. PunRPC-attribuutti (yllä). RPC-kutsu Photonissa (alla)

3.2.8 Viestintä

Pelaajille ei välittynyt pelissä tietoa siitä, kuka oli ampunut kenetkin tai oliko peliin liittynyt uusia pelaajia. Tätä varten tehtiin oma viestintäjärjestelmä, joka ilmoitti tekstikenttään erilaista infoa pelitapahtumista. Tekstin päivittäminen tehtiin RPC-kutsun avulla, joten kaikkien pelaajien tekstikenttä päivittyy yhtäaikaisesti. Koska kaikki info ei kuitenkaan ole olennaista kaikille pelaajille, tehtiin RPC-kutsulle oma metodi, jossa id-numeroarvoa käyttämällä voi rajoittaa viestin vastaanottaja ryhmiä.

4 YHTEENVETO

Työn tarkoituksena oli tutustua moninpelien toteuttamiseen Unity-pelimoottorilla. Tuloksena saatiin kaksi toteutusta: jaetun näytön peli ja verkkopeli.

Jaetun näytön toteutuksen lopputulos oli tavoitteisiin nähden sopiva. Yleisesti en ollut tyytyväinen työn lopputulokseen, mutta koska se oli vain lähinnä kokeellisella tasolla toteutettu peli, se sai jäädä osaksi työtä. Ongelmia jaetun näytön toteutusta tehdessä ei juurikaan ilmennyt, koska kaikki Unityn puolella tapahtuva koodaus ja peliobjektien asettelu on ilman verkkotoiminnallisuutta yksinkertaista tämän tason pelissä. Jaetun näytön toteutusta en ole jatkokehittämässä, vaikka siitä olisikin mahdollista tehdä pienillä parannuksilla ihan toimiva pelidemo.

Photonilla toteutettuun verkkopelin lopputulokseen olin erittäinkin tyytyväinen. Photon ja Unity toimivat yhdessä erittäin hyvin ja antoivat loistavan pohjan lähteä toteuttamaan työtä. Ongelmia työn aikana ilmeni vähän väliä, mutta ratkaisut näihin löytyivät ajan kanssa tutkimalla Photonin dokumentaatiota tai etsimällä foorumeilta tietoa ongelmakohdista. Suurimpana ongelmana nousi esille verkon yli lähetettävän ja vastaanotettavan datan käyttö eri tarkoituksissa. Tähän kuitenkin harjaantui sitä enemmän, mitä pidemmälle toteutus eteni. Verkkopelitoteutuksen tämän hetkinen tilanne kaipaa vielä jatkokehitystä. Pohjana ensimmäisen persoonan ammuskelupeliin se toimii erittäin hyvin, mutta kaipaa silti paljon pelillisiä elementtejä, ennen kuin sitä uskaltaa peliksi kutsua. Aion erittäin suurella todennäköisyydellä palata jatkokehittämään projektia myöhemmin.

Kokonaisuutena opinnäytetyö antoi erittäin paljon hyvää tietoa sekä taitoa moninpelien toteuttamisesta. Erityisesti verkkopelin toteuttamisesta saatu kokemus datan käsittelystä sekä Photonin käytöstä oli todella antoisaa. Olen erittäin kiinnostunut työskentelemään Photonin parissa myöhemmin eri projektien merkeissä.

LÄHTEET

1. Sherrod, Allen – Jones, Wendy 2012. Beginning DirectX 11 Game Programming. Google Books. Saatavissa: https://books.google.fi/books?id=7ZkLAAAAQBAJ&pg=PA55&dq=split+screen+render&hl=en&sa=X&ved=0ahUKEwjIpo3siNHJAhVIuBQKHaDAAfIQ6AEIIDAB#v=onepage&q&f=false. Hakupäivä 16.12.2015

2. Mario Kart 8. 2014. Nintendo. Saatavissa: http://hexdimension.com/wp-content/uploads/2014/06/Mario-Kart-8-4-player.jpg. Hakupäivä 22.12.2015.

3. Magicka 2. 2015. Paradox Interactive. Saatavissa: https://i.ytimg.com/vi/9B1Y4W2f-bE/maxresdefault.jpg. Hakupäivä 22.12.2015

4. League of Legends World Championships vievers. 2015. League of Legends. Saatavissa: http://na.leagueoflegends.com/en/news/esports/esports-editorial/one-world-championship-32-million-viewers. Hakupäivä 3.1.2016.

5. Game Server. 2016. Wikipedia. Saatavissa: https://en.wikipedia.org/wiki/Game_server. Hakupäivä 8.2.2016

6. Measuring network quality. 2016. Ookla. Saatavissa: http://www.ping-test.net/learn.php. Hakupäivä 2.2.2016

7. Source Multiplayer Networking. 2016. Valve. Saatavissa: https://developer.valvesoftware.com/wiki/Source_Multiplayer_Networking. Hakupäivä 4.2.2016

8. Photon SDK versions. 2015. Photon. Saatavissa: https://www.photonengine.com/en/Realtime/Download. Hakupäivä 25.12.2015.

9. Photon master server locations. 2016. Photon. Saatavissa: https://www.photonengine.com/en/Realtime. Hakupäivä 4.1.2016.

10. Bolt becomes part of the Photon family. 2015. Photon. Saatavissa: http://blog.exitgames.com/2015/06/bolt-announcement/. Hakupäivä 6.1.2016.

11. Photon pricing. 2016. Photon. Saatavissa: https://www.photonengine.com/en/Realtime/Pricing. Hakupäivä 6.1.2016

12. Photon Cloud regions. 2016. Photon. Saatavissa: https://doc.photonengine.com/en/realtime/current/reference/regions. Hakupäivä 5.1.2016.

13. Photon Server High Level Architecture. 2016. Photon. Saatavissa: http://doc.photonengine.com/en/onpremise/current/getting-started/photon-server-intro. Hakupäivä 7.1.2016.