LAB University of Applied Sciences

Verkkomoninpelin suunnittelu ja toteutus

LAB-ammattikorkeakoulu
Insinööri (AMK)
2024
Jere Syrjänen

Tiivistelmä

Tekijä(t)	Julkaisun laji	Valmistumisaika
Jere Syrjänen	Opinnäytetyö, AMK	2024
	Sivumäärä 26	

Työn nimi

Verkkomoninpelin suunnittelu ja toteutus

Tutkinto ja koulutusala

Insinööri (AMK), tieto- ja viestintätekniikan koulutus

Toimeksiantajaorganisaatio (jos opinnäytetyöllä on toimeksiantaja)

Ape Brain Games Oy

Tiivistelmä

Opinnäytetyön tavoitteena oli suunnitella ja toteuttaa Ape Brain Gamesille kehys verkkomoninpeliä varten. Toteutuksen verkkotoiminnot täytyi tehdä yhteensopiviksi reaaliaikaiselle ammuntapelille. Lisätavoitteena oli varmistaa verkkomoninpeli kehyksen yhteensopivuus Steam-myyntialustan kanssa.

Työssä tutkittiin ohjelmointia ja blueprint-työkalun käyttöä Unreal Engine pelimoottorissa. Näiden lisäksi tutkittiin Epic Gamesin ja Unreal Enginen tarjoamia verkkomoninpelipalveluja ja liitännäisiä.

Toteutuksen kehitys tapahtui käyttäen Unreal Engine pelimoottoria, C++- ja blueprint -ohjelmointia. Näiden lisäksi kehityksessä hyödynnettiin pelimoottorin tarjoamaa Online Subsystem liitännäistä ja sen rajapintoja.

Työn tuloksena saatiin toimeksiantajalle verkkomoninpelikehys Unreal Enginellä kehitettävään peliin, jonka yritys on ottanut käyttöön.

Asiasanat

Unreal Engine, C++, blueprint, verkkomoninpeli

Abstract

Author(s)	Type of Publication	Published
Jere Syrjänen	Thesis, UAS	2024
	Number of Pages 26	

Title of Publication

Creating an online multiplayer game

Degree, Field of Study

Engineer (UAS), Information and Communications Technology

Organisation of the client (if the thesis work is commissioned by another party)

Ape Brain Games

Abstract

The aim of the thesis was to design and implement an online multiplayer framework for a computer game developed by Ape Brain Games. The online functionality of the implementation had to be made compatible with a real-time shooting game. An additional goal was to ensure the compatibility of the online multiplayer framework with the Steam sales platform.

Research for this work involved programming and the use of the blueprint tool in the Unreal Engine game engine. In addition, the online multiplayer services and plug-ins provided by Epic Games and Unreal Engine were studied.

The implementation was developed using the Unreal Engine game engine, C++ and blueprint programming. In addition, the development made use of the Online Subsystem plug-in provided by the game engine, and its interfaces.

As a result of this work, the client was provided with an online multiplayer framework for a game developed with Unreal Engine, which has been implemented by the company.

Keywords

Unreal Engine, C++, blueprint, online multiplayer

Sisällys

1 Johdanto

Verkkomoninpelit lukeutuvat suosituimpien videopeli genrejen joukkoon. Suuret pelifirmat, kuten EA ja Ubisoft käyttävätkin näiden toteuttamiseen paljon resursseja. Verkkomoninpelaamisen suosioon vaikuttaa ihmisten halu olla vuorovaikutuksessa toistensa kanssa.

Opinnäytetyön toimeksiantajana on lahtelainen indie-peliyhtiö Ape Brain Games Oy. Ape Brain Games koostuu neljästä pelinkehittäjästä, joita on pääkehittäjä, ohjelmoija, 3D-mallintaja ja 2D-artisti. Ape Brain Gamesilla on yksi aiemmin julkaistu peli HexAlert, joka on saatavilla Steam-myyntialustalla.

Opinnäytetyön tavoitteena on verkkomoninpelin suunnittelu ja toteutus. Peliä kehitetään Unreal Engine 5 -pelimoottorilla ja se on tarkoitus julkaista PC:lle. Peli on lintuperspektiivistä pelattava reaaliaikainen ammuntapeli, jossa verkossa pelaavat pelaajat kilpailevat toisiaan vastaan lyhytkestoisissa otteluissa, minkä jälkeen siirrytään uuteen otteluun.

Opinnäytetyössä suunnitellaan ja toteutetaan kehys verkkomoninpelille hyödyntäen Epic Gamesin ja Unreal Engine -pelimoottorin verkkomoninpelipalveluita. Lisäksi tavoitteena on suunnitella ja toteuttaa kehys siten, että sen käyttäminen on helppoa ja sitä pystyy hyödyntämään pelin kaikkien verkonvälityksellä tapahtuvien asioiden toteuttamiseen. Suurimman osan verkkokutsuista on oltava reaaliaikaisille ammuntapeleille tyypillistä, nopeaa ja lyhytviiveistä datan siirtoa. Lisäksi toteutuksen tulisi olla yhteensopiva Steam-myyntialustan kanssa.

Toteutuksen ohjelmointi Unreal Enginessä tapahtuu käyttämällä C++-ohjelmointikieltä ja visuaalisen ohjelmoinnin työkalua, eli visual blueprint -työkalua. C++-ohjelmointikieltä käytetään tyypillisesti kirjoittamaan komponenttien backend-funktionaalisuus, jota voidaan hyödyntää pelimoottorissa käyttäen blueprint-työkalua.

Toteutuksen kehityksessä hyödynnetään pelimoottorin tarjoamia verkkomoninpelipalveluja, joihin kuuluu Online Subsystem. Online Subsystem tarjoaa rajapinnan verkkopelille tyypillisille kutsuille, sekä muita modernissa verkkopelaamisessa hyödynnettäviä apufunktioita.

2 Pelimoottori

2.1 Unreal Engine

Unreal Engine on Epic Gamesin vuonna 1998 julkaisema pelimoottori, joka julkaistiin yhdessä pelimoottorilla kehitetyn Unreal-nimisen pelin kanssa (Polygon 2012). Tänä aikana Epic Games teki erillisiä lisenssisopimuksia eri pelifirmojen kanssa, jonka takia indie-pelistudiot eivät juuri käyttäneet sitä. Unreal Engine 4 julkaistiin vuonna 2014, minkä yhteydessä se siirtyi kuukausimaksulliseen lisenssiin, sekä 5 prosentin rojaltimaksuihin pelien myynnistä. Tästä seuraavana vuonna Epic Games poisti kuukausimaksullisen lisenssin ja alkoi periä 5 prosentin rojaltimaksua yli 3000 dollaria tienaavilta peleiltä. (Gamedeveloper 2015.) Vuonna 2020 Epic Games alkoi periä rojaltimaksuja vain niiltä peleiltä, joita oli myyty yli miljoonan dollarin arvosta (Arstechnica 2020).

Unreal Engine on kirjoitettu C++ kielellä ja nykyisen Unreal Engine 5:n lähdekoodi on saatavilla GitHubista (Unreal Engine a). Pelimoottoria voi käyttää Windows-, Linux- ja OS X-alustoilla. Unreal Enginellä pystyy tekemään pelejä Windows-, Linux- ja OS X-alustoille, uusimmille konsoleille sekä Android ja iOS laitteille.

Unreal Engine tarjoaa pelinkehittäjille digitaalisen kauppapaikan, jossa voi myydä ja ostaa kaikenlaista pelien kehitykseen liittyvää, kuten 3D malleja, C++-, tai blueprint -komponentteja. Unreal Engine tarjoaa myös opetusmateriaalia pelinkehittäjille, ohjelmistorajapinnan suosituksia, sekä yhteisöjohtoisen tukialustan ja foorumin, jossa kehittäjät voivat auttaa toisiaan ongelmanratkaisussa.

2.2 Ohjelmointi Unreal Enginessä

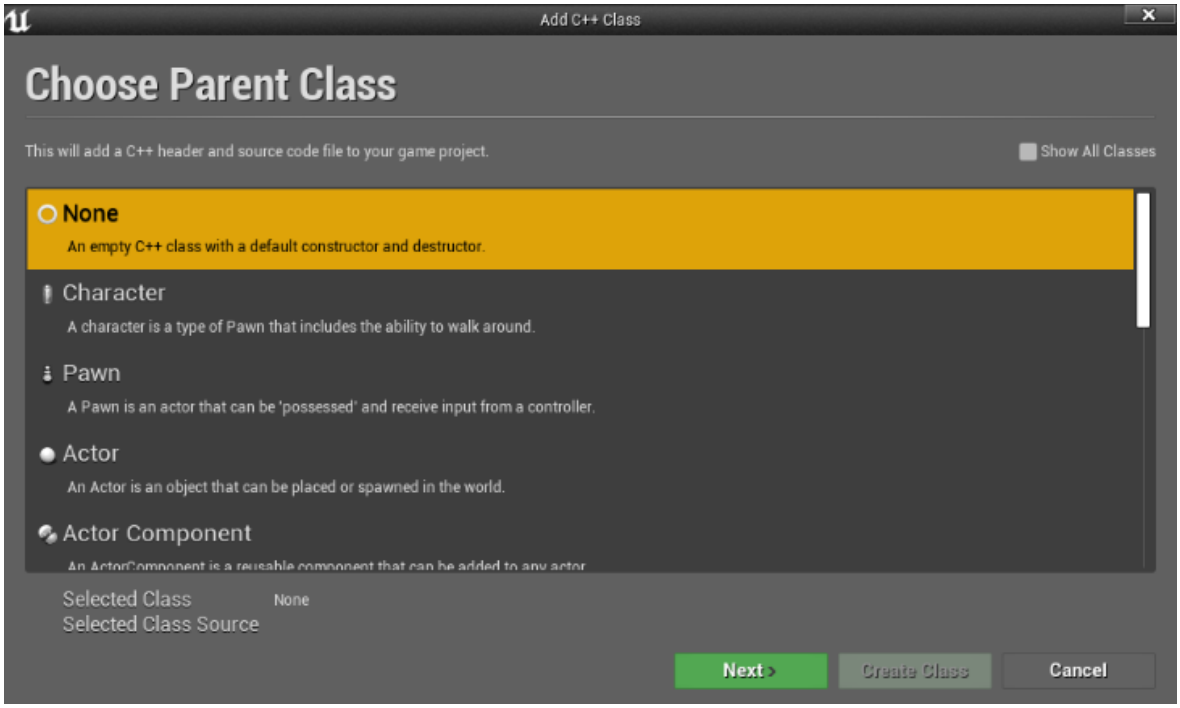
Unreal Enginessä on kaksi tapaa ohjelmoida tapahtumia ja elementtejä peliin, visuaalinen blueprint -työkalu ja ohjelmointi C++-ohjelmointikielellä. Blueprint Visual Scripting on Unreal Enginen tarjoama olioperustainen visuaalisen ohjelmoinnin työkalu. (Unreal Engine b.) Kummallakin tavalla pelejä pystyy kehittämään Unreal Enginessä, mutta paras toimintatapa on hyödyntää molempien vahvoja puolia (Andrii Chornyl).

Myös Python ohjelmointikielen käyttö on tullut mahdolliseksi Unreal Enginessä. Pythonia ei voi käyttää pelitapahtumien tai komponenttien ohjelmointiin, vaan sen käyttö on tarkoitettu muun muassa kehitysprosessin automatisointiin ja assettien luonnin pipelineen. (Unreal Engine c.) Assettien luonnin pipelinellä tarkoitetaan työtä, joka on tehtävä tuodakseen esimerkiksi 3D-malleja ulkoisesta 3D-mallinnus ohjelmasta Unreal Engineen.

2.2.1 C++

Standardin C++-ohjelmointikielen ohjelmointi on mahdollista Unreal Enginessä, mutta pelimoottori tarjoaa kattavan rajapinnan ohjelmointiin. Pelimoottori tarjoaa myös muita ohjelmoinnin työkaluja ja makroja, joiden hyödyntäminen ohjelmoidessa helpottaa komponenttien luonnissa, ja joiden avulla voidaan kontrolloida luodun C++-komponentin käyttäytymistä pelimoottorin sisällä. Unreal Engine e.)

Kuvan 1 mukaisesti C++-komponenttia luodessa valitaan ensin tarpeen mukaan Unreal Enginen Class Wizardin avulla kantaluokka, josta luotu komponentti peritään. Kantaluokka voidaan jättää myös tyhjäksi.



Kuva 1. Unreal Engine Class Wizard työkalu

Kantaluokkia, joita Unreal Engine tarjoaa valmiina, on muun muassa PlayerController, Character, Actor, Pawn, GameMode, Scene ja Actor Component. Jokainen näistä tarjoaa valmiiksi niille ominaisia muuttujia ja funktioita, ja joita käytetään erilaisiin tarkoituksiin. Esimerkiksi Character luokka, joka on peritty Pawn luokasta, sisältää SkeletalMesh komponentin, jonka avulla voidaan toteuttaa monimutkaisia animaatioita, sekä CharacterMovement ja Capsule komponentin, joiden avulla saadaan tehtyä hahmon liikkeet ja törmäyksentunnistus. Esimerkiksi Character kantaluokan käyttö sopii tämän takia erittäin hyvin pelattavien hahmojen ja NPC hahmojen kantaluokaksi. Myös tyhjän tai itse luotujen kantaluokkien valinta on mahdollista. (Unreal Engine f.)

C++-komponentin luonnin jälkeen voidaan aloittaa komponentin ohjelmointi. Komponenttia ohjelmoidessa on hyvä käyttää Unreal Enginen tarjoamia makroja, sillä ne toteuttavat toiminnallisuutta, joka mahdollistaa ja helpottaa C++-ohjelmointikielen käyttöä pelimoottorin sisällä. C++ luokille, muuttujille ja funktioille voidaan lisätä makrot, kuten UCLASS luokkamakro ja UPROPERTY muuttujamakro. Sen lisäksi makrot tarvitsevat niiden käyttäytymistä kontrolloivat muuttujat, mikäli halutaan mahdollistaa niiden käyttö blueprint-komponentin sisällä. Makrojen muuttujilla pystytään määrittämään asioita, kuten komponentin näkyvyyttä ja muokattavuutta pelimoottorin sisällä. (Unreal Engine g.)

2.2.2 Blueprint-ohjelmointi

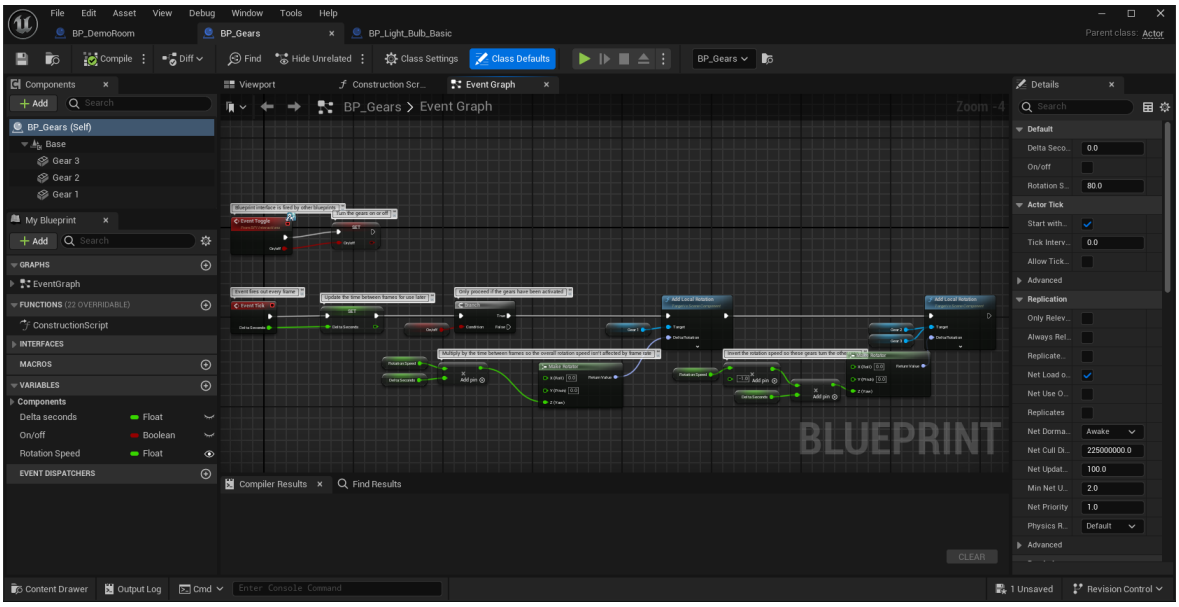
Blueprint on Unreal Enginen oma visuaalisen ohjelmoinnin työkalu, jonka tarkoitus on mahdollistaa nopea ja joustava työnteko kirjoittamatta koko ohjelmaa C++-kielellä. Blueprint-komponenteissa peritään C++-kielellä luoduista komponenteista blueprint-komponentteja, joita voidaan muokata, yhdistää muihin blueprint-komponentteihin ja kutsua C++-komponenttiin luotuja funktioita pelimoottorin sisällä. Tämä mahdollistaa sen, että suunnittelijan ei tarvitse tietää kaikkea, mitä C++-ohjelmassa tapahtuu. Riittävä tieto on, että minkä muuttujien arvoja tarvitsee muokata, mitä funktioita kutsumalla, ja niille annettuja parametreja käyttämällä saavutetaan haluttu lopputulos. (Unreal Engine d.)

Blueprint-ohjelmoinnin helppokäyttöisyyden vuoksi se on useammalle pelinkehittäjälle helpompi valinta Unreal Engineä käyttäessä, sillä ohjelmointi C++-kielellä on vaikeampaa ilman entuudestaan olemassa olevaa ohjelmointi kokemusta (Andrii Chornyl).

Unreal Enginellä ohjelmoijan on helppoa testata tekemiensä komponenttien toimintaa tekemällä niistä blueprint-komponentteja (Andrii Chornyl). Tämä johtuu siitä, että C++-ohjelmointikielellä kirjoitettuihin komponentteihin itseensä ei aina toteuteta omaa toiminnallisuutta, vaan toiminnallisuus on tarkoitus toteuttaa blueprint-komponentissa. Tällöin helpoin tapa testata toimivuutta on periä tehdystä C++-ohjelmointikielellä toteutetusta komponentista blueprint-komponentti ja tehdä tarvittavat testit pelimoottorin sisällä.

Blueprint-ohjelmoinnin käytössä on myös omat huonot puolensa, koska toisin kuin C++-ohjelmointikieli, joka käännetään suoraan konekieleksi, blueprint-ohjelmointikieli tulkataan pelimoottorin virtuaalikoneella suorituksen aikana. Tämän takia blueprint-ohjelmointi ei sovellu hyvin suurta suorituskykyä vaativan funktionaalisuden, kuten fysiikan, animaatioiden ja renderöinnin toteuttamiseen. (Unreal University.)

Kuvassa 2 on nähtävissä blueprint editor -käyttöliittymä. Kuten kuva osoittaa, blueprint-työkalun käyttö on node-pohjaista visuaalista ohjelmointia, jolla voidaan helposti toteuttaa funktionaalisuutta yhdistelemällä eri funktioita ja komponentteja toisiinsa.



Kuva 2. Unreal Engine Blueprint -käyttöliittymä (Unreal Engine d)

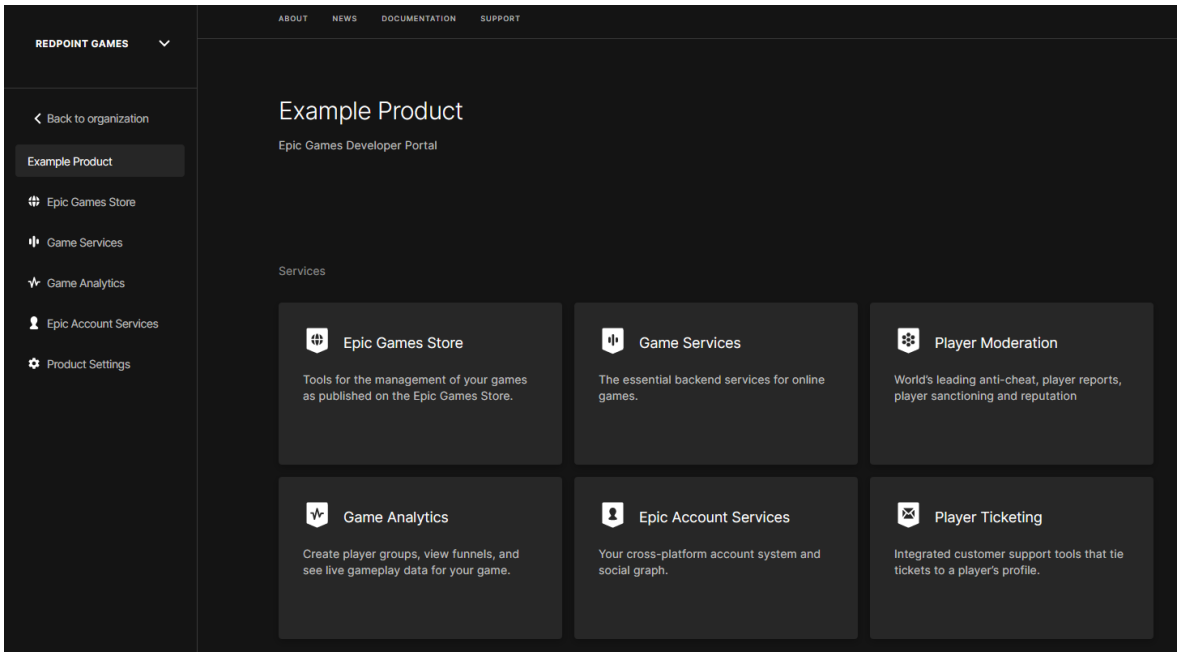
2.3 Epic Online Services (EOS)

Epic Online Services tarjoaa kattavan valikoiman verkkomoninpelin backend työkaluja, niin käyttäjätilille kuin itse pelille. Pelipalveluihin kuuluu itse verkkokehyksen toteutus, joka tarjoaa palveluita, kuten järjestelmäriippumattoman verkkopelaamisen, edistymisen seurantapalvelut ja operaatiot. (Epic Online Services a.)

Järjestelmäriippumattoman moninpelin toteutukseen kuuluu mm. keskustelupalvelu, matchmaking ja P2P-verkkototeutukset. EOS tarjoama matchmaking mahdollistaa sessiopohjaisen verkkototeutuksen, jonka pääasialliset toiminnot ovat järjestää, löytää ja olla vuorovaikutuksessa verkkosession kanssa. (Epic Online Services b.) Session pituus voi vaihdella asetusten mukaan yhdestä ottelusta pidempään moni kenttäiseen sykliin. Applikaatiolla voi olla useita sessioita samanaikaisesti, joista jokainen eroaa uniikilla nimellä. Aktiivinen sessio luodaan paikallisesti käyttäjän koneelle, kun sessio luodaan tai siihen liitytään. Koska aktiiviset sessiot ovat olemassa paikallisesti, applikaation tulee tuhota ne, kun niitä ei enää tarvita. Jos näin ei tehdä, sessio voi jäädä varatuiksi resursseiksi. (Epic Online Services e.)

P2P-rajapinta tarjoaa rajapinnan verkkopelaamiselle, jossa muodostetaan yhteys autentikoitujen käyttäjien välille käyttäen osoitteenmuunnosta. P2P-rajapinta on oletuksellisesti turvallinen, koska se käyttää Datagram Transport Layer Security suojausta eli DTLS. DTLS ansiosta yhteysnopeus on huomattavasti korkeampi, koska sen ei tarvitse päätellä mitä dataa tarvitsee lähettää ja kenelle. (Epic Online Services d.)

Edistymisen seurantapalveluihin kuuluu pelaajadatan tallennus, statistiikka, pelidatan tallennus (title storage) ja saavutukset. Title storage rajapinta tarjoaa kryptatun datan hakemista pilvipalveluista. Tallennettuun dataan pääsee käsiksi kaikki kirjautuneet käyttäjät, miltä tahansa laitteelta. Title storagen ero pelaajadatan tallennukseen on se, että title storageen tallennetaan pelille ominaista dataa, jonka tallennus tapahtuu kehittäjien puolesta Epic Developer Portalista. (Epic Online Services c.) Kuvassa 3 on Epic Developer Portalista esimerkki, jossa on tehtynä Example Product niminen tuote. Kuvassa näkyy Epic Online Servicesin tarjoamat palvelut.



Kuva 3. Epic Developer Portal (Redpoint Games)

2.4 Online Subsystem ja Online Subsystem Steam

Online Subsystem on Unreal Enginen tarjoama verkkopelin kehitykseen käytettävä liitännäinen, joka tarjoaa eri myyntialustoille ominaista verkkofunktionaalisuutta. Online Subsystem takaa myös, että kehittäjiltä vaadittavat muutokset eri alustoille julkaisemiseksi ovat vain hienosäätöä. Online Subsystem tarjoaa myös verkkomoninpelaamiseen olennaisia rajapintoja, kuten käyttäjään liittyvän rajapinnan, sekä pilvipalvelu ja sessio -rajapinnan, sekä muita hyödyllisiä rajapintoja kuten keskustelupalvelut, friends list ja kauppapaikan. (Unreal Engine j.)

Online Subsystem -liitännäinen käyttää delegaatteja kaikkiin verkko operaatioihin ja takaa että delegaatteja kutsutaan silloin, kun ne ovat saatavilla. Delegaatit ovat funktioita, jotka voidaan sitoa dynaamisesti toisen olion funktioon. Sitomisen jälkeen toisessa oliossa

sijaitsevaa funktiota kutsutaan aina kun delegaattia kutsutaan. Delegaattien kutsut tapahtuvat asynkronisesti. Delegaattien avulla ei tarvitse erikseen ohjelmoida omaa virheentarkastusta. (Unreal Engine j.)

Online Subsystem Steam -rajapinta mahdollistaa Unreal Enginellä toteutetun verkkomoninpelin toimivan julkaisun Steam-myyntialustalle. Steam moduulin päätarkoitus on avata moninpeli toimintoja, jotka ovat yhteensopivia Steam-alustan kanssa, kuten matchmaking ja leaderboard. Online Subsystem Steam hyödyntää useita Online Subsystemissä olevia rajapintoja, kuten pilvipalvelut ja keskustelupalvelut. (Unreal Engine k.)

Steam tukee P2P-matchmakingiä lobbyjen avulla, jotka mahdollistavat sekä oman palvelimen, että kuuntelupalvelimen käytön. Lobbyt mahdollistavat käyttäjien nähdä pelipalvelimen tietoja, kuten mitä kenttää tai pelityyppiä pelataan tällä hetkellä ja kuinka monta käyttäjää kyseisellä palvelimella on, liittymättä itse palvelimelle. Lobbyt ovat periaatteessa Steamin backend palvelussa olevia P2P-keskusteluhuoneita. (Unreal Engine k.)

2.5 Verkkoarkkitehtuuri

2.5.1 Unreal Enginen verkkoarkkitehtuuri

Unreal Engine käyttää asiakas-palvelinarkkitehtuuria verkkomoninpeleissä. Verkkomoninpeleissä pelaajat muodostavat yhteyden omilla koneillaan yhteen koneeseen, jota kutsutaan palvelimeksi. Palvelimeen yhteyden muodostaneita koneita kutsutaan asiakasohjelmiksi. Palvelin jakaa informaatiota pelistä ja sen kulusta asiakasohjelmille ja tarjoaa mahdollisuuden eri koneilla pelaavien pelaajien väliseen kommunikaatioon. (Unreal Engine h.)

Asiakasohjelmien välillä voi olla erilainen tieto pelin tilasta kuin toisella. Tähän vaikuttaa asiakasohjelmien väliset erot yhteysnopeuksissa, sekä syötetyn datan häviäminen epävakaassa verkossa. Tämän takia palvelimella, joka isännöi (hostaa) peliä, on oletusarvoisesti oikea tieto pelin tilasta. (Unreal Engine h.)

Unreal Enginessä jokainen asiakasohjelma omistaa Pawn luokan olion, joka on olemassa palvelimella. Pawn luokka on kantaluokka kaikille Actoreille, joita joko pelaaja tai tekoäly voi kontrolloida. Asiakasohjelmilla on kontrolli Pawn oliosta. Toteuttaakseen tapahtumia pelissä, asiakasohjelmat lähettävät etäproseduurikutsuja omilta paikallisilta Pawn olioilta palvelimella sijaitsevaan Pawn olioon. Tämän jälkeen palvelin replikoi informaation pelin kulusta jokaiselle yhdistyneelle asiakasohjelmalle. Replikoitua informaatiota voi olla esimerkiksi Actorien sijainti tai ominaisuuksien arvo. Asiakasohjelmat hyödyntävät tätä informaatiota simuloidakseen mitä palvelimella tapahtuu. (Unreal Engine h.)

Verkkomoninpelissä samat pelitapahtumat tapahtuvat useassa eri pelimaailmassa, palvelimella sijaitsevassa, sekä jokaisessa yhdistyneen asiakasohjelman pelissä. Jokaisella pelimaailmalla on omat olionsa, kuten Actorit ja Pawnit. Palvelin on se, jossa itse peliä oikeasti pelataan, mutta jokaisen asiakasohjelman tulee replikoida palvelimella tapahtuvia pelitapahtumia tarkasti, jonka takia on tärkeä lähettää asiakasohjelmille oikeaa informaatiota, jotta ne voivat luoda tarkan kuvan palvelimella sijaitsevasta pelimaailmasta. (Unreal Engine h.)

2.5.2 Kuuntelupalvelinmalli

Kuuntelupalvelimet toimivat samanaikaisesti palvelimena, sekä asiakasohjelmana pelialustalla. Ne välittävät oman ympäristön ja resurssit pelaamiselle, eli yksi pelaaja isännöi ja pelaa peliä samanaikaisesti, samalla kuin muut verkkopelaajat liittyvät isännän palvelimelle. Kuuntelupalvelinmallin suurimmat hyödyt tulevat helposta asennuksesta ja alhaisista kustannuksista. Kuuntelupalvelinmallin käyttöön ei tarvita erillistä infrastruktuuria tai kalustoa. Kuuntelupalvelinmallilla on myös haittapuolensa, kuten epävakaa verkko, viive ja epäreilu peliympäristö, jossa isännällä on pääsy koko pelimaailman informaatioon. (Liquid Web.)

Oman palvelimen ja kuuntelupalvelimen suurimpina eroina on, että kuuntelupalvelin toimii käytännössä samalla tavalla kuin P2P-alusta, koska kuuntelupalvelimessa yksi pelaajista toimii isäntänä, toisin kuin oma palvelin, joka vaatii oman kaluston, eli erillisen PC:n, koska oma palvelin ei voi olla samalla koneella, jolla pelataan. (Liquid Web.)

Oma palvelin voittaa kuuntelupalvelimen suorituskyvyssä ja vakaudessa, koska omalla palvelimella ei ole samanaikaista resurssien käyttöä, vaan kaikki koneen suorituskyky ja verkko on käytettävissä kaikille liittyneille pelaajille. Kuuntelupalvelin on täysin riippuvainen isännän suorituskyvyn resursseista ja internetyhteydestä, jolloin isännällä ilmenevät ongelmat aiheuttavat ongelmia kaikille liittyneille pelaajille. (Liquid Web.)

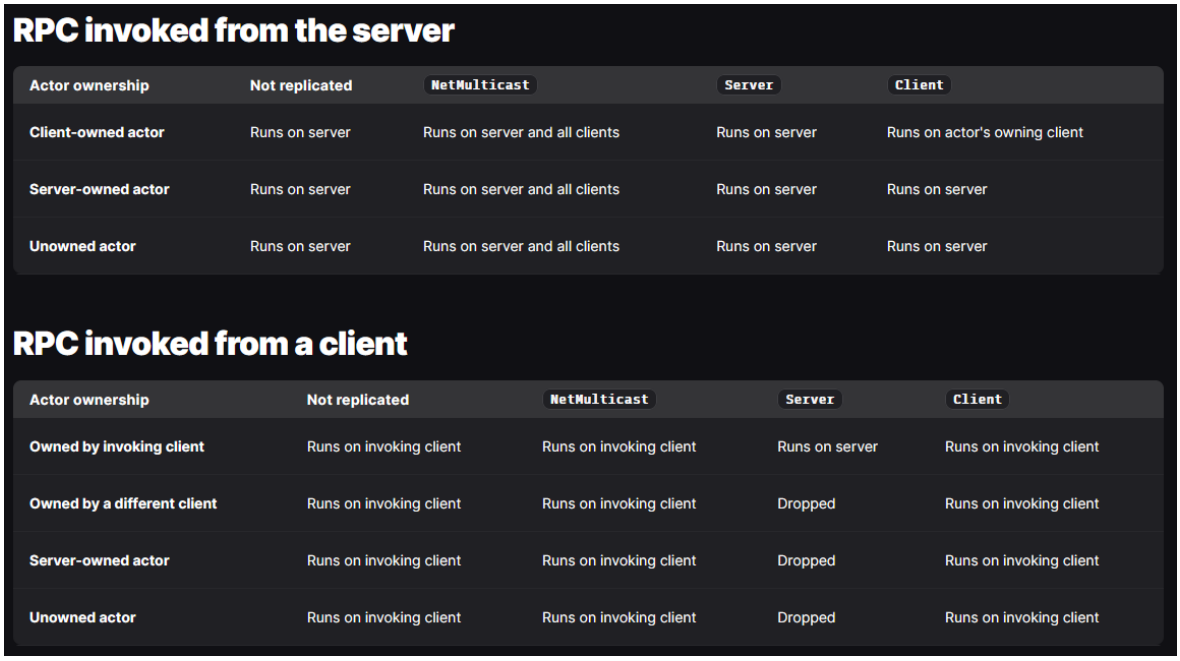
2.5.3 Etäproseduurikutsut

Etäproseduurikutsut ovat funktioita, joita kutsutaan paikallisesti ja suoritetaan toisella koneella. Etäproseduurikutsuja käytetään, kun halutaan lähettää viestejä asiakasohjelmalta palvelimelle, tai toisinpäin. (Unreal Engine i.)

Etäproseduurikutsujen pääasiallinen käyttötarkoitus Unreal Enginellä kehitettävään verkkomoninpeliin on epäluotettavien kosmeettisten tai ohimenevien -efektien kutsuihin. Näitä efektejä voi olla esimerkiksi äänitehosteiden toistaminen tai partikkelien luonti. Etäproseduurikutsuja käyttäessä on tiedettävä Actor luokkien omistajat, sillä se vaikuttaa siihen, millä asiakasohjelmalla etäproseduurikutsut suoritetaan. Unreal Enginessä PlayerController

luokka on se, joka viimekädessä määrittää omistajuuden. Omistajuuden tarkistus tapahtuu tietokantahaulla, joka hakee ylimmän omistajan. Ylin omistaja on luokka, jonka omistaja on joko palvelin tai asiakasohjelma. Jotta etäproseduuri on mahdollista suorittaa, ylimmän omistajan tulee olla PlayerController. Esimerkiksi Pawn luokka, jonka omistaja on PlayerController, jakaa omistajuuden kaikkien itse omistamiensa asioiden kanssa. (Unreal Engine i.)

Etäproseduurikutsujen käyttö tapahtuu kuvan 4 määrittämän omistajuuden mukaisesti, jossa näkyy etäproseduurikutsujen kutsuminen sekä palvelimelta, että asiakasohjelmalta. Kuvassa näkyy client, server ja NetMulticast tyyppiset etäproseduurikutsut ja miten ne käyttäytyvät Actoreissa. NetMulticast etäproseduurikutsu eroaa asiakasohjelman ja palvelimen kutsuista siten, että se kutsutaan sekä palvelimella, että kaikilla asiakasohjelmilla, kunhan sen kutsu tulee palvelimelta. NetMulticast kutsun käyttö on helppoa, mutta ongelmallista, koska kehittäjällä ei ole kontrollia siitä, milloin multicast kutsut käynnistetään. (Unreal Engine i.)



RPC invoked from the server

Actor ownership	Not replicated	NetMulticast	Server	Client
Client-owned actor	Runs on server	Runs on server and all clients	Runs on server	Runs on actor's owning client
Server-owned actor	Runs on server	Runs on server and all clients	Runs on server	Runs on server
Unowned actor	Runs on server	Runs on server and all clients	Runs on server	Runs on server

RPC invoked from a client

Actor ownership	Not replicated	NetMulticast	Server	Client
Owned by invoking client	Runs on invoking client	Runs on invoking client	Runs on server	Runs on invoking client
Owned by a different client	Runs on invoking client	Runs on invoking client	Dropped	Runs on invoking client
Server-owned actor	Runs on invoking client	Runs on invoking client	Dropped	Runs on invoking client
Unowned actor	Runs on invoking client	Runs on invoking client	Dropped	Runs on invoking client

Kuva 4. Etäproseduurikutsut palvelimelta ja asiakasohjelmalta (Unreal Engine i)

2.5.4 Replikointi

Replikoinnilla tarkoitetaan tilannetta, jossa palvelin lähettää dataa palvelimeen yhdistyneille asiakasohjelmille. Palvelimella on aina oikea tieto pelin tilasta, jonka takia yhdistyneet asiakasohjelmat kopioivat tämän tilan paikallisesti. (Unreal Engine h.)

Unreal Engine hoitaa yleisimmän osan replikoitavia ominaisuuksia automaattisesti. Näihin lukeutuvat olioiden luominen ja tuhoaminen, sekä pelaajien liikkuminen. Muut pelitapahtumat eivät replikoidu itsestään, vaikka Actor olio laitettaisiin replikoitavaksi. (Unreal Engine h.)

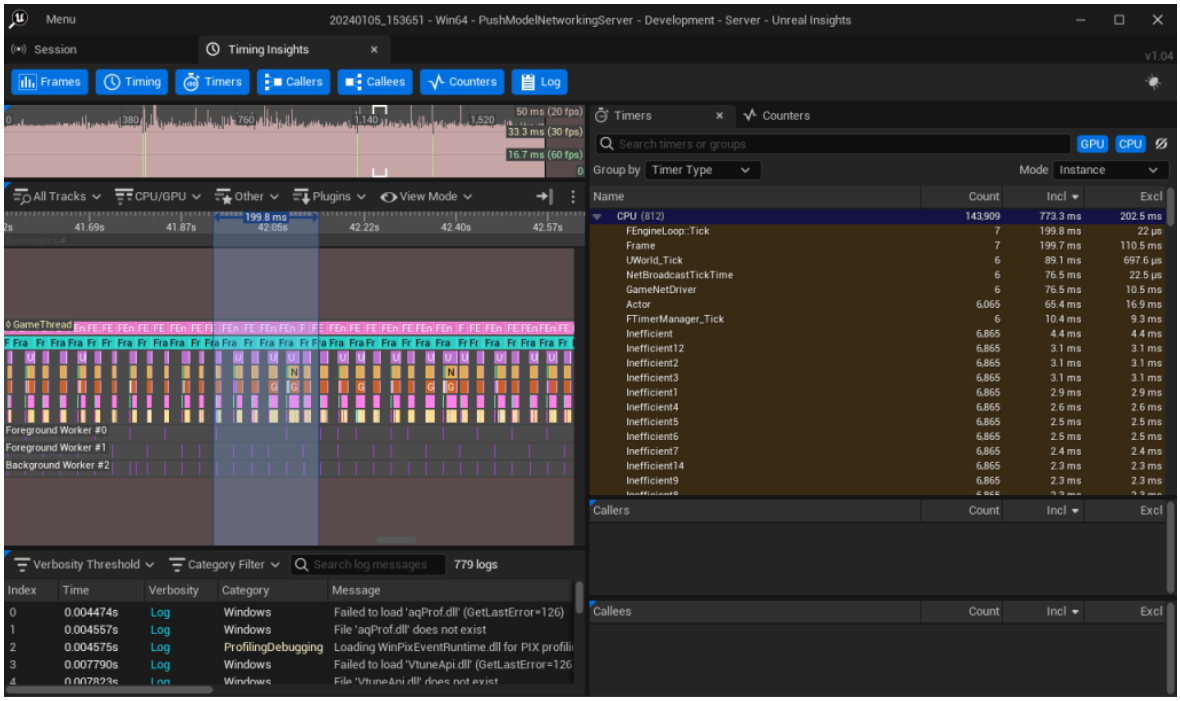
Unreal Enginessä jokaisella Actorilla on lista ominaisuuksia, jotka voidaan replikoida. Replikoidessa palvelin lähettää päivityksen jokaiselle asiakasohjelmalle, kun replikoidun ominaisuuden arvo muuttuu, joka mahdollistaa asiakasohjelman päivittää arvo paikalliseen versioon Actorista. Päivitykset tulevat aina palvelimelta, asiakasohjelmat eivät koskaan lähetä ominaisuus päivityksiä palvelimelle tai toisille asiakasohjelmille. Replikoidun muuttujan arvon muuttaminen asiakasohjelmalla ei ole hyvä käytäntö, koska arvo on eri kuin palvelimella, ennen kuin palvelin havaitsee muutoksen ja lähettää päivityksen.

Actorin ominaisuuksien replikointi on luotettavaa, joka tarkoittaa, että ominaisuuden arvo asiakasohjelmalla päivittyy samaksi kuin palvelimella oleva arvo. Ominaisuuksien arvon päivitysten tiheyttä on mahdollista muuttaa, sen mukaan kuinka tärkeä Actor on. Tällä voidaan saavuttaa tasaisempi pelikokemus varsinkin rajoitetuilla kaistanleveyksillä.

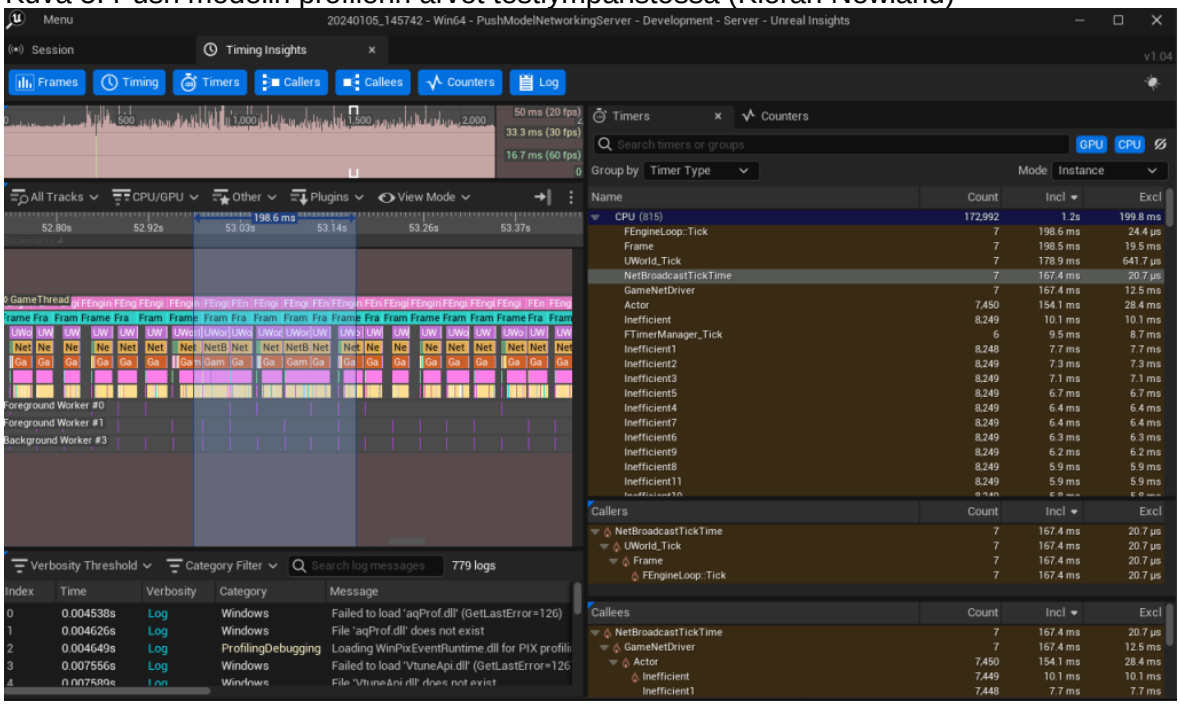
Rep notify on kutsu, jota voidaan kutsua, kun replikoidun ominaisuuden arvo muuttuu. Rep notify käyttäytyy samankaltaisesti kuin multicast etäproseduurikutsu, mutta se on luotettava, eli se suoritetaan heti, kun ominaisuuden arvo on muutettu. Blueprint-ohjelmoinnissa rep notify käyttäytyy eri tavalla kuin C++-ohjelmoinnissa. C++-ohjelmoinnissa rep notifyä kutsuttaessa, se käynnistetään vain asiakasohjelmilla, kun taas blueprint-ohjelmoinnissa rep notify kutsu käynnistetään myös palvelimella.

Unreal Enginessä uudempi tapa käyttää rep notify kutsua on käyttää push modelia, jolloin kehittäjän vastuulla on itse merkata ominaisuuden arvo muutettavaksi. Tämä tapahtuu merkkaamalla muutettu ominaisuus likaiseksi (dirty). Tällä mallilla voi säästää resursseja, koska pelimoottorin ei tarvitse tehdä jatkuvia vertailu tarkastuksia ominaisuuden arvojen välillä. Kuvassa 5 ja 6 on verrattu push modelin ja tavallisen replikoinnin nopeutta testiympäristössä. Testiympäristönä on palvelin ja 4 yhdistyvää asiakasohjelmaa, joista jokainen luo 20 Actoria, joilla on yksi replikoitava ominaisuus. Profilerin otanta on seuraavat 200ms sen jälkeen, kun viimeinen asiakasohjelma on yhdistynyt. Kuten kuvien vertailusta on nähtävissä NetBroadcastTickTime arvo on noin 50 % pienempi push modelilla verrattuna

tavalliseen replikointiin. Testiympäristön esimerkki on liioiteltu, mutta antaa hyvän kuvan siitä, millaista säästöä push modelilla on mahdollista saavuttaa. (Kieran Newland.)



Kuva 5. Push modelin profilerin arvot testiympäristössä (Kieran Newland)



Kuva 6. Tavallisen replikoinnin arvot testiympäristössä (Kieran Newland)

3 Verkkomoninpelin toteutus

3.1 Toteutuksen suunnittelu

Toimeksiantona oli suunnitella ja toteuttaa verkkomoninpeli pohja, joka olisi mahdollisimman helppokäyttöinen, ja toimisi videopelin tarpeiden mukaisesti. Alkuun ideana oli käyttää Epic Online Services palvelua, mutta koska peli olisi tarkoitus julkaista vain Steam myyntialustalle, ja koska ristiinpelaamis mahdollisuuksia, tai muita Epic Online Servicen erikoisominaisuuksia, ei ole tarkoitus käyttää, niin Epic Online Servicesin käyttö ei ollut perusteltua. Toteutukseen valittiin siis Unreal Enginen Online Subsystem liitännäinen, sekä kyseisen liitännäisen Steam lisäosa. Online Subsystem on Epic Online Serviceä rajoittuneempi, mutta tehokkaampi. Sen lisäksi se vaatii vähemmän konfigurointia, sekä sen tarjoamat palvelut riittävät hyvin toteutuksen tekemiseen.

Verkkomoninpeli kehyksen tulisi toimia 2–4 pelaajalle samanaikaisesti, eikä pelin tarkoitus ole olla kilpailullinen, vaan enemminkin hauskaa ajanvietettä. Sen lisäksi julkaistun pelin ylläpitoon käytettävät kustannukset tulisi olla mahdollisimman pienet, joten kuuntelupalvelimen käyttö oli järkevin vaihtoehto, jossa omien maksullisten palvelimien sijaan käytettäisiin isäntä pelaajan konetta palvelimena. Vaihtoehto olisi epäreilu, mikäli peliin olisi suunniteltu kilpailuelementtiä, sillä kuuntelupalvelimen verkon vakaus on täysin isännän verkon vakaudesta kiinni, sekä isännällä itsellään olisi aina 0ms ping pelissä. Pelissä ping tarkoittaa sitä aikaa, joka kuluu datan kulkuun asiakasohjelman ja palvelimen välissä.

Itse kehyksen toteutukseen tarvittiin C++- ja blueprint -ohjelmointia, sekä jonkun verran pelimoottorin toimintaa kontrolloivien tiedostojen muuttamista. Verkko-ominaisuuksien backend kehitettiin C++-ohjelmointikielellä ja pelielementit toteutettiin blueprint-ohjelmoinnilla, joitakin peliominaisuuksia tehtiin myös C++-ohjelmointikielellä, kun elementtien käyttö on toistuvaa, kuten pelissä ampuminen. Blueprint-ohjelmointia hyödynnettiin myös toimivien prototyyppien kokeilemiseen ennen niiden C++ ohjelmointia.

Tarvittavia verkossa toimivia komponentteja olivat matchmaking järjestelmä, pelaajien ja ottelun toiminnot pelissä, sekä osa käyttöliittymä elementeistä. Matchmakingin luonnissa hyödynnettiin Online Subsystem liitännäisen rajapintaa, joka tarjoaa tarvittavan funktionaalisuden matchmakingin luontiin. Pelaajien toimintoihin Unreal Engine pelimoottorista löytyi suora vastaus osaan ongelmista, kuten pelaajien liikkumiseen sekä animointiin, sillä nämä toiminnot saatiin toimimaan helposti Actorin ja sen omistamien komponenttien replicate arvon avulla. Muita pelaaja toimintoja, kuten ampuminen ja pelaajien erikoisominaisuudet oli tehtävä itse hyödyntäen etäproseduurikutsuja ja replikointia. Tarvittavia luotavia käyttöliittymä elementtejä oli muun muassa verkon ylitse välittyvät tiedot pelaajan elämäpisteistä ja

kestävyydestä, sekä kaikille näkyvä yhden pelaajahahmon parannus valikko, jossa sitä käyttävän pelaajan valinnat näkyisivät muille pelaajille.

3.2 Matchmaking

Ensimmäinen asia, jota lähdettiin toteuttamaan, oli matchmakingiin vaadittava session aloittaminen ja toisten pelaajien sessioon liittyminen. Tässä käytettiin Online Subsystemin rajapintaa, josta löytyy tarvittava funktionaalisuus. Toimiakseen oikein Unreal Enginessä, Online Subsystem vaatii muutoksia kahteen pelimoottorin omaan tiedostoon, konfiguraatiota ja buildaamista varten. Isännöinnin ja liittymisen toteuttamiseen tehtiin luokka OnlineGameInstance, joka perittiin Unreal Enginen tarjoamasta GameInstance luokasta. GameInstance luokka sisältää luotavalle komponentille tarpeellista funktionaalisuutta, kuten asiakasohjelman siirtämistä isännän mukana lobbysta pelattavaan kenttään, sekä verkkovirheiden käsittelyä. Itse isännöinnin ja liittymisen luontiin tarvittiin 3 delegaattia, session luonti, session etsiminen ja sessioon liittyminen. Sen lisäksi tarvittiin funktio itse palvelimen luontiin.

Palvelimen ja session luonti toteutui Online Subsystemin Session rajapinnan avulla, josta löytyi valmiit funktiot näiden luontiin, ja joille tarvitsi antaa vain tarvittavat asetukset niiden luontiin. Palvelimelle tarvitsi asettaa nimi, maksimi pelaaja määrä, sekä onko palvelin LAN vai verkkopohjainen. Session luontiin tarvitsi luoda sessioasetukset, johon tarvitsi palvelimelta saatavan tiedon lisäksi, tiedon käytetäänkö lobbyjä, sekä tietoa verkkonäkyvyyden mainostamisesta. Palvelimen luonnin toteuttava funktio on nähtävissä kuvassa 7.

```
void UOnline_GameInstance::CreateServer(FCreateServerInfo ServerInfo)
{
    UE_LOG(LogTemp, Warning, TEXT("Creating server"));

    FOnlineSessionSettings SessionSettings;
    SessionSettings.bAllowJoinInProgress = true;
    SessionSettings.bIsDedicated = false;

    UE_LOG(LogTemp, Warning, TEXT("Subsystem in use: %s"), *IOnlineSubsystem::Get()->GetSubsystemName().ToString());
    if (IOnlineSubsystem::Get()->GetSubsystemName() != "NULL")
    {
        SessionSettings.bIsLANMatch = false;
    }
    else
    {
        SessionSettings.bIsLANMatch = true;
    }
    SessionSettings.bShouldAdvertise = true;
    SessionSettings.bUsesPresence = true;
    SessionSettings.NumPublicConnections = ServerInfo.MaxPlayers;

    SessionSettings.BuildUniqueId = 1;
    SessionSettings.bUseLobbiesIfAvailable = true;

    SessionSettings.Set(FName("SERVER_NAME_KEY"), ServerInfo.ServerName, EOnlineDataAdvertisementType::ViaOnlineServiceAndPing);

    SessionInterface->CreateSession(0, MySessionName, SessionSettings);
}
```

Kuva 7. Session luonti

Sessioiden etsimiseen ja niihin liittymiseen tarvitsi tehdä tietokantahaku, jolla löydettiin kaikki luodut sessiot, jotka voidaan listata pelinäkymässä ja mahdollistaa niihin liittyminen listan kautta. Tietokantahaussa ilmeni ensimmäinen ongelma, koska peliä tehdessä käytettiin Steamin kehittäjäpalvelun DevAppID:tä, jonka takia tietokantahaku löysi sessioita kaikista samaa kehittäjien DevAppID:tä käyttävistä sovelluksista, jonka takia itse luotujen sessioiden löytäminen ei onnistunut. Tähän löydetty ratkaisu oli asettaa sessioita luodessa oma uniikki ID, siihen asti, että yhtiö sai Steamilta oman kehittäjä ID:n. Ongelman ratkaisun jälkeen sessioiden listaus toimi. Sessioon liittyminen toteutettiin sessioiden tietokantahaun antamilla tuloksilla, liittymällä haluttuun sessioon.

Isännän ja asiakasohjelman siirto pelattavaan kenttään toteutettiin automaattisesti, kun palvelimelle on liittynyt asetettu maksimipelaaja määrä. Tämä toteutettiin pääasiallisesti testaamisen takia, koska mitään käyttöliittymää itse lobbylle ei ollut tehtynä. Toiminnallisuuden siirto käyttöliittymään sen tullessa valmiiksi vaatii vain yhden funktion muuttamista siten, että funktiota kutsutaan käyttöliittymästä, C++ automaation sijaan.

3.3 Pelaajan toiminnot

Seuraavaksi toteutettava asia oli pelaajien toiminnot, joista pelaajan liikkuminen, sekä liikkumisen animointi verkon yli toteutui Unreal Enginen oman replikoinnin avulla. Tärkein pelaajan toiminto ammunta pelissä liikkumisen jälkeen on ampuminen, jota lähdettiin toteuttamaan C++-ohjelmointikielellä käyttäen etäproseduuri kutsuja, koska ammukset lukeutuvat ohimeneviksi efekteiksi pelissä. Ampumista ja muuta pelaajan funktionaalisuutta varten luotiin Unreal Enginen Character luokasta peritty OnlineCharacter luokka. Character luokan käyttö oli mahdollista, sillä PlayerController omistaa Characterin ja mahdollistaa siksi verkkotoimintojen käytön. Ampumiselle tehtiin OnFire funktio, jota voitaisiin kutsua blueprint-komponentissa. OnFire funktiossa luodaan ammus paikallisesti, jonka jälkeen kutsutaan joko ServerOnFire funktiota, tai MultiOnFire funktiota riippuen siitä, kutsutaanko OnFire funktiota asiakasohjelmalta vai palvelimelta, eli isännältä. ServerOnFire funktio kutsuu myös MultiOnFire funktiota, mutta ServerOnFire funktiossa on käytössä Server makro, koska etäproseduurikutsut täytyy kutsua asiakasohjelmalta palvelimelle. MultiOnFire funktiossa, jossa käytössä on NetMulticast makro, luodaan ammus. Ennen ammuksen luontia tarkastetaan, kutsutaanko funktiota samalta ohjelmalta, jolla ammus oli jo luotu paikallisesti. Tämän avulla voidaan välttää useamman ammuksen luontia samalla ohjelmalla.

Ammuntaan liittyvää funktionaalisuutta on myös staminan, eli jaksamispisteiden, käyttö ja pelaajien elämäpisteet, joiden funktionaalisuudessa käytettiin replikointia. Näiden toteuttamiseen käytettiin push model tyyppistä rep notify replikointia, jossa SetHealth ja SetStamina ovat blueprint-komponentissa kutsuttavia funktioita, jotka asettavat elämäpiste ja stamina

muuttujille uudet arvot. Ennen arvojen muuttamista tehdään tarkistus siitä, onko ohjelmalla auktoriteetti, jolla varmistetaan, että arvojen muuttaminen tapahtuu palvelimella. Tämän jälkeen arvo merkataan likaiseksi, joka antaa palvelimelle tiedon siitä, että muutetut arvot tulisi lähettää asiakasohjelmille. Arvojen muuttamisen jälkeen suoritetaan OnRep kutsut, joita voidaan kuunnella blueprint-komponenteissa, hyödyntäen niitä esimerkiksi audion tai visuaalisten efektien toistamiseen. Tämä on nähtävissä kuvan 8 toteutuksessa.

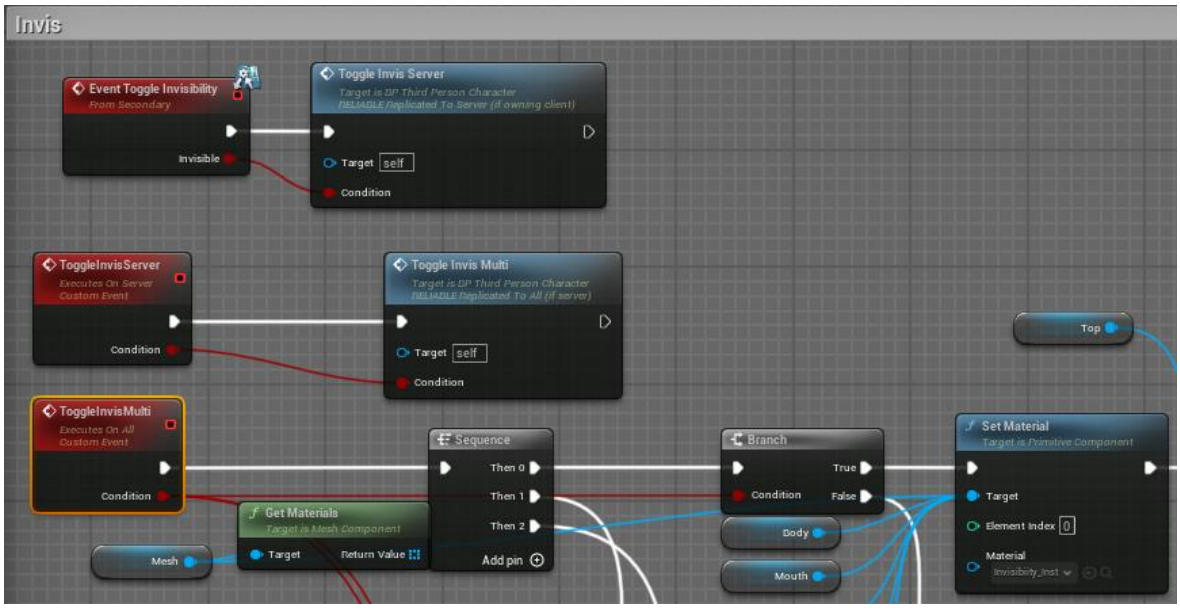
```
void AMultiplayerCharacter::SetStamina(float amount)
{
    if (HasAuthority())
    {
        Stamina = amount;

        MARK_PROPERTY_DIRTY_FROM_NAME(AMultiplayerCharacter, Stamina, this);
    }
}

void AMultiplayerCharacter::GetLifetimeReplicatedProps(TArray<FLifetimeProperty>& OutLifetimeProps) const
{
    Super::GetLifetimeReplicatedProps(OutLifetimeProps);

    //DOREPLIFETIME(AMultiplayerCharacter, Health);

    FDoRepLifetimeParams SharedParams;
    SharedParams.bIsPushBased = true;
    SharedParams.RepNotifyCondition = REPNOTIFY_Always;
    DOREPLIFETIME_WITH_PARAMS_FAST(AMultiplayerCharacter, Health, SharedParams);
    DOREPLIFETIME_WITH_PARAMS_FAST(AMultiplayerCharacter, Stamina, SharedParams);
}

void AMultiplayerCharacter::OnRep_SpendStamina()
{
    UE_LOG(LogTemp, Warning, TEXT("On rep Spend Stamina, Stamina: %d"), int(Stamina));
}
```

Kuva 8. Push model rep notify

Pelaajahahmoille luotiin myös struct tyyppinen tietorakenne, joka sisältää useita eri pelinaikana käytettäviä tilasto elementtejä, kuten ammunta- ja latausnopeus, hahmon liikkeiden nopeus, sekä muita vastaavia. Tietorakenteesta perittiin pelimoottorin sisällä Blueprint Function Library, joka asetettiin replikoitumaan. Replikoitu Blueprint Function Library toimii samalla tavalla, kuin mikä tahansa replikoitu ominaisuus, jonka lisäksi siinä voi olla mukana funktioita, jonka avulla saatiin tietorakenteessa oleville arvoille get ja set funktiot toteutettua.

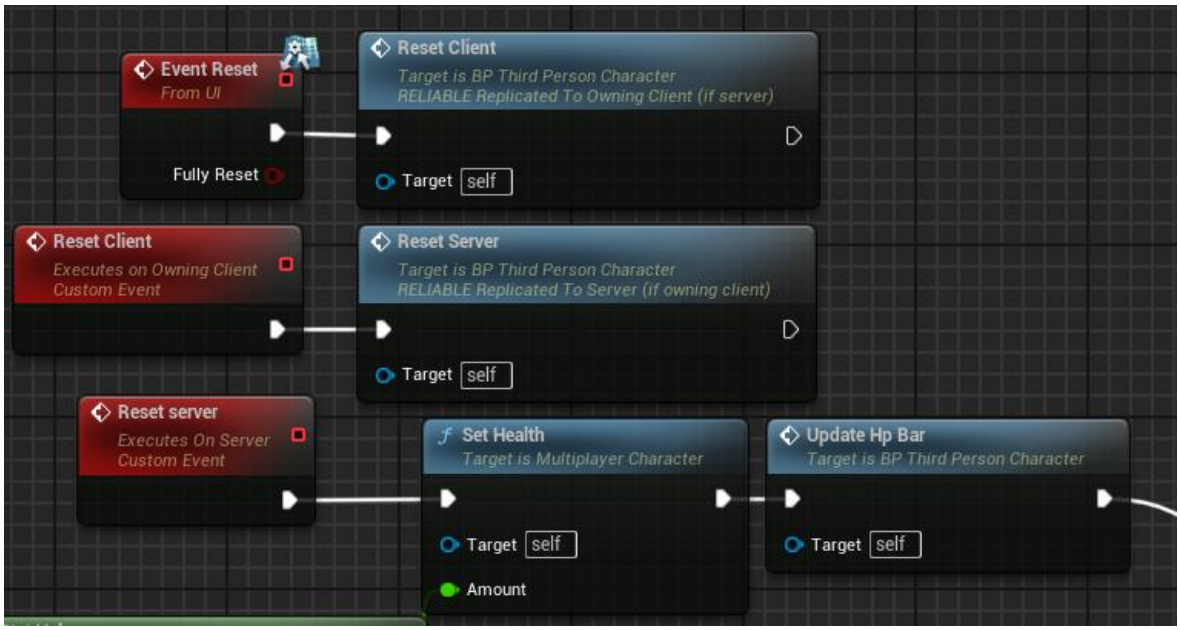
Lisäksi pelaajille tehtiin erikoiskykyjä, kuten väliaikainen näkymättömyys, jossa pelaajahahmon materiaalien shaderia muutettiin. Näiden erikoiskykyjen toteutus tehtiin kokonaan blueprint-komponentissa, koska suurin osa funktionaalisuutta oli tehty valmiiksi paikallisesti, joten näiden verkkototeutuksen sai helposti tehtyä blueprint-komponentin sisällä. Näihin käytettiin etäproseduurikutsuja, jotka blueprint-työkalussa tehdään custom tapahtumilla. Rakenne menee siten, että ensin on luotava paikallinen tapahtuma, joka kutsuu vastaavaa tapahtumaa palvelimelta, joka kutsuu joko asiakasohjelma tai

NetMulticast -tapahtumaa, jossa itse tapahtuman funktionaalisuus toteutetaan. Kuvassa 9 on toteutettu näkymättömyyskyvyn funktionaalisuus blueprint-komponentissa, jossa paikallinen tapahtuma Event Toggle Invisiblity kutsuu ToggleInvisServer palvelin tapahtumaa, joka kutsuu vastaavaa ToggleInvisMulti NetMulticast tapahtumaa, jossa itse näkymättömyyden funktionaalisuus on toteutettu.



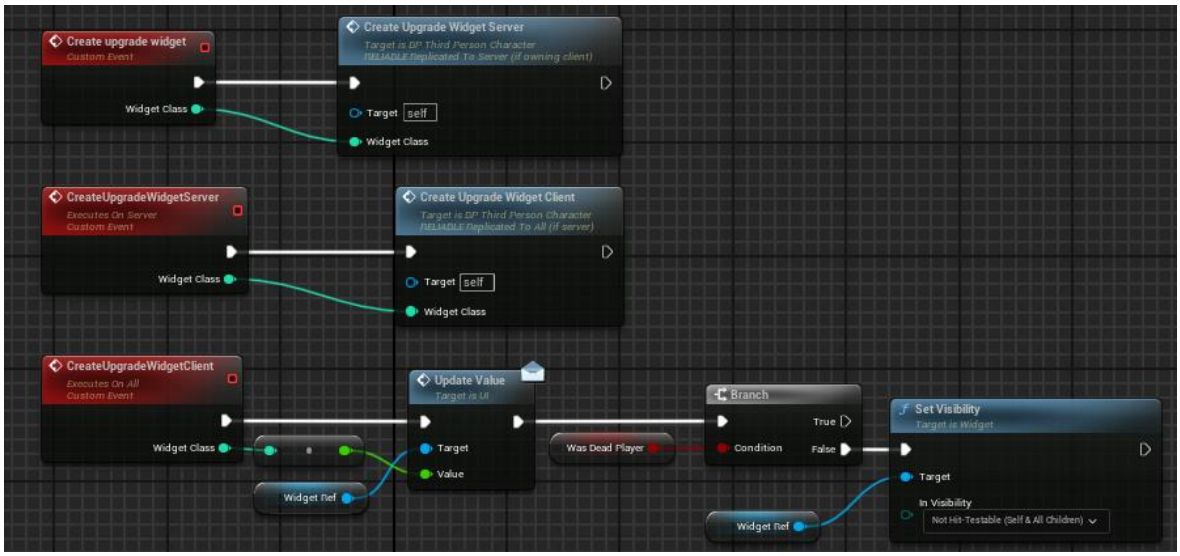
Kuva 9. Näkymättömyyskyvyn toteutus blueprint-työkalulla

Myös muut samanlaiset erikoiskyvyt toteutettiin kuvan mukaisella tavalla. Ainut poikkeava etäproseduurikutsuketju tuli ottelun päättymisestä, jossa pelaajat siirretään takaisin aloituspaikoille, sekä asetetaan elämäpisteiden ja staminan arvot takaisin aloitusarvoiksi. Tässä tapauksessa palvelimelta asiakasohjelmalle tuleva kutsu ei toiminut oikein, vaan oikea ratkaisu oli kutsua asiakasohjelmalta palvelimelle, joka suorittaa tapahtuman funktionaalisuuden, kuten kuvassa 10 on toteutettu.



Kuva 10. Uuden ottelun aloitus, edellisen päätyttyä

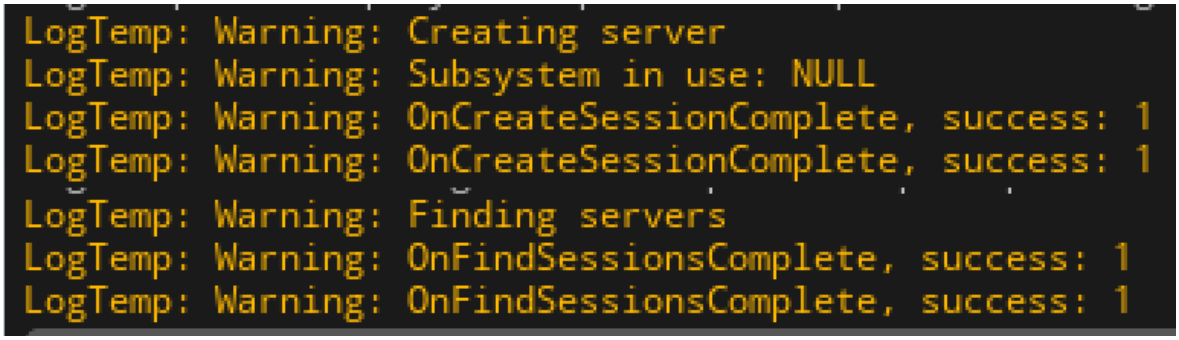
Myös käyttöliittymäelementtien toteutus tehtiin blueprint-ohjelmoinnin etäproseduurikutsuilla, jotka täytyi kutsua Character blueprintistä. Luoduilla käyttöliittymä elementeillä, eli Unreal Enginen UI Widgeteillä, ei ole omistajia, vaan ne ovat olemassa vain paikallisesti, jonka takia niihin ei pysty tekemään mitään verkkokutsuja. Käyttöliittymäelementtien luonti tapahtui samalla tavalla, kuin pelaajan erikoiskykyjen toteutus, jossa UI Widget luotiin multicast funktiossa. Pelihahmon ominaisuuksien kehittämiselle tehtyä käyttöliittymäelementtiä varten tarvittiin tieto kuolleesta pelaajasta, jotta kontrolli hahmon kehityksen käyttöliittymästä voidaan antaa oikealle pelaajalle. Tätä varten otettiin referenssi kuolleeseen pelaajahahmoon elämäpisteiden vähentyessä nollaan, ja replikoitiin referenssin arvo. Kuvassa 11 on pelaajahahmon kehityksen käyttöliittymä elementin luonti Widget Class referenssin pohjalta, jossa otetaan kontrolli käyttöliittymästä pois kaikilta paitsi kuolleelta pelaajalta NetMulticast kutsussa.



Kuva 11. Hahmonkehitys käyttöliittymän luonti

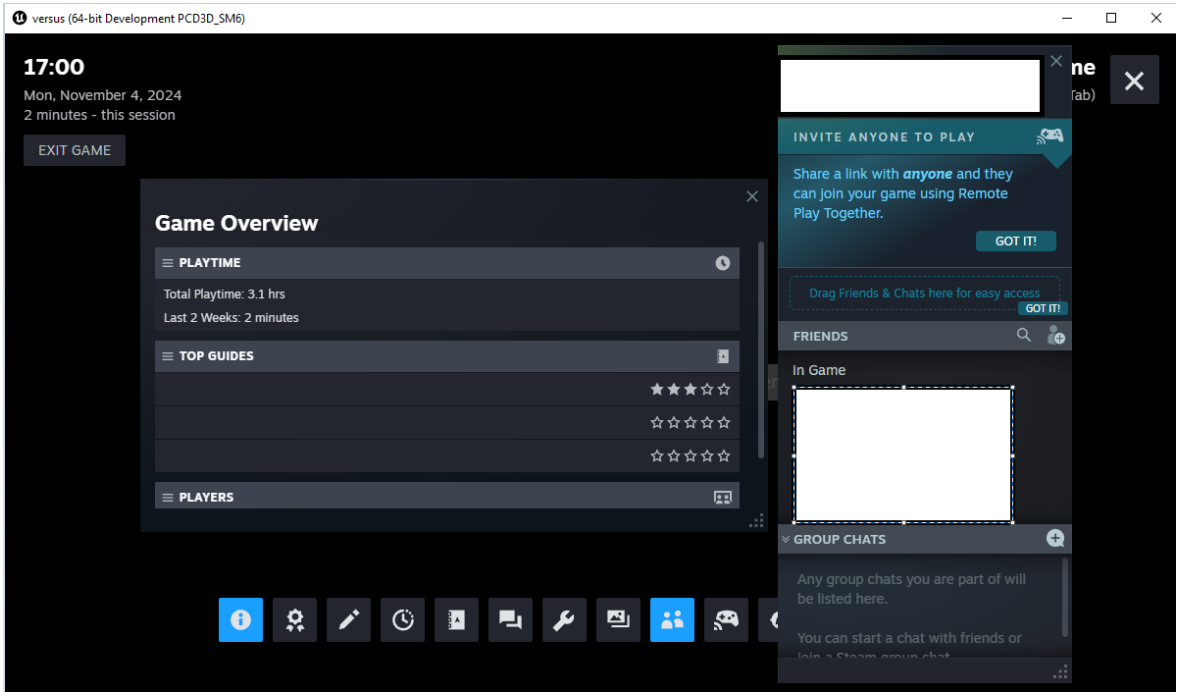
4 Toimivan toteutuksen testaus

Kaikkien pyydettyjen toimintojen valmiiksi toteuttamisen jälkeen voitiin aloittaa toteutuksen testaaminen. Ensimmäinen testaus tehtiin matchmaking järjestelmässä, testaamalla lobbyjen toimivuus, eli testattiin sitä, että isäntä kykeni luomaan session ja asiakasohjelma pystyi löytämään, sekä liittymään sessioon. Ensimmäinen testi toteutettiin Unreal Enginen sisällä, käyttäen play as listen server toimintoa, joka luo 2 ohjelmaa, joista toinen simuloi kuuntelupalvelinta ja toinen asiakasohjelmaa. Session luonti ja löytäminen molemmat toimivat, kuten kuva 12 osoittaa. Kuvassa oleva Subsystem NULL tarkoittaa sitä, että mihinkään Online Subsystem liitännäiseen ei saatu yhteyttä, koska palvelin ja asiakasohjelmat ovat samassa LANissa, miten sen pelimoottorin sisällä kuuluukin olla. Kuvassa olevat muut kirjaukset viittaavat palvelimen luonnin, session luonnin, sekä session löytämisen onnistumiseen.



Kuva 12. Session luonti ja liittyminen

Seuraavaksi testattiin Steam-alustan yhteensopivuus ja jotta tätä voitiin testata, applikaatio täytyi buildata. Tämäkin testi onnistui, sillä Steam overlay toimi testauksessa, kuten kuvassa 13 näkyy. Tämän jälkeen voitiin testata Steam verkkomoninpelin toimiminen. Tätä testiä varten avattiin applikaatio PC:llä ja isännöitiin sillä sessio, jonka lisäksi toimittiin asiakasohjelmana toisella PC:llä, joka oli kytketty eri verkkoon kuin isännän PC. Tässä tapauksessa hyödynnettiin puhelimen verkon mobiilitukiasemaa. Toisen PC:n täytyi olla kirjautuneena myös eri Steam tunnuksella kuin isäntä, koska sama Steam profiili ei toimi samanaikaisesti isäntänä ja asiakasohjelmana.



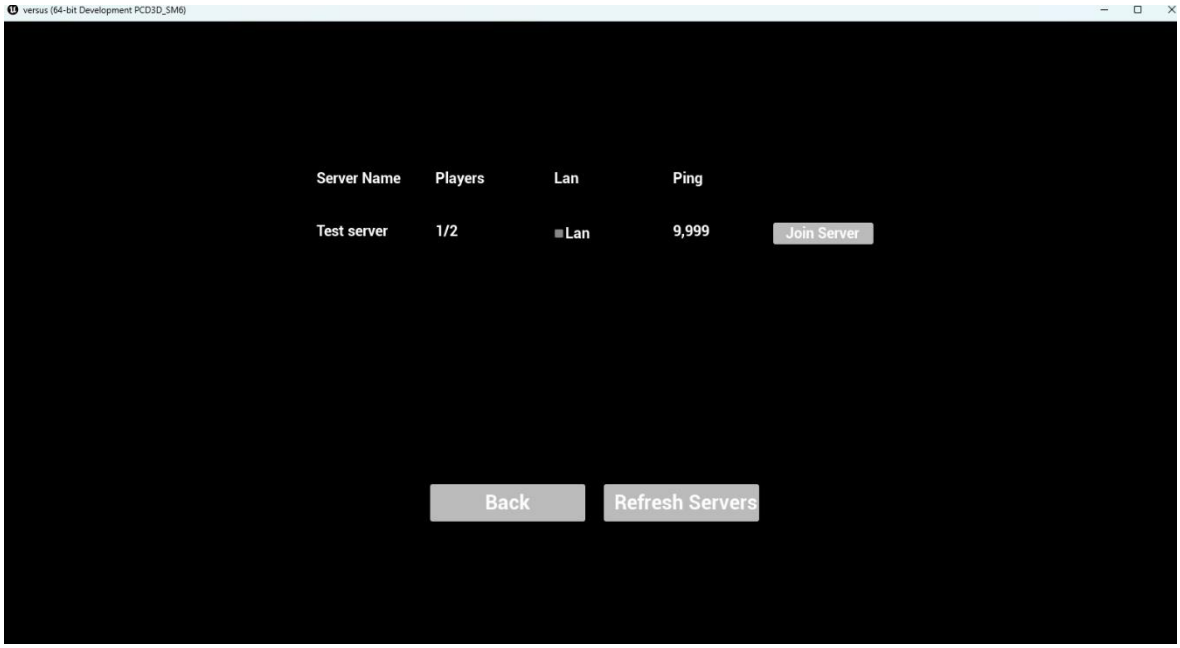
Kuva 13. Pelin Steam overlay

Isännällä session luominen onnistui, kuten kuvan 14 loki osoittaa. Test server nimellä löytyvä sessio saatiin luotua, kuten kuvan toisiksi viimeisen rivin kirjaus osoittaa. Sessio luotiin käyttäen Subsystem Steamia, joka on näkyvissä kuvan lokin toisella rivillä.

```
[793]LogTemp: Warning: Creating server
[793]LogTemp: Warning: Subsystem in use: STEAM
[840]LogOnlineSession: Verbose: OSS: dumping NamedSession:
[840]LogOnlineSession: Verbose: OSS:   SessionName: My Session
[840]LogOnlineSession: Verbose: OSS:   HostingPlayerNum: 0
[840]LogOnlineSession: Verbose: OSS:   SessionState: Pending
[840]LogOnlineSession: Verbose: OSS:   RegisteredPlayers:
[840]LogOnlineSession: Verbose: OSS:       0 registered players
[840]LogOnlineSession: Verbose: OSS: dumping Session:
[840]LogOnlineSession: Verbose: OSS:   OwningPlayerName: [redacted]
[840]LogOnlineSession: Verbose: OSS:   OwningPlayerId: [redacted]
[840]LogOnlineSession: Verbose: OSS:   NumOpenPrivateConnections: 0
[840]LogOnlineSession: Verbose: OSS:   NumOpenPublicConnections: 1
[840]LogOnlineSession: Verbose: OSS:   SessionInfo: HostIP: INVALID SteamP2P: 76561198035557718:7777
[840]LogOnlineSession: Verbose: OSS: dumping SessionSettings:
[840]LogOnlineSession: Verbose: OSS:   NumPublicConnections: 2
[840]LogOnlineSession: Verbose: OSS:   NumPrivateConnections: 0
[840]LogOnlineSession: Verbose: OSS:   bIsLanMatch: false
[840]LogOnlineSession: Verbose: OSS:   bIsDedicated: false
[840]LogOnlineSession: Verbose: OSS:   bUsesStats: false
[840]LogOnlineSession: Verbose: OSS:   bShouldAdvertise: true
[840]LogOnlineSession: Verbose: OSS:   bAllowJoinInProgress: true
[840]LogOnlineSession: Verbose: OSS:   bAllowInvites: false
[840]LogOnlineSession: Verbose: OSS:   bUsesPresence: true
[840]LogOnlineSession: Verbose: OSS:   bAllowJoinViaPresence: false
[840]LogOnlineSession: Verbose: OSS:   bAllowJoinViaPresenceFriendsOnly: false
[840]LogOnlineSession: Verbose: OSS:   BuildUniqueId: 0x01f83457
[840]LogOnlineSession: Verbose: OSS:   Settings:
[840]LogOnlineSession: Verbose: OSS:           SERVER_NAME_KEY=Test server : OnlineServiceAndPing
[840]LogTemp: Warning: OnCreateSessionComplete, success: 1
```

Kuva 14. Isännöintiin käytetyn koneen sessioiden luonnin loki

Kuva 15 on otettu asiakasohjelmalta, jolla etsitään sessioita, ja löytynyt Test server niminen sessio osoittaa, että verkkomoninpelin isännöinti, sessioiden etsiminen ja sessioihin liittyminen toimi vaaditulla tavalla. Testauksen helpottamiseksi pelaajien maksimilukumääräksi oli asetettu 2, jotta kenttään siirtyminen tapahtuu ensimmäisen asiakasohjelman liittymisen jälkeen. Kuvassa näkyvä ping arvo on tietokantahaun antama maksimiarvo, sillä ping arvon hakua ei ollut testivaiheessa implementoitu. Se olisi vaatinut erillistä konfigurointia sessioita käyttäessä, eikä sen tekeminen ollut prioriteetti. Myös maksimipelaajamäärän täyttyessä liittymisen jälkeinen isännän ja asiakasohjelman siirto pelattavaan kenttään toimi.



Kuva 15. Asiakasohjelman sessioiden etsimiseen tehty käyttöliittymä

Seuraavaksi testattavana oli pelaajan toiminnot verkossa. Pelaajan animaatioissa ja liikkumisessa ei ilmennyt ongelmia, joka oli oletettavissa, koska pelimoottorin sisäiset verkkotoiminnot hoitivat ne. Ampumista testattaessa ammuttavan pelaajan elämäpisteet vähenivät epäluotettavasti, eli joissain tapauksissa asiakasohjelmalta ammuttaessa isäntä pelaajan elämäpisteet vähenivät kaksinkertaisen määrän. Ongelma löydettiin siitä, että luotavan ammuksen Actorissa oli laitettu replikointi päälle pelimoottorin sisällä, samanaikaisesti, kun C++ koodissa ammusta luodessa käytettiin etäproseduriikutsuja, jonka takia asiakasohjelmalla luotiin kaksi ammusta samanaikaisesti. Muita ongelmia pelaaja toimintoja testatessa ei havaittu. Tilastopisteiden tietorakenteen, sekä ammusten ja staminan arvot replikoituivat oikein. Pelaajan kuolema laukaisi ottelun päättymisen tapahtuman liittyneille pelaajille, sekä avasi oikean käyttöliittymän pelaajille. Käyttöliittymän pelaajan valinnat näkyivät oikein muille pelaajille, mutta kontrolli käyttöliittymässä oli aluksi kaikilla pelaajilla. Ongelma johtui NetMulticast etäproseduurikutsun käytöstä, sillä se antoi epäluotettavia arvoja kuolleen pelaajan referenssistä. Tämä saatiin korjattua asettamalla auktoriteetti kysely NetMulticast kutsuun, ennen kuolleen pelaajan referenssin asettamista, jolloin pelkästään palvelimella oli oikeus asettaa replikoitavan arvon referenssistä jokaiselle.

5 Yhteenveto ja pohdinta

Tavoitteena oli toteuttaa toimiva verkkomoninpelin kehys, jota muut kehittäjät voivat hyödyntää pelin valmiiksi saamisessa. Tavoitteeseen päästiin pienistä ongelmista huolimatta. Myös muut tavoitteet, eli Steam alustan yhteensopivuus ja kehyksen toimivuus reaaliaikaisessa ammuntapelissä, saatiin toteutettua. Tavoitteisiin pääseminen oli joiltain osin hankalaa, sillä Unreal Enginen, ja etenkin pelimoottorin verkkomoninpeli liitännäisten, dokumentaatio oli paikoin puutteellista, jonka takia joitakin asioita piti selvittää testaamalla. Pelimoottorin Online Subsystem liitännäiset olivat kuitenkin monipuoliset. Niiden rajapinnoista löytyi kaikki tarpeellinen isommankin verkkomoninpeli toteutuksen kehittämiseen, vaikka suurinta osaa liitännäisen tarjoamista palveluista ei tässä projektissa tarvinnut käyttää. Ilman pelimoottorin verkkomoninpeli liitännäisiä verkkomoninpeli kehyksen toteuttaminen ei olisi onnistunut yhdeltä kehittäjältä. Vaikeuksista huolimatta verkkomoninpelin kehittäminen Unreal Enginellä oli helpompaa, kuin esimerkiksi aikaisemmin kerätty kokemus verkkomoninpelin kehittämisestä Unity pelimoottorilla. Uskon tämän johtuvan siitä, että Unreal Engine pelimoottori on valmiiksi kehitetty verkkomoninpelien kehittämiseen, sillä ensimmäinen sillä julkaistu peli oli verkkomoninpeli Unreal.

Tällä hetkellä verkkomoninpelin kehys on käytössä Ape Brain Gamesin tulevan pelin kehityksessä, ja on toiminut suurimmilta osin. Joitakin muutoksia ja uudistuksia on pelin kehityksen edetessä joutunut tekemään, ja todennäköisesti näin tarvitsee myös jatkossa tehdä. Matchmaking järjestelmästä on myös tullut kehua, että se on toiminut moitteettomasti.

Kehityksen aikana, ja sen jälkeen, opin Unreal Enginen verkkomoninpelin toteuttamisesta ja käyttöönotosta huomattavasti, josta syystä myös mahdollisia jatkokehitys kohteita ilmeni. Esimerkiksi käyttöliittymän elementtien ohjelmointiin olisi ollut järkevämpää käyttää push model replikointia kuin etäproseduurikutsuja, jotta ne olisivat olleet luotettavia. Vaikka niitä testatessa ei suoraan ilmennytkään ongelmia, on todennäköistä, että jatkossa niitä joutuu suunnittelemaan uudelleen. Muutenkin olisi ollut järkevämpää välttää mahdollisimman paljon turhaa NetMulticast etäproseduurikutsujen käyttöä, mutta annettujen aikarajojen, sekä oman aiemman kokemuksen puutteen takia, NetMulticast kutsujen helppokäyttöisyyden takia, niitä tuli omasta mielestä käytettyä liikaa.

Mitään toiminnallisuutta huijaamisen estämiseksi ei tullut tehtyä, jonka takia pelaajat voivat lähettää asiakasohjelmalta palvelimelle itse muuttamiaan virheellisiä arvoja. Vaikka peli ei ole kilpailuhenkinen, mielestäni tämä olisi ollut silti toteuttamisen arvoinen, eikä sen toteuttamiseen olisi vaadittu muuta, kuin verrata palvelimella olevia arvoja asiakasohjelman lähettämiin arvoihin.

Myös verkko ominaisuuksien profilointiin olisi ollut hyvä käyttää aikaa. Vaikka peli onkin kooltaan pieni, eikä vaadi suuria resursseja koneelta tai verkolta, olisi ollut hyödyllistä saada jonkinlaista tietoa verkon käytöstä. Tämän avulla toteutuksen verkko ominaisuuksia olisi voinut optimoida valmiiksi, jotta sen käyttö voitaisiin mahdollistaa tulevissa suuremmissa projekteissa.

Lähteet

Andrii Chornyl. 2024. Informaatiota Bluerinttien ja C++ koodin eroista. Viitattu 1.11.2024. Saatavissa https://codefinity.com/blog/Blueprints-vs.-C-plus-plus-in-Unreal-Engine

Arstechnica. 2020. Unreal Engine lisensöintimaksuista. Viitattu 8.10.2024. Saatavissa https://arstechnica.com/gaming/2020/05/unreal-engine-is-now-royalty-free-until-a-game-makes-a-whopping-1-mill/

Liquid Web. Oman palvelimen ja kuuntelupalvelimen vertailua. Viitattu 22.10.2024. Saatavissa https://www.liquidweb.com/help-docs/dedicated-vs-listen-server/

Epic Online Services a. Yleistä tietoa Epic Online Servicesistä. Viitattu 22.10.2024. Saatavissa https://onlineservices.epicgames.com/en-US/services

Epic Online Services b. Epic Online Services dokumentaatio. Viitattu 22.10.2024. Saatavissa https://dev.epicgames.com/docs/epic-online-services

Epic Online Services c. Title Storage rajapinnan dokumentaatio. Viitattu 22.10.2024. Saatavissa https://dev.epicgames.com/docs/game-services/title-storage

Epic Online Services d. NAT P2P rajapinnan dokumentaatio. Viitattu 22.10.2024. Saatavissa https://dev.epicgames.com/docs/game-services/p-2-p

Epic Online Services e. Sessio rajapinnan dokumentaatio. Viitattu 22.10.2024. Saatavissa https://dev.epicgames.com/docs/game-services/lobbies-and-sessions/sessions/sessions-intro

Gamedeveloper. 2015. Unreal Engine lisensöintimaksuista. Viitattu 6.10.2024. Saatavissa https://www.gamedeveloper.com/business/unreal-engine-4-is-now-free-to-download-for-everyone

Kieran Newland. 2024. Push model replikointi. Viitattu 18.10.2024. Saatavissa https://www.kierannewland.co.uk/push-model-networking-unreal-engine/

Polygon. 2012. Unreal Engine historia. Viitattu 1.10.2024. Saatavissa

https://www.polygon.com/2012/10/1/3438196/better-with-age-a-history-of-epic-games

Redpoint games. Epic Online Services. Viitattu 22.10.2024. Saatavissa https://docs.redpoint.games/eos-online-subsystem/docs/core_getting_started/

Unreal Engine a. Lähdekoodi. Viitattu 8.10.2024. Saatavissa https://www.unrealengine.com/en-US/ue-on-github

Unreal Engine b. Unreal Enginen blueprint ohjelmoinnin dokumentaatio. Viitattu 8.10.2024. Saatavissa https://dev.epicgames.com/documentation/en-us/unreal-engine/overview-of-blueprints-visual-scripting-in-unreal-engine?application_version=5.4

Unreal Engine c. Tietoa Python ohjelmoinnista Unreal Enginessä. Viitattu 8.10.2024. Saatavissa https://dev.epicgames.com/documentation/en-us/unreal-engine/scripting-the-unreal-editor-using-python?application_version=5.4

Unreal Engine d. Unreal Engine blueprint-työkalun käytöstä. Viitattu 22.10.2024. Saatavissa https://dev.epicgames.com/documentation/en-us/unreal-engine/introduction-to-blueprints-visual-scripting-in-unreal-engine

Unreal Engine e. Yleistä informaatiota C++-ohjelmoinnista Unreal Enginessä. Viitattu 22.10.2024. Saatavissa https://dev.epicgames.com/documentation/en-us/unreal-engine/introduction-to-cplusplus-programming-in-ue4?application_version=4.27

Unreal Engine f. Lisää informaatiota Unreal Engine C++ ohjelmoinnista. Viitattu 22.10.2024. Saatavissa https://dev.epicgames.com/documentation/en-us/unreal-engine/programming-quick-start?application_version=4.27

Unreal Engine g. Informaatiota C++ makroista. Viitattu 22.10.2024. Saatavissa https://dev.epicgames.com/documentation/en-us/unreal-engine/ufunctions-in-unreal-engine?application_version=5.1

Unreal Engine h. Unreal Enginen verkkomoninpeli arkkitehtuurista. Viitattu 7.11.2024. Saatavissa https://dev.epicgames.com/documentation/en-us/unreal-engine/networking-overview-for-unreal-engine

Unreal Engine i. Unreal Enginen etäproseduurikutsujen dokumentaatio. Viitattu 22.10.2024. Saatavissa https://dev.epicgames.com/documentation/en-us/unreal-engine/rpcs?application_version=4.27

Unreal Engine j. Online Subsystem dokumentaatio. Viitattu 26.10.2024. Saatavissa https://dev.epicgames.com/documentation/en-us/unreal-engine/online-subsystem?application_version=4.27

Unreal Engine k. Online Subsystem Steam dokumentaatio. Viitattu 26.10.2024. Saatavissa https://dev.epicgames.com/documentation/en-us/unreal-engine/online-subsystem-steam?application_version=4.27

Unreal University. 2024. Blueprint ja C++ -ohjelmointikielien vertailua. Viitattu 1.11.2024. Saatavissa https://www.unreal-university.blog/blueprints-vs-c-unreal-engine/