LAB University of Applied Sciences

Uudelleenkäytettävä Replay-paketti

LAB-ammattikorkeakoulu
Tieto- ja viestintätekniikan insinööri (AMK)
2022
Verneri Haaja

Tiivistelmä

Tekijä(t)	Julkaisun laji	Valmistumisaika
Haaja, Verneri	Opinnäytetyö, AMK	2022
	Sivumäärä 30	

Työn nimi

Uudelleenkäytettävä Replay-paketti

Tutkinto ja koulutusala

Insinööri (AMK) Tieto- ja viestintätekniikka

Toimeksiantajan nimi, titteli ja organisaatio (jos opinnäytetyöllä on toimeksiantaja)

Kuura Games OY

Tiivistelmä

Työn tavoitteena oli kehittää replay-paketti käyttämällä Unreal Engine pelimoottorin replay-ominaisuuksia. Paketin oli tarkoitus tallentaa pelin tapahtumia viimeisen viiden sekunnin aikana. Paketin täytyi myös toistaa uusintoja välittömästi, kun tarve vaatii. Työn paketin täytyi olla yksinkertaisesti siirrettävissä toiseen projektiin.

Työn toimeksiantaja on Kuura Games Oy. Kuura Games on kotkassa toimiva pelialan yritys, joka julkaisee pelejä ja konsultoi omaa osaamistaan.

Projekti toteutettiin hyödyntämällä Unreal Engine pelimoottorin replay-järjestelmän Streamer-olioita. Streamer-oliot vaativat C++-ohjelmointi kieltä. Käytössä oleva Streamer-olio on MemoryStreamer, joka mahdollistaa välittömien uusintojen käytön. Paketin muu toiminnallisuus luotiin blueprint-systeemin avulla.

Lopputuloksena luotiin uudelleen käytettävä replay-paketti, joka toimii annettujen määreiden mukaan. Pelimoottorin tarjoamat replay-ominaisuudet ovat rajoittuneita ja ne ovat huonosti dokumentoituja. Pelimoottorin replay-ominaisuudet vaikuttavat olevan parempia koko pelien uusintoihin, jotka katsotaan pelin jälkeen.

Asiasanat

Replay, Pelit, Unreal Engine, Moninpeli

Abstract

Author(s)	Type of Publication	Published
Haaja, Verneri	Thesis, UAS	2022
	Number of Pages	
	30	

Title of Publication

Reusable replay package

Possible subtitle(s)

Degree and field of study

Engineer (UAS)

Name, title and organisation of the client (if the thesis work is commissioned by another party)

Kuura Games Oy

Abstract

The goal of the thesis was to create a replay package using Unreal Engine replay properties. The package needs to record the latest events of the game for five seconds. Then the package needs to play the recording as an instant replay when it is called. The package needs to be simply moved to another project.

Works client was Kuura Games Oy. Kuura Games is a game company based in Kotka. The company releases games and gives consultations regarding various things.

The project is made using Unreal Engines built in replay systems Streamers. Streamers require C++ programming to work. This project uses Memory Streamer as its Streamer of choice. Memory Streamer allows instant replays in the package. The rest of the functionality is created using the blueprint system.

In the end the replay package was created and the requirements for it was met. The replay properties given by the engine are limited and poorly documented. The replay properties seem to better be suited for replays of the whole game and to be viewed after the game.

Keywords

Replay, Game, Unreal Engine, Multiplayer

Sisällys

1 Johdanto

Viimeisien vuosien aikana moninpelien kysyntä on kasvanut räjähdysmaisesti. Syynä kasvulle on vuonna 2020 tapahtunut korona pandemia. Ihmisten täytyi sopeutua uudenlaiseen maailman tilanteeseen ja etsiä uusia tapoja viettää aikaa muiden ihmisten kanssa. Nuorten keskuudessa pelaamisen määrä kasvoi 93 % korona karanteenien aikana. Yllättävästi myös yli 60-vuotiaiden keskuudessa pelaaminen lisääntyi 200 % ja suuri osa uusista vanhemmista pelaajista pelaa nyt enemmän kuin ennen koronaa. Korona karanteenien ja lähikontaktien välttäminen on kasvattanut ihmisten kiinnostusta suuriin moninpeleihin. Pelit kuten World of Warcraft ja Grand Theft Auto V Online ovat saavuttaneet usean sadan prosentin kasvun korona pandemian alun jälkeen. Pelaajakasvun takia myös kilpailullisuus on kasvanut peleissä. Pelaajien on mahdollista kehittää omia taitojaan tutkimalla parempia pelaajia, kuten hyödyntämällä pelin sisäisiä replay-toimintoja. (G2A.)

Korona pandemian aikana massiiviset moninpelit ovat lisänneet muitakin toimintoja peleihin, kuten Fortnite. Fortnite pelin alustaa on hyödynnetty järjestämällä konsertteja, elokuva trailereita ja jopa paneeli keskusteluja. Kaikki nämä ominaisuudet mahdollistavat pelimoottori Unreal Engine 4, jonka on kehittänyt Epic Games. Unreal Enginen grafiikka- ja fysiikkaominaisuudet tekevät siitä kilpailukykyisen kehitysympäristön AAA-pelitaloille ja harrastelijoille.

Opinnäytetyön tavoitteena on kehittää replay-paketti. Replay-paketin on tarkoitus tallentaa pelin tapahtumia viiden sekunnin ajan. Paketin on tarkoitus myös toistaa tallennettuja tapahtumia välittömänä uusintana. Paketti on myös tarkoitus yksikertaisesti siirtää muihin projekteihin. Teoriaosiossa perehdytään Unreal Engine 4 historiaan ja ominaisuuksiin, Unreal Enginen verkkotoimintoihin, Advanced Sessions lisäosaan ja replay-järjestelmään. Tämän jälkeen toteutusosuudessa esitellään, ja kuinka replay-paketti toteutetaan.

Opinnäytetyön toimeksiantajana toimii Kuura Playhouse Oy, joka sijaitsee Kotkassa. Kuuran pääasiallinen toimiala on IT-konsultointi ja IT-palvelut. Kuura kehittää pelejä omalla nimellä ja myös konsultoi osaamistaan muille alan yrityksille. Kuuralla on tällä hetkellä 3 työntekijää.

2 Unreal Engine

2.1 Yleistä

Unreal Enginen ensimmäinen versio julkaistiin vuonna 1998, koska Epic Gamesin kehittäjät tarvitsivat pelimoottoria Unreal gamesin kehitykseen. Ensimmäinen versio pelimoottorista kasvoi suureen suosioon pelitiedostojen kustomoinnin takia. Kustomoinnin avulla pelaajat pystyivät muuttamaan pelin sisältöä omilla asseteilla, hyödyntäen UnrealScriptiä. Unreal Tournament pelin kehityksen kautta pelimoottoriin pystyttiin lisäämään moninpeli toimintoja ja PlayStation 2 tuki, Windowsin ja Macin rinnalle. (Lee 2016, 2.)

Pelimoottorin seuraava versio Unreal Engine 2 julkaistiin 2002. Unreal Engine 2 oli suuri harppaus pelimoottorin ensimmäiseen versioon. Pelinkehittäjien oli nyt mahdollista rakentaa uudenlaisia tasoja pelimoottorin sisäisessä taso editorissa. Verrattuna ensimmäiseen versioon toinen versio suoriutui paremmin renderöimisessä, pelifysiikassa ja törmäyksien tunnistuksessa. Myös Xbox ja GameCube lisättiin tuettujen laitteiden listalle. (Lee 2016, 2.)

Vuonna 2006 Epic Games julkaisi suosituimman version pelimoottoristaan, Unreal Engine 3. Unreal Engine 3 avulla kehittäjät pystyivät luomaan todella realistisia hahmoja ja olioita, hyödyntämällä DirectX 9/10 teknologiaa. DirectX lisäksi Unreal Engine hyödynsi uutta visuaalista ohjelmointi systeemiä nimeltä Kismet. Kismet mahdollisti yksinkertaisemman tavan pelitason kehitykseen ja pelimekaniikoista pystyttiin tekemään dynaamisempia. Myös artistit kiinnostuivat Unreal Enginestä, koska heillä oli tarve nopeaan prototyyppien luontiin itsenäisesti ilman koodin kirjoittamista. (Joanna Lee 2016, 3.)

Unreal Engine 4 on viimeisin versio pelimoottorista. Neljännessä versiossa tapahtui suuria muutoksia, jotka helpottivat kaikkia kehittäjiä. UnrealScript poistui täysin ja se korvattiin C++ ohjelmoinnilla, jotta kehittäjien ei enää tarvinnut opetella uutta kieltä. C++ lisäys pelimoottoriin antoi kehittäjille laajemman kontrollin kehitysalustasta, koska he pystyivät tekemään muutoksia lähdekoodiin. Unreal Engine on Open source pohainen pelimoottori ja lähdekoodi on saatavilla GitHubissa. UnrealScriptin poistumisen lisäksi, Kismet muutettiin Blueprint systeemiin. Unreal Engine 4 avulla on mahdollista kehittää pelejä kaikille suosituimmille konsoleille, kuten PlayStation 4, Xbox one ja Nintendo Switch. (Sufyan bin Uzayr 2022, 4.)

2.2 Blueprint

Blueprint on Unreal Enginen sisäänrakennettu visuaalinen ohjelmointi järjestelmä, jonka avulla luodaan pelielementtejä Unreal Editorin sisällä. Blueprint systeemi on samanlainen kuin kirjoitetut ohjelmointi kielet ja jakaa monia ominaisuuksia niiden kanssa. Data pohjaisten muuttujien ja taulukoiden lisäksi blueprint systeemi on myös oliopohjainen. Jokaisella blueprintillä on oma tapahtumakartta. Tapahtumakartan sisällä luodaan pelin toiminnallisuus kutsumalla funktioita ja tapahtumia. (Epic Games. b.)

Blueprintin tarkoituksena oli luoda systeemi, joka mahdollistaa taitotasosta riippumatta kehityksen Unreal Enginellä. Pelien kehitys nopeutuu, koska blueprinttiä on helppo muokata ja tarkastella editorissa ilman, että koko peli täytyy uudestaan koota. Yksinkertaisuuden lisäksi blueprintin avulla datan säilöntä on helpompaa ja turvallisempaa kuin C++ luokissa. Blueprint on parhaimmillaan, kun luokissa käytetään logiikkaa ja dataa. (Epic Games. b.)

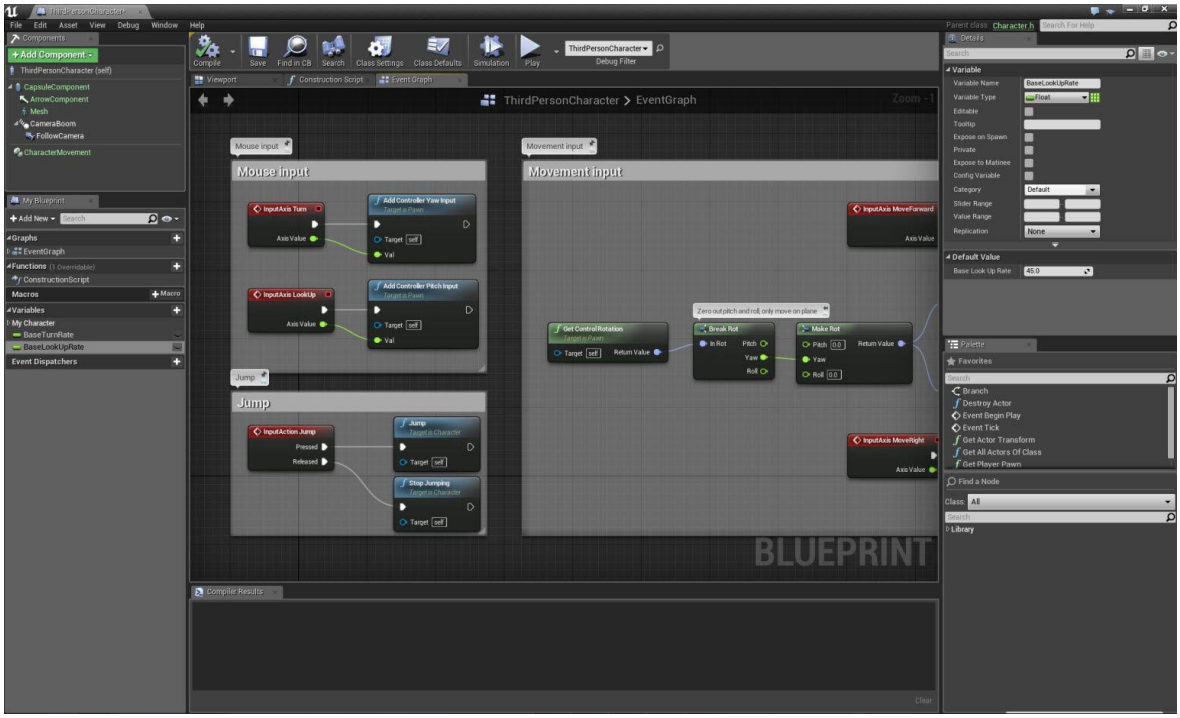
Blueprintin suurin heikkous on suorituskyky. Yleisesti yksittäisen blueprint-solmun ajaminen on hitaampaa kuin saman toiminnon toteuttaminen C++-koodilla. Suurin ero huomataan, jos ohjelmassa on paljon solmuja tai sisäkkäisiä makroja, jotka sisältävät paljon solmuja. Yleisin suorituskyvyn heikentäjä on Tick-funktio blueprintin sisällä. Tick-funktion tarkoitus on suorittaa koodia pelin sisällä jokaisen ruudunpäivityksen yhteydessä. Blueprintin sisäisen Tick-funktion suoritus on aina hitaampaa, kuin natiivin Tick-funktion. Tick-funktioita täytyy useasti välttää, jos luokasta on paljon instansseja. (Epic Games. c.)

Pelin sisällä olevat asiat voidaan jakaa logiikkaan ja dataan. Pelin logiikkaa kuvataan itsenäisinä ohjeina ja rakenteena pelille. Pelin sisäisellä datalla kuvataan, mitä peli tekee ja useasti data on logiikan käyttämää. Logiikka tehdään blueprintin tapahtumakartassa tai funktioissa, jotka ovat kutsuttavissa tapahtumakartassa. Blueprint systeemissä data voidaan sijoittaa luokkiin, jotka seuraavat samoja sääntöjä kuin C++ luokat. Luokat mahdollistavat datan perinnän. (Epic Games. c.)

2.2.1 Käyttöliittymä

Blueprint editori on solmupohjainen editori, jonka tarkoitus on luoda ja muokata visuaalista ohjelmointi solmuverkkoa. Editori on konteksti pohjainen systeemi, joka sisältää paljon työkaluja, jotka auttavat luomaan tarvittavia olioita tai asioita. Sen sisällä on sisäänrakennettu useita testaus- ja analysointi työkaluja. Ulkomuoto editorissa muuttuu sen mukaan, millaista blueprint verkkoa käsitellään. (Epic Games. q.)

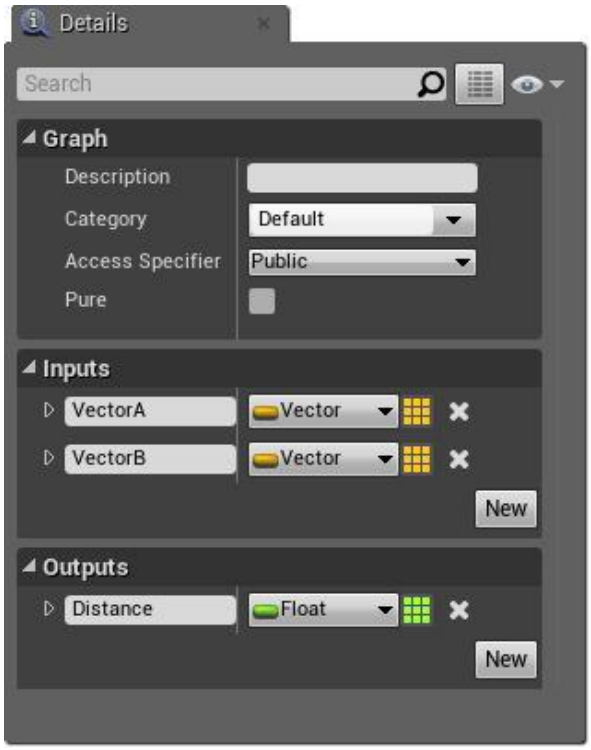
Kuva 1 esittää, miltä blueprint editori näyttää pelimoottorin sisällä. Kuvassa vasemmassa laidassa on lista kaikista komponenteista blueprintin sisällä ja sen alapuolella on muuttuja- ja funktiolista. Keskellä kuvaa on tapahtumakartta näkymä, jonka sisällä blueprint solmut luodaan. Oikealla kuvassa on yksityiskohtia näkymä, joka näyttää funktioiden, muuttujien tai komponenttien lisätietoja.



Kuva 1. Blueprint editori pelimoottorin sisällä

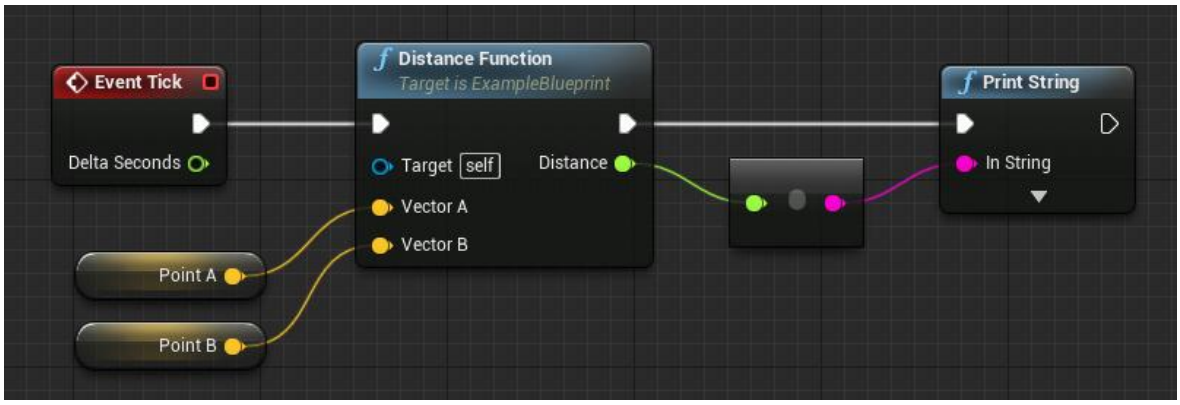
2.2.2 Funktiot

Funktiot ovat osa blueprinttiä, jonka sisällä niitä voidaan suorittaa tai kutsua toisesta funktiosta. Funktioissa on yksi sisääntulo piste, jonka kautta se liitetään solmuverkostoon. Funktiolla myös voi olla omia muuttujia ja useita eri syöte- tai ulostulomuuttujia. Muiden ohjelmointikielien mukaan myös blueprint funktioissa voi määrittää sen näkyvyyden. Kuvassa 2 esitetään miltä syöte- ja ulostulomuuttujat näyttävät editorissa. Samassa hallintapaneelissa voidaan muuttaa funktion näkyvyyttä painamalla Access Specifier laatikkoa. (Epic Games a.)



Kuva 2. Funktio editorissa

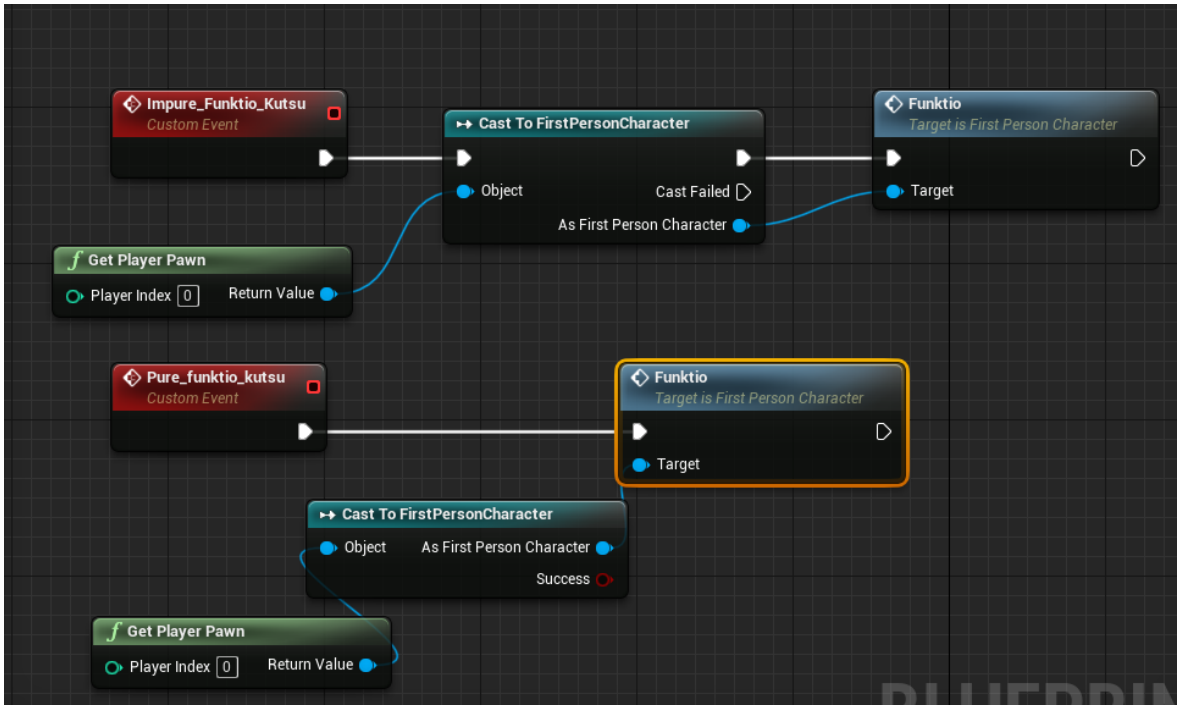
Funktioiden kutsuminen tapahtuu tapahtumakartassa erilaisten tapahtumien avulla. Funktioita on mahdollista kutsua ulkoisesta blueprintistä, jos se sisältää viittauksen funktion omistavaan blueprinttiin. Kuva 3 on esimerkki siitä, kuinka funktiota kutsutaan. Ensin liitetään funktio Tick-tapahtumaan, joka kutsuu funktiota Distance Function. Seuraavaksi funktion syötemuuttujat täytetään vektori muuttujilla (Point A ja B). Sitten Tick-tapahtuma kutsuu funktiota kerran syklissä, jonka jälkeen se välittää ulostulomuuttujan seuraavalle solmulle.



Kuva 3. Funktion kutsu blueprintin sisällä

Funktiot voivat olla Pure tai Impure tyyppisiä, mutta ei samaan aikaan molempia. Tärkein ero näiden kahden välillä on niiden vaikutus luokan tilaan. Pure funktiot eivät muokkaa tilaa ollenkaan ja Impure funktiot ovat vapaita muokkaamaan tilaa kuten haluavat. Kuvas-

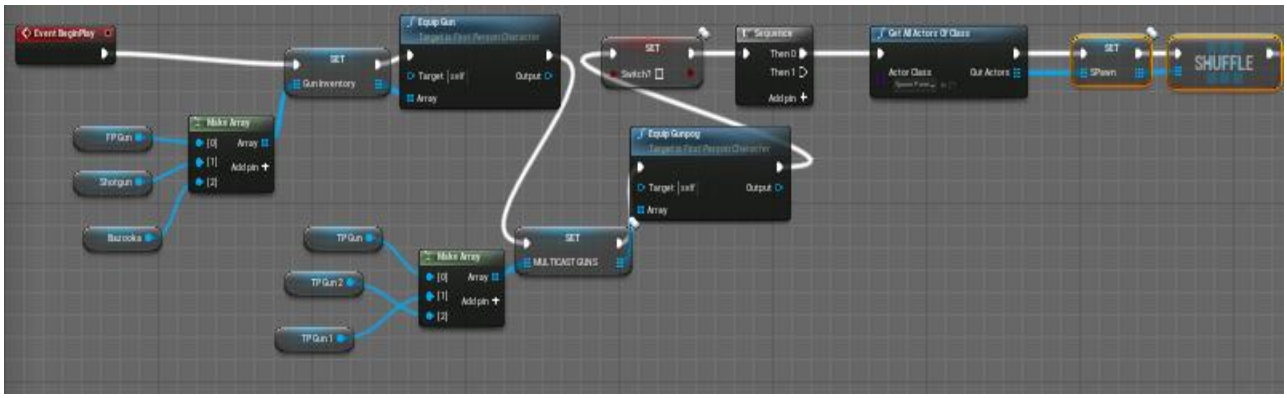
sa 4 esitetään kuinka, Impure-funktio on yhdistetty solmuverkkoon suoraan ja Pure-funktio on liitetty funktioon ilman solmuverkosto liitäntää. Impure funktio vaatii suoran kutsun yhdistämällä sen solmuverkostoon. Kääntäjä kutsuu automaattisesti pure funktioita, kun niihin liitetty data on tarpeellista. (Epic Games. a.)



Kuva 4. Pure – ja Impure-funktion kutsu

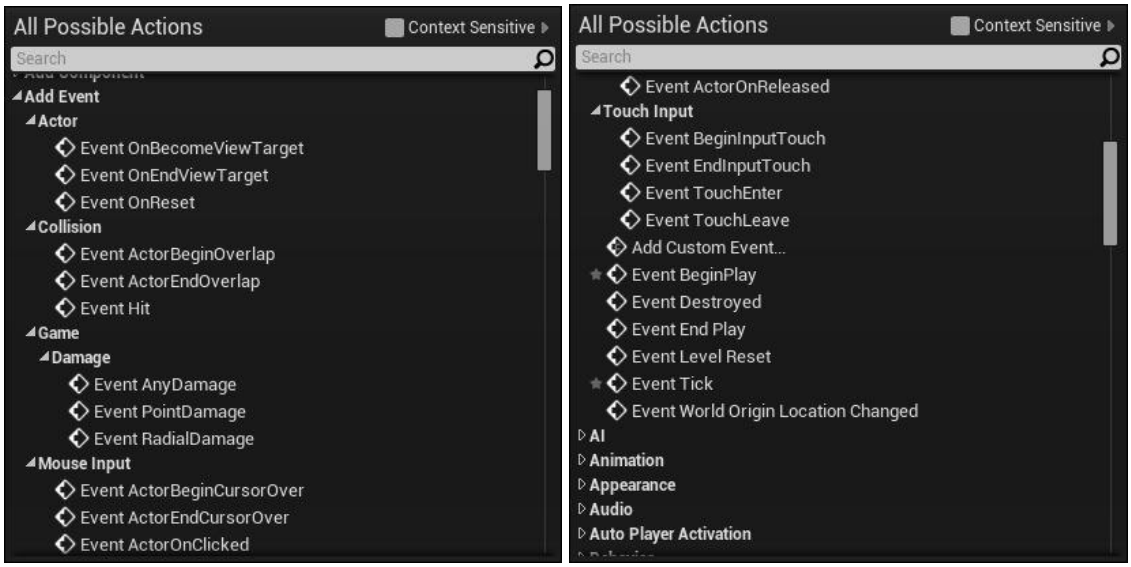
2.2.3 Tapahtumat

Tapahtumat ovat solmuja, jotka aloittavat yksittäisten verkostojen suorituksen, kun niitä kutsutaan pelin sisäisen koodin kautta. Tapahtumaan liitetty koodi suoritetaan, kun tapahtuma aktivoituu, kuten esimerkiksi kun pelaaja vahingoittuu tai peli alkaa. tapahtumakartan sisällä voi kutsua tapahtumaa vain kerran. Tapahtuma voi suorittaa vain yhden olion, minkä takia useampi toiminto ketjutetaan yhteen. (Epic Games. d.) Kuva 5 esittää tapahtumaa BeginPlay, joka suoritetaan pelin alussa. Tapahtumaan on liitetty useampi toiminto, jotka suoritetaan seuraamalla valkoista viivaa lineaarisesti.



Kuva 5. BeginPlay-tapahtuma blueprintin sisällä

Blueprint systeemissä on useita sisäänrakennettuja tapahtumia, jotka suoritetaan, kun ennalta määritelty toiminta tapahtuu pelin sisällä. Kuvassa 6 on esitetty sisäänrakennettuja tapahtumia. Tapahtumia voi myös luoda itse ja luotuja tapahtumia voidaan kutsua osana blueprintin sykliä. Itse luotuja tapahtumia ei kutsuta automaattisesti pelin sisäisen koodin kautta, joten niille täytyy määrittää kutsumis- tapa. Yleisin tapa kutsua itse luotuja tapahtumia on liittää ne osaksi sisäänrakennettujen tapahtumien solmuverkostoa. (Epic Games. e.)



Kuva 6 Sisäänrakennetut tapahtumat.

3 Replikaatio ja verkkorakenne

3.1 Client -Server malli

Unreal Engine käyttää client-server mallia moninpeli ominaisuuksien toteutukseen. Client-server mallissa yksi peli-instanssi toimii serverinä, ja muut peli-instanssit liittyvät serverille client-pelaajina. Malissa server on auktoritatiivinen ja hallitsee pelin tilaa. Client pelaaja ohjaa hahmoa, jonka hän omistaa serverillä. Serveri replikoi tietoa pelin tilasta jokaiselle clientille, määrittäen mitkä asiat pelissä ovat olemassa ja miten ne käyttäytyvät. Jokainen client käyttää tätä tietoa simuloidakseen tarkasti, mitä serverillä tapahtuu. (Epic Games. f.)

Verkkopelissä pelaajien väliset kanssakäymiset sijoittuvat usean eri maailman sisään. Server-pelaajalla on maailma, jossa peliä oikeasti pelataan. Jokaisella client-pelaajalla on oma maailma, jonka sisään kopioidaan muut hahmot ja heidän toimintonsa, kuten ampuminen ja liikkeet. (Epic Games. f.) Kuvassa 7 esitetään kuinka client 1-pelaaja antaa paikallisesti käskyn server-hahmolleen ampua. Server välittää käskyn kaikille client-pelaajille ja luo pelaajan ampuman luodin. Client 2-pelaajan hahmo serverillä vahingoittuu ammutusta luodista. Tieto välitetään client 2-paikalliselle hahmolle ja serveri määrittää, mitä toimintoja tehdään.

Kuva 7. Client yksi pelaaja ampuu client kaksi pelaajaa

Client-server mallissa on kaksi tapaa luoda serveri: Dedicated server ja Listen server. Molemmilla tavoilla on oma paikkansa moninpeleissä. Dedicated server hallintamalli toimii

erillisellä tietokoneella, joka hyväksyy yhteyksiä etä-clienteiltä. Dedicated hostaus metodilla ei ole ollenkaan paikallisia pelaajia. Dedicated server ei renderöi ylimääräisiä grafiikoita tai logiikkaa, jos se ei ole tarpeellista tietoa serverille. Tästä syystä serveri prosessoi peliominaisuudet tehokkaasti. Suuremmissa paljon prosessointi tehoa vaativissa peleissä käytetään dedicated serveriä. (Epic Games. f.)

Listen server mallissa yksi pelaajista isännöi moninpeli sessiota. Serverillä on siis etä-clienttejä ja paikallinen pelaaja. Systeemin paras puoli on sen helppous ja vaivattomuus, koska se ei vaadi ulkoisia servereitä. Listen serverillä pelin hostaajalla on suurin etu, koska hän pelaa suoraan serverillä. Paras tapa hyödyntää Listen serveriä on pienissä vähän verkkoliikennettä vaativissa peleissä. (Epic Games. f.)

3.2 Server matkustus

Unreal Engine 4 pelimoottorissa on mahdollista matkustaa kahdella eri tavalla pelitasojen välillä, tavat ovat seamless -ja non-seamless-matkustus. Tapojen erona on se, että seamless-matkustus on ei blokkaava operaatio, kun taas non-seamless on blokkaava kutsu. Clientin kutsuessa non-seamless-matkustusta, client katkaisee yhteyden serverille ja hetken päästä yhdistää takaisin, jolloin serveri on ehtinyt lataamaan uuden tason. Unreal Engine 4 suosittelee, että moninpeleissä käytetään seamless-matkustusta, aina kun mahdollista. Seamless-matkustus on parempi vaihtoehto ja auttaa välttämään ongelmia uudelleen yhdistyksen aikana tapahtuvassa prosessissa. (Epic Games. g.)

Matkustusta on mahdollista käyttää kolmella eri funktiolla: Browse, ServerTravel ja ClientTravel. Funktioita käytetään konsoli komento solmua käyttäen. Browse-funktio käyttää aina non-seamless-matkustusta, koska se pakottaa pelaajat katkaisemaan yhteyden, kun pelin tasoa vaihdetaan. ClientTravel-funktion toiminta riippuu siitä, kutsuuko client vai server sitä. Client kutsu siirtää kutsujan uudelle serverille, ja serveri kutsu siirtää tietyn clientin uuteen karttaan, mutta pitää sen samalla serverillä. ServerTravel-funktiota voidaan kutsua vain serveriltä ja sen tarkoitus on siirtää kaikki yhdistäneet clientit serverin mukana uuteen karttaan. ServerTravel-komento toimii parhaiten seamless-matkustuksen avulla. (Epic Games. g.)

Seamless-matkustus vaatii siirtymäkartan, jotta metodi voidaan ottaa käyttöön. Siirtymäkartta vaaditaan, koska pelissä täytyy olla koko ajan kartta ladattuna. Pelien kartat voivat olla suuria ja viedä paljon muistia, joten on huono idea käynnistää uutta karttaa, jos tason lataus on vielä kesken. Siirtymäkartat ovat pienikokoisia. (Epic Games. g.) Seuraavassa

kuvassa (Kuva 8) on esitetty, kuinka seamless-matkustus otetaan käyttöön painamalla valintaruutu todeksi.

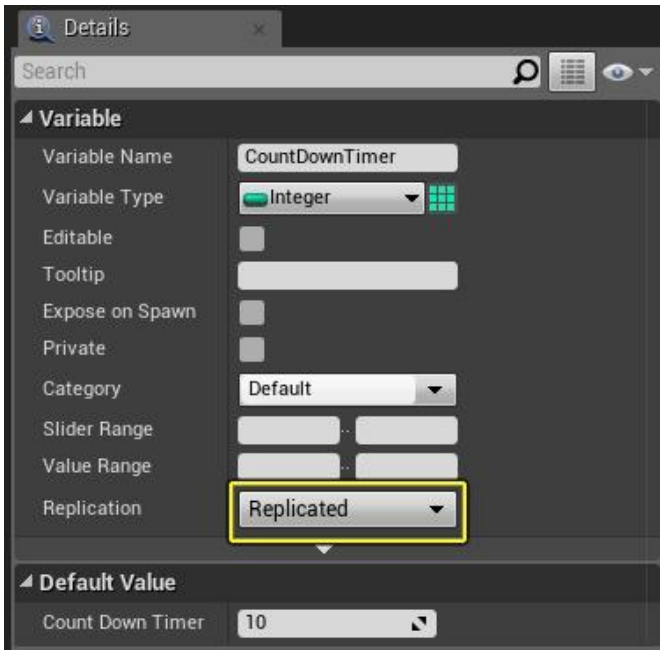
Kuva 8. Seamless-matkustuksen käyttöönotto

3.3 Replikointi

Moninpelit useasti vaativat suuren datan määrän synkronoinnin monelle clientille, jotka sijaitsevat ympäri maailmaa. Tällöin tärkeäksi asiaksi muodostuu miten ja millaista dataa lähetetään käyttäjille, koska se vaikuttaa suuresti pelikokemukseen. Unreal Enginessä datan replikointi clientin ja serverin välillä kutsutaan replikaatioksi. (Epic Games. h.)

Replikointi tapahtuu pääosin Actor-luokassa. Serveri päivittää Actor-luokkia kahdella tavalla, ominaisuus päivityksillä ja RPC-kutsuilla. Erona näillä tavoilla on se, että ominaisuudet on replikoitu automaattisesti, kun ne muuttuvat, ja RPC kutsut päivittyvät, kun niitä kutsutaan. Muuttujat on mahdollista replikoida omistavalta pelaajalta etä-clientille, kun sen

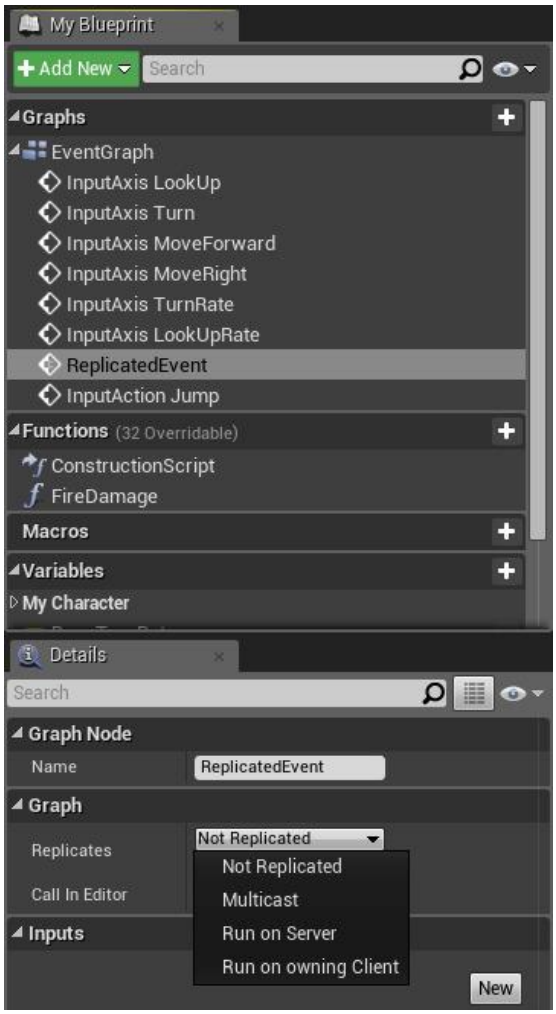
arvot muuttuvat. (Epic Games. h.) Kuva 9 esittää muuttujan replikoinnin aktivointia blueprint editorin sisällä. Muuttujan tiedoissa painetaan replikaatio laatikkoa ja valitaan replicated.



Kuva 9. Muuttujan replikoinnin aktivointi

Replikoidut funktiot tarkoittavat RPC-funktioita eli Etäproseduurikutsuja. RPC-funktioita on mahdollista kutsua kaikilta yhdistetyiltä laitteilta, mutta funktioiden toiminta koskee vain yhtä laitetta. RPC-funktiot ovat hyödyllisiä, koska ne mahdollistavat clientin ja serverin välisen keskustelun verkon yli. (Epic Games. i.)

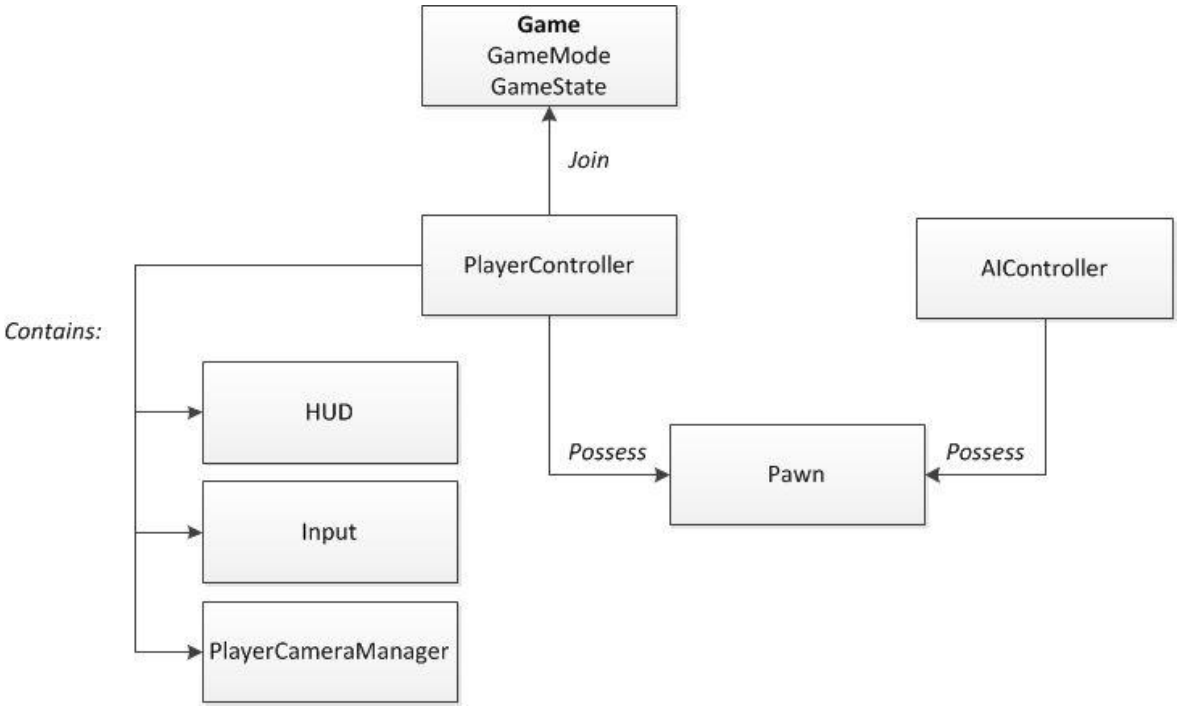
Replikoituja funktioita on kolmea eri tyyppiä: Multicast, Run on Server ja Run on owning Client. Multicast-fuktioita voi kutsua serveriltä tai clientilitä. Serveriltä kutsuttaessa Multicast-funktiot suoritetaan serverin paikallisessa peli-instanssissa ja jokaisella yhdistyneellä clientillä. Multicast-funktion toiminnot suoritetaan paikallisesti, jos sen kutsuu client. Run-on-server funktiot kutsutaan client-laitteelta, mutta ajetaan serverillä. Run-on-owning-client funktiot kutsutaan serveriltä ja ne ajetaan omistavalla clientillä. (Epic Games. i.) Kuva 10 kuvaa funktion replikointia blueprintissä.



Kuva 10. Funktion replikointi valikko

3.4 Luokat

Pelihahmojen toiminnat perustuvat Unreal Enginen sisäiseen inputsysteemiin, joka muuntaa käyttäjän syötteet toiminnoiksi pelin sisällä. Inputsysteemin hallinta ja konfigurointi sisältyy yleisen Gameplay-viitekehyksen sisään. Gameplay-viitekehyksen tarkoituksena on seurata pelin tilaa ja pelin sisäisiä sääntöjä. Pelin sääntöjä tai tilaa hallitaan peliluokilla. Yleisimmät pelin sisäiset luokat ovat Pelimuoto, pelitila ja pelaajantila. Kuva 11. Havainnollistaa pelin luokkia ja kuinka ne ovat yhteydessä toisiinsa. Peli koostuu pelitilasta ja pelimuodosta ja kun pelaaja liittyy peliin hänet, liitetään PlayerController luokkaan. (Lee 2016, 7.)



Kuva 11. Gameplay-viitekehyksen luokkia (Epic Games p)

Pelin sisällä tapahtuvan informaation hallintaan on kaksi pääluokkaa: Pelimuoto ja Pelitila. Pelimuoto luokka koostuu erilaisista säännöistä. Näitä sääntöjä on monenlaisia, kuten sallittujen pelaajien määrä tai onko peli pysäytetty vai ei. Kun pelissä tapahtuu sääntöjen pohjalta tapahtumia, kutsutaan pelitilaa, joka jakaa ja synkronoi informaation kaikille pelaajille. Pelitilan informaatiota on esimerkiksi tieto siitä, kuinka kauan peliä on pelattu tai onko peli alkanut. (Epic Games. j.)

3.4.1 Pelimuoto

Pelin sisällä on rajaton määrä sääntöjä, jotka vaikuttavat pelaajien toimintoihin ja tapahtumiin. Pelimuodon tehtävänä on määritellä ja implementoida sääntöjä pelin sisällä. Tällä hetkellä pelimuotoja on kaksi eri luokkaa AGameModeBase ja AGameMode. Kaikki pelimuodot ovat alaluokkia AGameModeBaselle, joka sisältää paljon perustoiminnallisuutta. Pelimuotoja voi olla rajaton määrä projektin sisällä, mutta vain yksi niistä voi olla käytössä pelin aikana. (Epic Games. j.)

Pelimuoto on vain olemassa serverillä, joten sitä ei replikoida etä-clientille. Paikallisella pelaajalla on mahdollisuus nähdä alkuperäinen pelimuoto, mutta pelaaja ei pääse tarkastelemaan muuttujia tai pelin aikaisia muutoksia. Tarvittava informaatio pelimuoto muutoksista säilytetään AGameStateBase actorissa, joka luodaan pelimuodon kanssa ja replikoidaan etä-clienteille. (Epic Games. j.)

Pelimuotoja on mahdollista implementoida C++-ohjelmoinnilla tai blueprint systeemillä. Blueprint pelimuodot mahdollistavat helpon ja nopean tavan vaikuttaa useampaan tasoon yhdellä muutoksella. Kehittäjän ei tarvitse muokata yksittäisen tason koodia, jos muutos halutaan useampaan tasoon. (Epic Games. j.)

3.4.2 Pelitila

Pelitilan tehtävänä on antaa clienteille mahdollisuus tarkkailla pelin tilaa. Pelitila hallitsee pelimuodon informaatiota, joka koskee kaikkia yhdistettyjä clienttejä, eikä vain tiettyä clienttiä. Parhain tapa hyödyntää pelitilaa on siis säilyttää tietoa koko pelimaailmasta, kuten yhdistetyt pelaajat tai joukkueiden pisteet. Pelaaja kohtaiset tiedot, kuten pelaajan omat pisteet, säilytettään pelaajan omassa tilassa. (Epic Games. j.)

Pelitila sijaitsee vain serverillä ja se replikoidaan kaikille clienteille, jotta kaikki pelaajat pysyvät ajan tasalla pelin tapahtumista. Perus pelitilan implementointi toimii AGameStateBase-luokan avulla. AGameStateBasea käytetään pohjana, jonka avulla rakennetaan blueprint, joka pitää pelaajat ajan tasalla. Tarvittavat ylimääräiset muutokset pelitilaan tulee toteuttaa vastaavan pelimuodon kautta. (Epic Games. j.)

3.5 Advanced sessions

Advanced sessions on Unreal Enginen ulkoinen lisäosa, joka mahdollistaa yksinkertaisen tavan implementoida verkkotoimintoja blueprintin avulla. Lisäosa ei ole Epic Gamesin kehittämä virallinen ominaisuus vaan yhteisön jäsenen. Advanced sessions moduuli koostuu kahdesta osasta: AdvancedSessions ja AdvancedSteamSessions. Moduulien erottamisen avulla kehittäjä kykenee lisäämään Steam kohtaisia ominaisuuksia ilman, että Steam alijärjestelmä latautuu projekteissa, jotka eivät sitä tarvitse. Lisäosan tarkoituksena on yksinkertaistaa ja lisätä toiminnallisuutta jo olemassa oleville online alijärjestelmä toiminnoille. (Epic Games. k.)

Online alijärjestelmä on rajapinta, jonka avulla on mahdollista hallita verkkopalveluiden toimintoja. Rajapinta on suunniteltu käsittelemään asynkronista kommunikaatiota verkko palveluiden kanssa, kuten Steamin. Paikalliset laitteet eivät tiedä taustapalveluiden verkko nopeuksista mitään, joten kommunikaatio aikaa palveluiden kanssa ei voi ennustaa. Tästä syystä online alijärjestelmä rajapinta käyttää Delegates-tapahtumia kaikkiin etä- toimintoihin. Delegates-tapahtumaa kutsutaan, kun tuettua asynkronista toimintoa käytetään. De-

legates-tapahtumat ovat tapahtumia, joita voidaan kutsua tai määritellä normaalisti. Erona muihin tapahtumiin on se, että kun delegates-tapahtumia suoritetaan niin kaikki, jotka kuuntelevat sitä, saavat tiedon tapahtuvista toiminnoista. (Epic Games. l.)

Valve on auttanut Unreal Enginen kehittäjiä kehittämään Online alijärjestelmä rajapinnan Steam -palvelulle. Steam -rajapinnan tarkoituksena on mahdollistaa Unreal Engine 4 sovellusten vaivaton kehitys Steam -palvelulle. Moduuli laajentaa Online alijärjestelmä rajapintaa käyttämällä Steamworks ohjelmankehitys pakettia (SDK). SDK avulla voidaan hyödyntää Steam-rajapinnan toimintoja, kuten cloud-toimintoja. (Epic Games. m.)

Steam tukee peer-to-peer-hostausta, dedicated - ja listen-serverien kautta. Rajapintaa hyödyntäen pelin aloitus tapahtuu aulassa, jonka kautta pelaajille voidaan jakaa informaatiota serveristä tai pelistä, kuten pelimuoto tai kartta. Aulat ovat peer-to-peer instansseja, jotka toimivat Steamin backend-palveluissa. (Epic Games. m.)

3.6 DemoNetDriver ja Streamer

Unreal Enginessä on sisäänrakennettu replay-systeemi, jonka avulla voidaan tallentaa pelin tapahtumia, ja katsoa niitä myöhemmin uudestaan. Replay-systeemiä on mahdollista käyttää projekteissa eikä se riipu verkkorakenteesta. Systeemi käyttää DemoNetDriver-ominaisuutta, joka lukee dataa replikointisysteemiltä. Vaikka peleissä ei olisi moninpeli toiminnallisuutta, stand alone-pelien replikointi data toimii täysin replay-systeemissä ilman ylimääräisiä muutoksia. (Epic Games. n.)

Replay-systeemin tarkoitus on pelin sisällä aloittaa -ja lopettaa tallennus, ja toistaa jo tallennettu video. Playback-tilan sisällä, replay-systeemi tukee komentoja tauko, toiston nopeuden muutos ja tiettyyn pisteeseen siirtyminen. Komentojen lisäksi tallenteisiin on mahdollista lisätä metatageja, ja niiden avulla löytää olemassa olevia tallenteita. Replay-syteemi toimii C++-koodin avulla, hyödyntäen UGameInstance -ja Uworld-luokkia. C++-koodin lisäksi konsoli komentojen avulla on mahdollista käyttää replay-systeemiä. (Epic Games. n.)

DemoNetDriver on verkkoajuri, joka siirtää replikoitua dataa Streamer-olioille, joka tallentaa informaation myöhempää toistoa varten. Streamer-olioiden tarkoitus on siis tallentaa ja luoda tallenteita. Pelimoottorin sisällä on useita eri Streamer-olioita, joita voidaan liittää DemoNetDriveriin, riippuen kuinka replay-dataa on tarkoitus tarkastella. Oletuksena systeemi käyttää Local File Streameriä, ja se tallentaa pelin tapahtumia suoraan levylle, jonka takia se sopii hyvin singleplayer-peleihin. Moninpeleissä käytetään Memory- tai HTTP-

Streameria. Memory-Streamer-olio toimii Client-laitteella ja sopii parhaiten välittömiin uusintoihin ja räiskintä pelien tyyliseen ”Killcam” systeemiin. HTTP-Streamer-olion tarkoituksena on lähettää dataa toiselle laitteelle internetin välityksellä, kuten Dedicated-serverille. Tämän takia HTTP-Streamer sopii parhaiten peleihin, joissa on paljon pelaajia. (Epic Games. o.)

Memory Streamer on uniikki verrattuna muihin Streamereihin, koska sen tarkoitus on suorittaa replay-ominaisuuksia pelin sisällä yhden session aikana. Meneillään oleva peli jatkuu huomaamattomasti, kun yksi pelaajista siirtyy katsomaan uusintaa. Pelimootori kerää kaikki käytössä olevat tasot kolmeen ryhmään: Static tasot, Dynamic Source tasot ja Dynamic Duplicated tasot. Ryhmät määrittelevät kuinka pelin tasot vuorovaikuttavat replay-systeemin kanssa. Static taso tarkoittaa tasoa, joka ei ole pelin päätaso. Näihin tasoihin ei vaikuteta pelinaikana ja ne näytetään uusinnan aikana. Dynamic Source taso on pysyvä. Siihen vaikuttaa pelin aikainen toiminta, ja se on uusinnan aikana piilotettu. Dynamic Duplicate taso on kopio dynamic source tasosta ja se on piilotettu pelin aikana, mutta uusinnat tapahtuvat tällä tasolla. (Epic Games. o.)

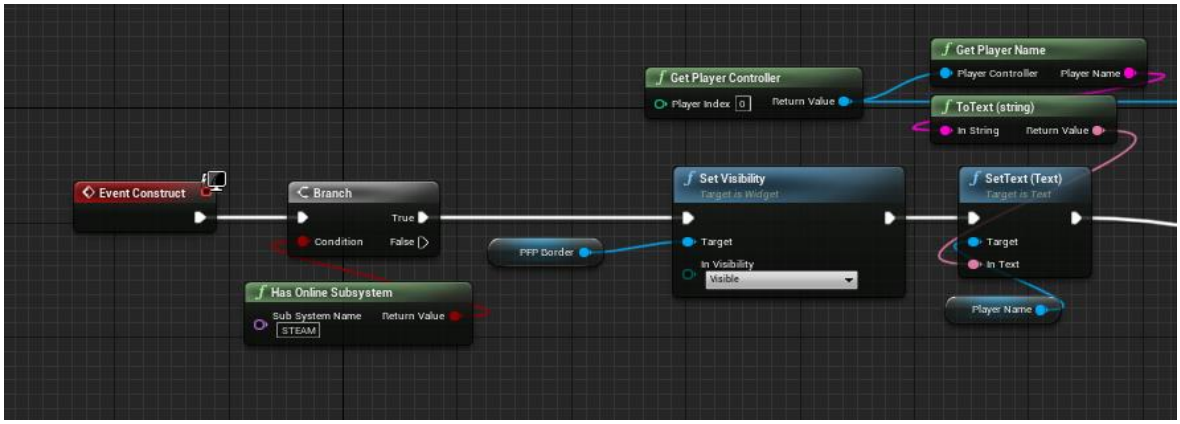
Pelin sisäinen replay-systeemi rakennetaan luomalla Dynamic Source tasolle ja Dynamic Duplicate tasolle omat DemoNetDriverit. Tämä mahdollistaa peli tapahtumien tallennuksen Dynamic Source tasolla, jonka jälkeen tallennettu tapahtuma voidaan toistaa Dynamic Duplicate tasolla. Dynamic Source taso piilotetaan uusinnan aikana pelaajalta, mutta peli silti jatkuu normaalisti ja saa normaalisti päivityksiä serveriltä. Kun tallennus päättyy ja pelaaja siirtyy takaisin Dynamic Source tasolle, Dynamic Duplicate taso tuhotaan. (Epic Games. o.)

4 Case

4.1 Networking

Pelin moninpeli toiminnot on toteutettu Listen server mallin avulla. Mallin lisäksi pelissä hyödynnetään Steam-alijärjestelmää, jonka kautta server-toimintoja käytetään. Alijärjestelmän kautta saadaan pelaajien heidän nimensä ja profiilikuva.

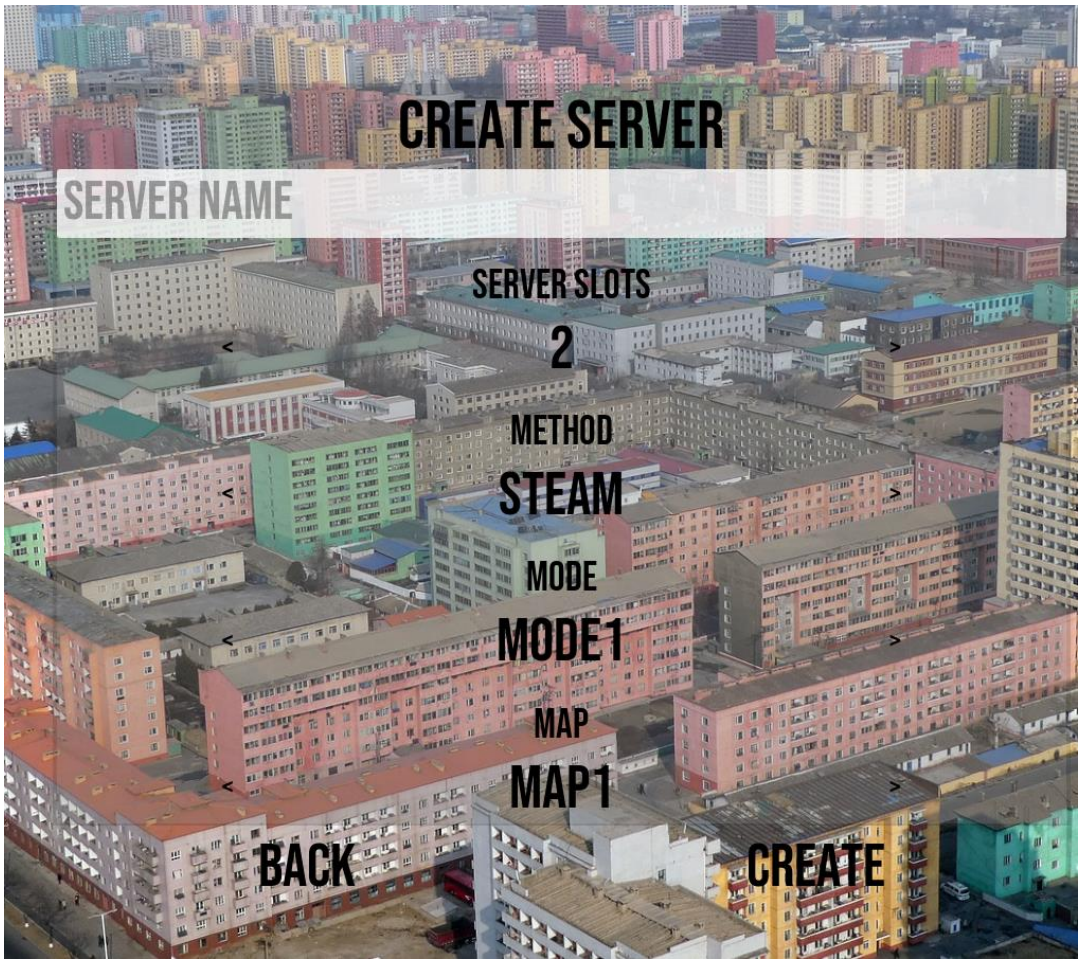
Moninpeli ominaisuudet ovat suuri osa peliä ja ne aktivoituvat heti, kun pelaaja käynnistää pelin. Päävalikon avautuessa kutsutaan Event Construct-tapahtumaa, joka tarkistaa, onko pelaajan laitteella online-alijärjestelmä-Steam. Jos pelaajalla on Steam, solmu palauttaa tosiarvon ja jatkaa if-lauseesta eteenpäin. Kuva 12. esittää pelin avautuessa tapahtuvaa solmuverkostoa, joka tarkistaa onko pelaajalla Steam-palvelua.



Kuva 12. Event Construct tapahtuman toiminnot

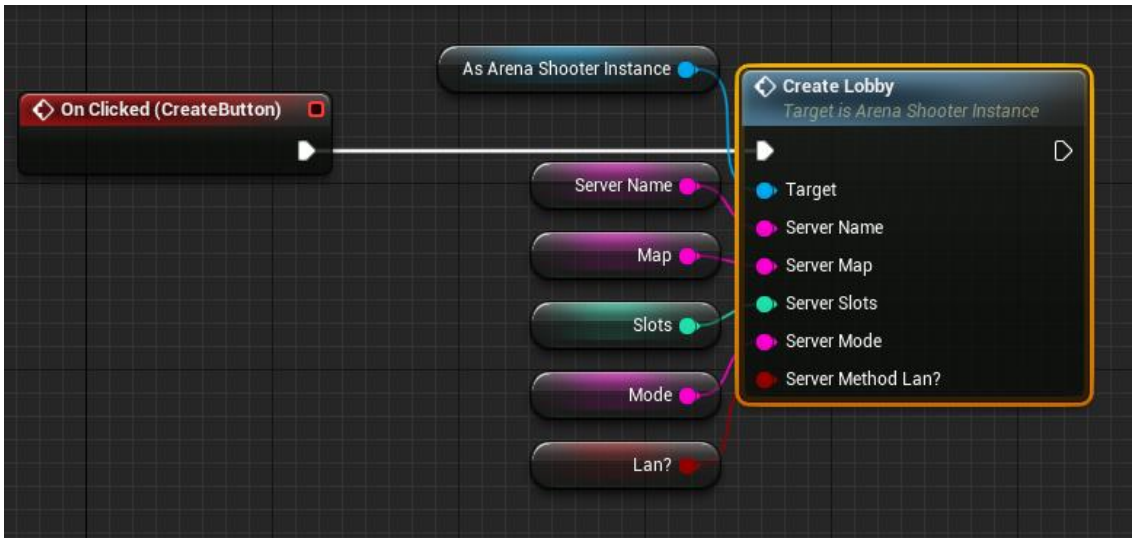
4.1.1 Hostaaminen

Pelin sisällä serverin luonti toimii create server-menun avulla. Hostaava pelaaja täyttää kentät, jotka määrittävät peliin vaikuttavat säännöt. Peliä voidaan hostata lokaalisti LAN-verkon -tai Steam-online-alijärjestelmän kautta. Kuva 13 esittää serverin luontia ja miltä sääntöjen määritys näyttää UI-elementeissä. Pelaajan täytyy antaa serverin nimi, kuinka monta pelaaja serveri sisältää (slots), hostaus metodi, pelimuoto ja pelin kartta.



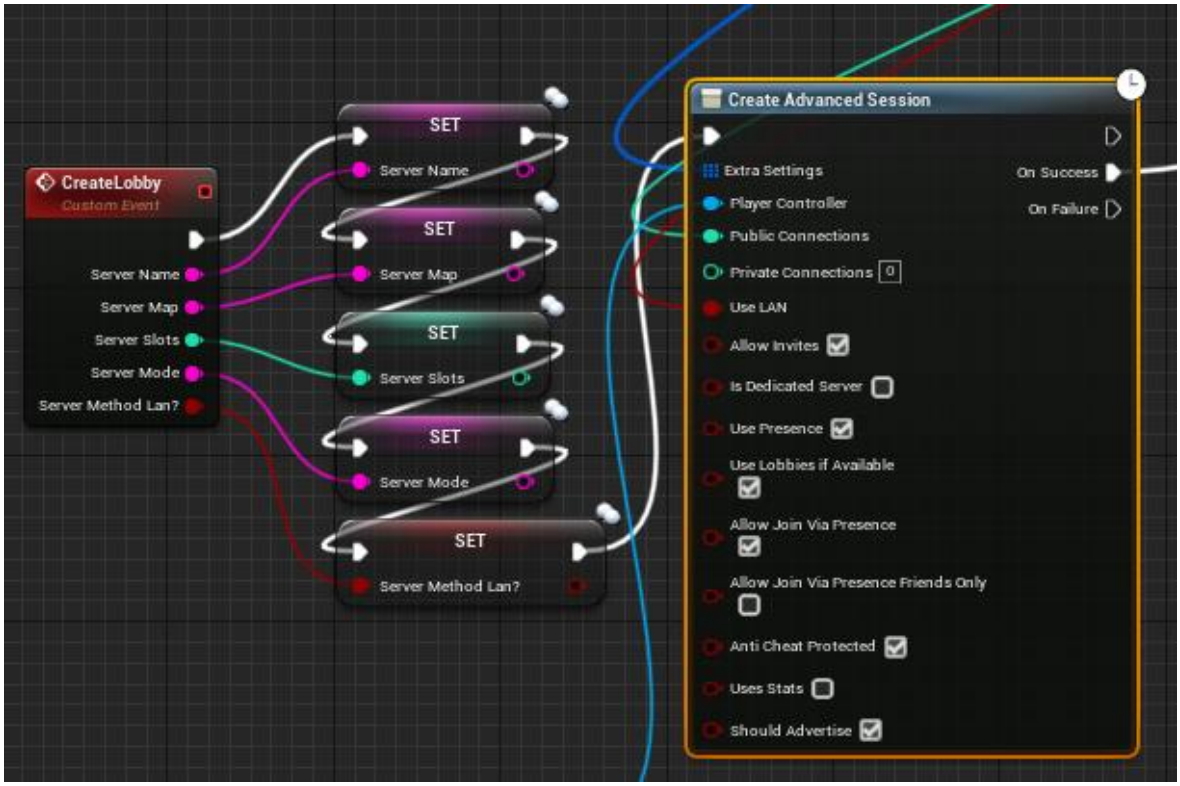
Kuva 13. Serverin luonti menu

Napin painallustapahtumaan liitetty blueprint suoritetaan, jonka jälkeen serverille luodaan aula. Kuva 14 esittää kuinka napin tapahtumaa kutsuttaessa edetään solmuverkossa kutsumaan create lobby funktiota. Create lobby funktiolle välitetään valmiiksi määriteltyjä muuttujia, jotka määrittävät pelin sääntöjä. Target parametri määrittelee kohteen, jossa funktio suoritetaan. Funktio sijaitsee Arena Shooter instanssissa, joten serverin luonti menun blueprinttiin luodaan muuttuja, jonka tyyppi on Arena Shooter instanssi.



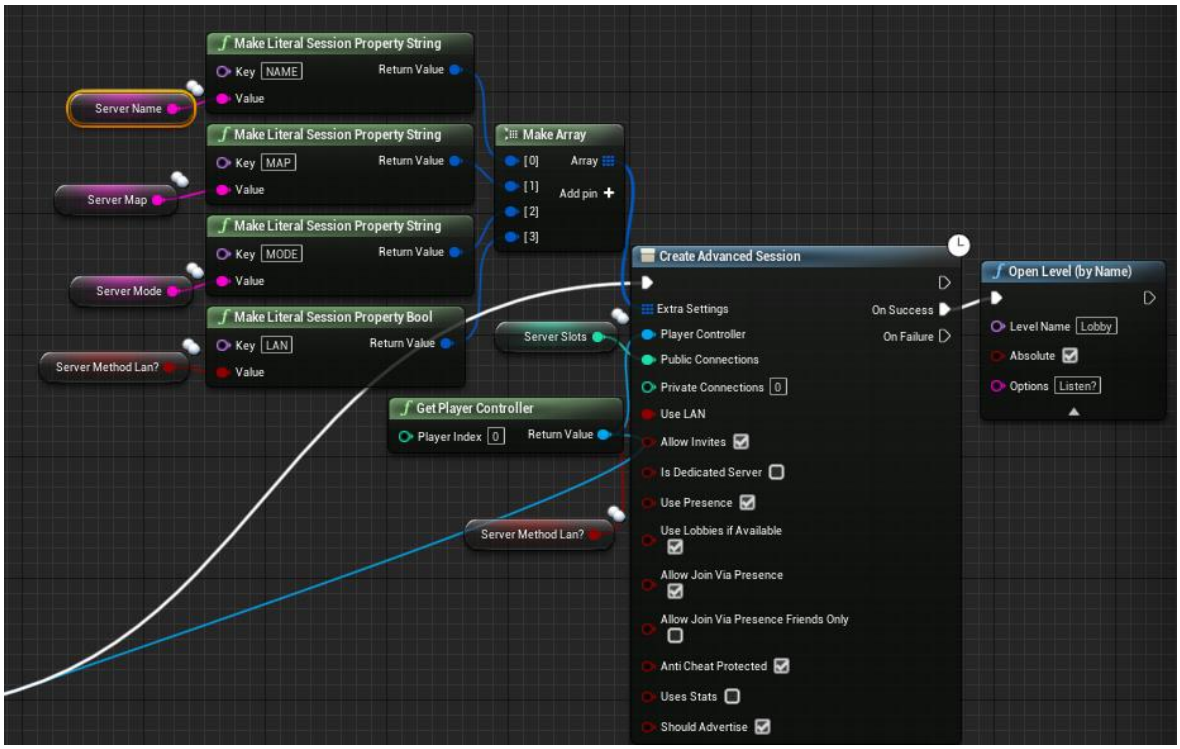
Kuva 14. Create lobby funktio

Create lobby funktiota kutsutaan peli-instanssin sisällä, ja sen sisällä tapahtuu todellinen toiminta. Kuva 15 esittää create lobby funktiota, jonka avulla luodaan serveri. Kun CreateLobby funktiota kutsutaan instanssin sisällä, ensimmäiseksi sijoitetaan kutsujan antamat arvot instanssin sisäisiin muuttujiin. Seuraavaksi kutsutaan Create Advaced Session solmua, joka luo itse serverin. Create Advanced Session solmun sisään sijoitetaan jo aikaisemmin määritetyt säännöt.



Kuva 15. CreateLobby-tapahtuma ja Advanced Session solmu

Advanced sessions solmun sisään luodaan ylimääräisiä sääntöjä hyödyntämällä Extra Settings parametriä. Peli-instanssin sisällä oleville muuttujille luodaan Sessiokohtainen ominaisuus merkkijono, jonka avulla Advanced Sessions solmu pääsee muuttujiin käsiksi. Kaikki sessiokohtaiset ominaisuus merkkijonot sijoitetaan taulukkoon, jonka avulla voidaan yksinkertaisesti hallita kaikkia pelin ylimääräisiä sääntöjä. Kuva 16 esittää kuinka säännöt luodaan ja sijoitetaan Advanced Sessions solmuun. Kuvassa on sijoitettu taulukon lisäksi pelaajan player controller, pelaajien määrä ja serverin hostaus metodi solmuun. Kun Advanced Session solmua kutsutaan, se yrittää luoda serverin ja jos se onnistuu, avataan serverin lobby, johon muut pelaajat voivat liittyä. Jos suoritus epäonnistuu ei tehdä mitään.



Kuva 16. Liitetyt parametrit Advanced Session solmussa

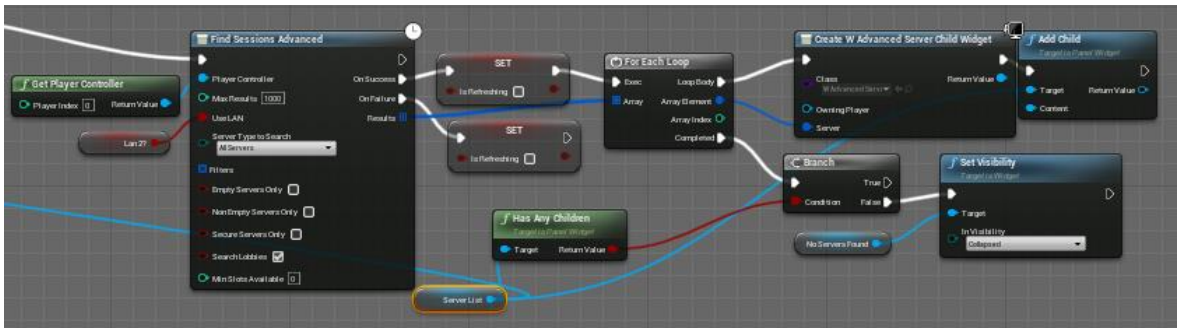
Aulan luomisen jälkeen host-pelaajalle luodaan pelaajakontrolleri ja pelaaja kirjautuu automaattisesti sisään lobbyyn. Pelaajan kirjautuminen kutsuu OnPostLogin-tapahtumaa Aulan pelimuodossa. Tapahtuma lisää omistavan peli-instanssin laitteen taulukkoon ja hakee peli-instanssissa serveriin sijoitetut ylimääräiset säännöt, jotka sijoitetaan pelimuodon omiin muuttujiin. Tämän jälkeen kutsutaan pelimuodon sisällä olevaan tapahtumaa UpdateServer, joka suoritetaan vain serverillä.

4.1.2 Liittyminen

Pelaajan liittyminen tapahtuu servereiden lista näkymän avulla. Lista näkymässä on mahdollista valita pelin hakumuoto painamalla yläkulmassa sijaitsevaa laatikkoa. Kun pelaaja on valinnut hakumuodon ja painanut refresh-nappia, alkaa peli hakemaan kaikkia mahdollisia servereitä. Serverin löytyessä ilmestyy ruudulle musta laatikko, joka kertoo pelaajalle serverin nimen, pelimuodon, pelaajamäärän ja latenssin. Painamalla Join nappia pelaaja voi liittyä serverille. Kuva 17 esittää serveri lista näkymää, joka näkyy pelaajille, jotka etsivät serveriä.

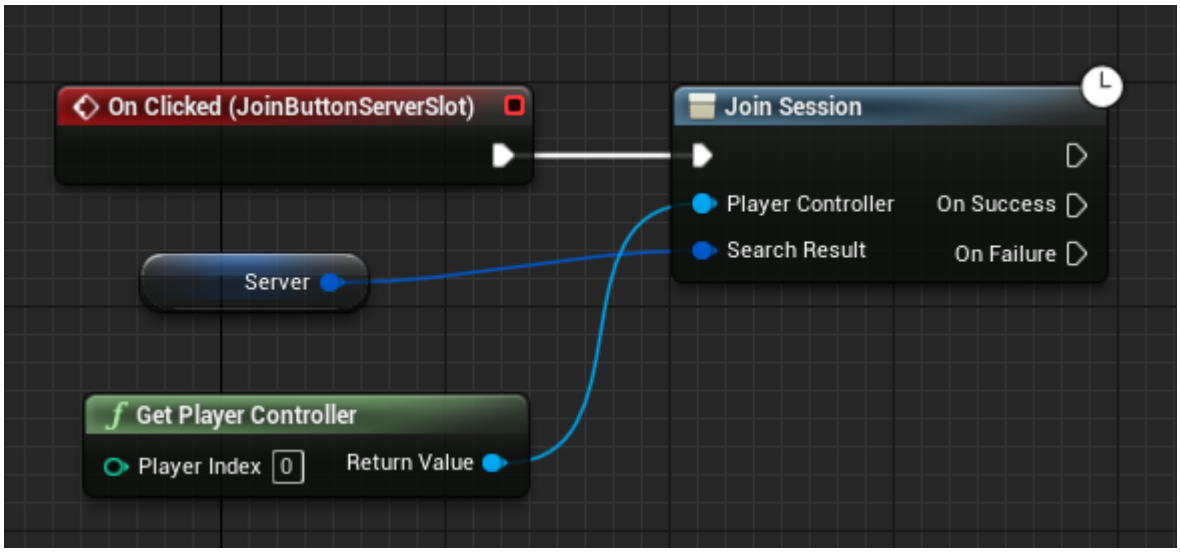
Kuva 17. Serveri lista näkymä

Pelaajan painaessa refresh-painiketta, kutsutaan RefreshServers-funktiota. Kuva 18 esittää Find Sessions Advanced solmua ja kuinka löydetyt serverit sijoitetaan serveri lista näkymään. Kutsuttaessa funktio tyhjentää löydettyjen serverien listan ja merkitsee isRefreshing muuttujan todeksi. Seuraavaksi solmuverkosto siirtyy Find Sessions Advanced solmuun, jonka tehtävänä on etsiä kaikki mahdolliset serverit, johon pelaaja voi liittyä. Find Sessions Advanced solmu palauttaa kaikki löydetyt serverit taulukossa. Servertaulukko käydään läpi silmukan avulla ja jokaiselle serverille luodaan oma widget-elementti, joka sijoitetaan serveri listaan näkyville.



Kuva 18. Servereiden haku ja sijoitus näkymään

Kun pelaaja haluaa liittyä serverille, hän painaa join-nappia. Join-painike suorittaa kutsun, joka kutsuu Join Session solmua. Kuva 19 esittää, kuinka pelaaja liittyy löydetylle serverille. Kuvassa Join Session solmun sisään ilmoitetaan pelaajan pelaajakontrolleri ja serveri, jonka sisään pelaaja haluaa liittyä. Liittyessään serverille pelaaja siirretään aulaan muiden pelaajien kanssa ja heille luodaan uusi pelaajakontrolleri, joka on määritelty aulapelimuodon luokka asetuksissa. Pelaajalle luotu uusi pelaajakontrolleri kutsuu OnPostLogin-tapahtuman aulan sisällä ja suorittaa tapahtuman toiminnot.



Kuva 19. Serverille liittymisen toiminnallisuus

4.1.3 Aula

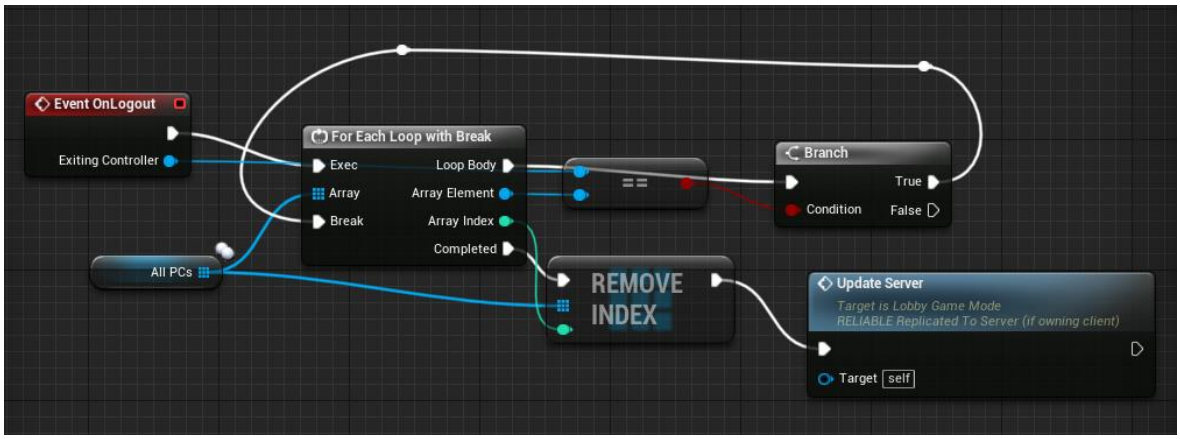
Kun pelaaja luo serverin tai liittyy serverille, sijoitetaan heidät aulaan. Aulassa pelaajat odottavat, kunnes host-pelaaja aloittaa pelin. Kuva 20. esittää pelin sisäistä aulaa. Aulanäkymässä vasemmalla on pelin nimi, pelin kartta, pelimuoto, aulasta poistumis- painike

ja pelin aloitusnappi, joka näkyy vain serverin hostaajalle. Oikeassa laidassa kuvaa on lista pelaajista sekä pelaajien maksimimäärä. Pelaajalla on mahdollisuus poistua palvelimelta Leave Lobby-painikkeen avulla, minkä tarkoitus on tuhota pelaajan paikallinen sessio. Aulan toiminnot suoritetaan aulan pelaajamuodon ja pelaajakontrollerin kautta.

Kuva 20. Aulan näkymä

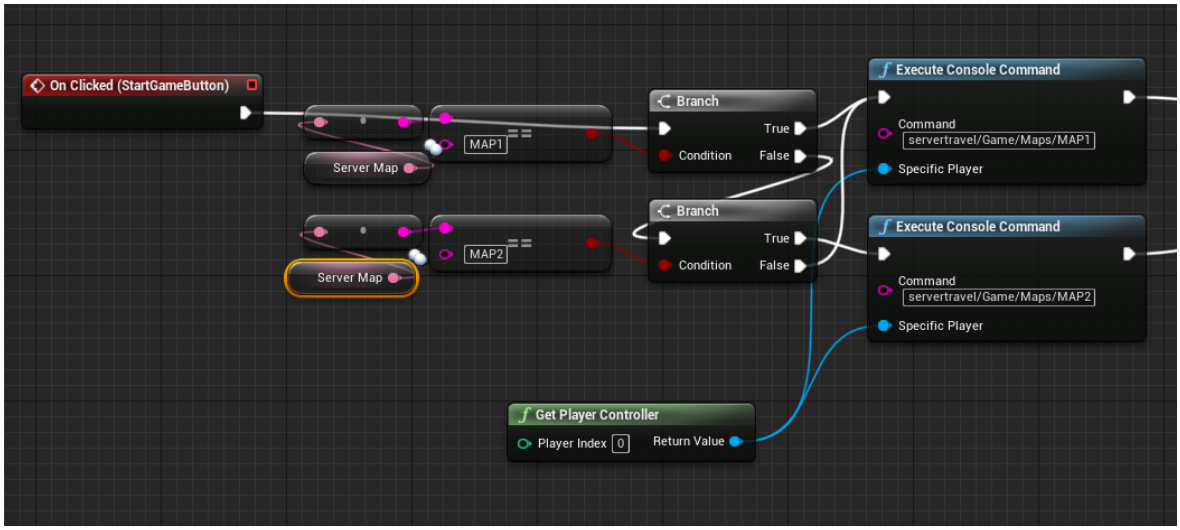
Pelaajien liittyessä aulaan on heille jo luotu aulan pelaajakontrolleri. Kun serveri havaitsee, että uusi pelaaja kirjautuu serverille, serveri kutsuu OnPostLogin-tapahtumaa. OnPostLogin-tapahtuma lisää uuden laitteen pelaaja taulukkoon, jonka jälkeen se kutsuu updateserver-funktiota. Funktio käy läpi kaikki yhdistäneet pelaajat silmukan avulla ja hakee niiden pelinsisäiset nimet ja profiilikuvat. Pelaajien tiedot haetaan Steam-alijärjestelmän kautta. Kun pelaajien tiedot on haettu, kutsutaan Add Player funktiota, joka lisää pelaajien tiedot aulan pelaajalistaan. Silmukan päättyessä päivitetään pelaaja määrä muuttuja kutsumalla UpdatePlayerNumbers funktiota, joka sijaitsee Aulan pelaajakontrollerissa.

Serverin omistajalla on mahdollisuus poistaa ei haluttuja pelaajia ulos sessiosta. Vain omistavalla pelaajalla on näkyvissä pelaajien nimen vieressä kick painike. Kuva 21. esittää kuinka pelaaja poistetaan pelaajalistalta, jonka jälkeen kutsutaan UpdateServer funktiota. Kick painike kutsuu KickPlayer funktiota, joka tuhoaa pelaajan session serverillä. Pelaajan poistuttua tapahtuma OnLogout aktivoituu ja potkittu pelaaja poistetaan yhdistettyjen laitteiden listalta, jonka jälkeen kutsutaan UpdateServer funktiota.



Kuva 21. OnLogout-tapahtuman toiminnallisuus

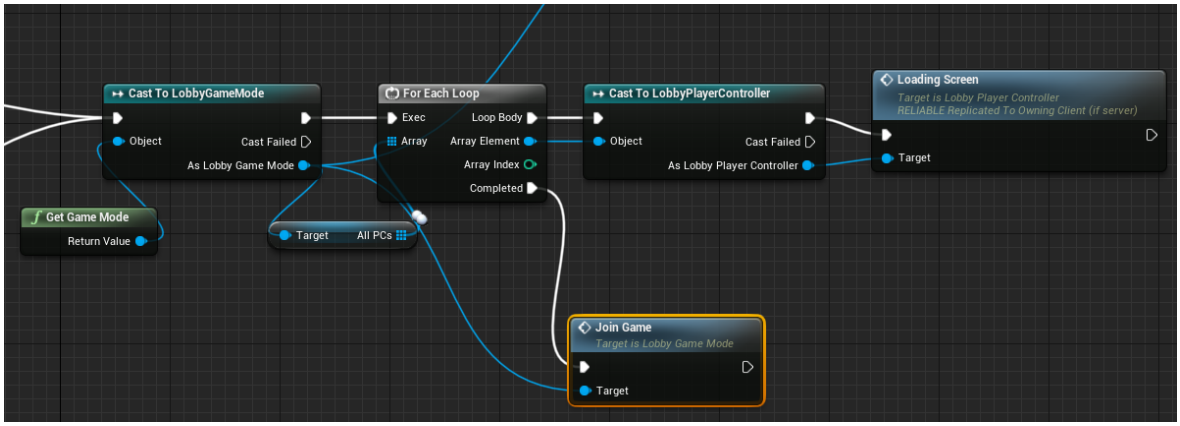
Serverin host pelaajan painaessa start-painiketta, peli tarkistaa, mikä kartta on valittuna. Pelin muilla ehdoilla ei ole vielä merkitystä, koska serverin täytyy tietää mihin karttaan pelaajat sijoitetaan. Kartan tarkastuksen jälkeen pelaajat lähetetään valittuun karttaan, konsoli komennon servertravel avulla. Servertravel-komento siirtää kaikki yhdistäneet pelaajat haluttuun sijaintiin. Kuva 22 esittää, kuinka prosessi suoritetaan solmuverkoston avulla. Kuvassa, kun painiketta on painettu, tarkistetaan valittu kartta, jonka jälkeen konsoli komento solmun avulla koko server siirretään karttaan. Servertravel komennon hyödyntäminen vaatii seamles-matkustuksen aktivoinnin pelimuodossa, jotta pelaajien yhteydet eivät katkea.



Kuva 22. Pelin aloitus solmuverkoston alku

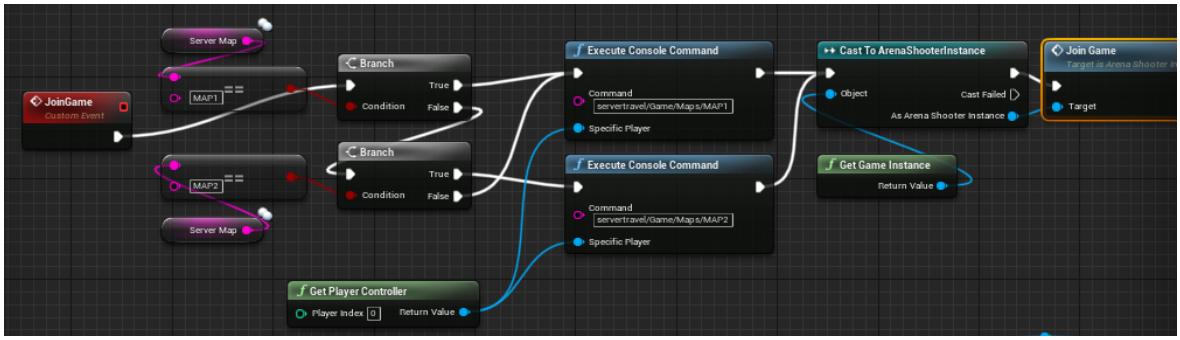
Seamles-matkustus vaatii siirtymä kartan, joten aulan ja kartan välille on luotu latausruutu. Kuva 23 havainnollistaa, kuinka siirtymisprosessi suoritetaan solmuverkossa. Kuvassa cast to solmut ovat viitauksia eri peliluokkiin. Siirtymisen alussa haetaan käytössä oleva

pelimuoto, jonka kautta viitataan LobbyGameModeen. LobbyGameModessa sijaitsee tiedot kaikista yhdistäneistä laitteista. Kaikki yhdistetyt laitteet sijoitetaan taulukkoon. Taulukko käydään läpi silmukan avulla, ja sen tarkoituksena siirtää pelaajat latausruutuun, jonka jälkeen, kutsutaan LobbyGameModessa sijaitsevaa funktiota Join Game.



Kuva 23. Pelaajien siirtymis- prosessi solmuverkossa

Join game-funktio tarkastaa valitun kartan ja suorittaa servertravel-komennon valitulle kartalle. Tällä kertaa servertravel-komennon avulla pelaajat sijoitetaan karttaan latausruudun sijasta. Kuva 24 esittää join game-funktion toimintaa aulan pelimuodon sisällä. Kun sekvenssi on suorittanut halutun servertravel-komennon, viitataan käytössä olevaan peli-instanssiin ja kutsutaan sen sisällä sijaitsevaa Join Game-funktiota. Instanssin sisällä olevan funktio tehtävänä on tuhota aulan sisällä sijaitseva sessio.



Kuva 24. JoinGame-tapahtuman toiminta

Pelaajien siirtyessä lopullisesti karttaan, kutsutaan BeginPlay-tapahtumaa kartan blueprintin sisällä. BeginPlay-tapahtuma hakee kaikki kartan sisällä olevat syntymispisteet ja sijoittaa pelaajat niihin, jonka jälkeen peli alkaa.

4.2 Replay-paketti

Replay-paketin toiminta on toteutettu C++-koodin avulla, koska Unreal Enginen replay-toimintoihin pääsee käsiksi vain sen avulla. Replay-toiminta sijoitetaan peli-instanssin sisään, koska instanssi on aina olemassa, vaikka pelin taso muuttuisi. Kuvassa 25 esitetään, kuinka replay-funktiot ja muuttujat on määritetty header-tiedostossa. Ensin määritetään kaksi muuttujaa RecordingName ja FriendlyRecordingName. RecordingName muuttuja määrittelee tallenteen nimen ja FriendlyRecordingName on mahdollista sijoittaa UI-elementteihin. Seuraavaksi määriteellään kolme funktiota: StartRecording, StopRecording ja StartReplay. Funktioiden määrittämisessä täytyy käyttää avainsanaa BlueprintCallable, jotta niitä voidaan kutsua blueprintin sisältä. Funktiot tämän jälkeen implementoidaan itse C++-tiedostoon.

```
7   #include "ReplayPackage.generated.h"
8
9   /**
10  *
11  */
12  UCLASS()
13  class REPLAY_API UReplayPackage : public UGameInstance
14  {
15      GENERATED_BODY()
16
17  public:
18
19      UReplayPackage();
20
21          UPROPERTY(EditDefaultsOnly, Category = "Replays")
22          FString RecordingName;
23
24      UPROPERTY(EditDefaultsOnly, Category = "Replays")
25      FString FriendlyRecordingName;
26
27      UFUNCTION(BlueprintCallable, Category = "Replays")
28          void StartRecording();
29
30      UFUNCTION(BlueprintCallable, Category = "Replays")
31          void StopRecording();
32
33      UFUNCTION(BlueprintCallable, Category = "Replays")
34          void StartReplay();
35
36
37  };
38
```

Kuva 25. Funktioiden ja muuttujien määritys header-tiedoston sisällä

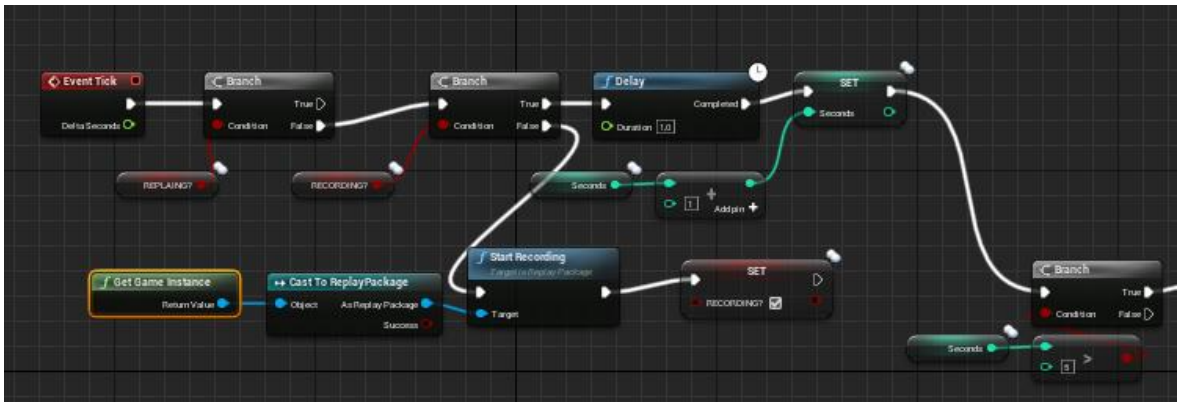
Header-tiedostossa määritetyt funktiot saavat kaiken toiminnallisuuden C++-tiedoston sisällä. Kuva 26 havainnollistaa kuinka funktiot ja niiden toiminta on luotu. StartRecording-

funktiossa luodaan dynaaminen taulukko. Taulukon sisään sijoitetaan ohitus sääntö, jonka tarkoituksena on määrittää MemoryStreamer käyttöönotto. StartRecordingReplay-funktio on Replay-järjestelmän sisäänrakennettu funktio, joka aloittaa uusinnan tallentamisen. Funktiolle annetaan syötemuuttujiksi tallenteen nimi, kuvaava nimi ja Options-taulukko, joka sisältää Streamer-olion valinnan. Seuraavaksi StopRecording-funktioon sijoitetaan StopRecordingReplay-funktio, joka on replay-järjestelmän sisäänrakennettu toiminto. Toiminto lopettaa uusinnan tallentamisen, jos tallentaminen on käynnissä. StartReplay-funktio hyödyntää PlayReplay-toimintoa, joka kutsuttaessa toistaa valitun uusinnan. Toiminto käyttää MemoryStreamer-oliota.

```
4    #include "ReplayPackage.h"
5
6
7    UReplayPackage::UReplayPackage()
8    {
9        RecordingName = "MyReplay";
10       FriendlyRecordingName = "My Replay";
11   }
12
13   void UReplayPackage::StartRecording()
14   {
15       TArray<FString> Options;
16       Options.Add("ReplayStreamerOverride=InMemoryNetworkReplayStreaming");
17       StartRecordingReplay(RecordingName, FriendlyRecordingName, Options);
18   }
19
20   void UReplayPackage::StopRecording()
21   {
22       StopRecordingReplay();
23   }
24
25   void UReplayPackage::StartReplay()
26   {
27       TArray<FString> Options;
28       Options.Add("ReplayStreamerOverride=InMemoryNetworkReplayStreaming");
29       PlayReplay(RecordingName, nullptr, Options);
30   }
31
32
```

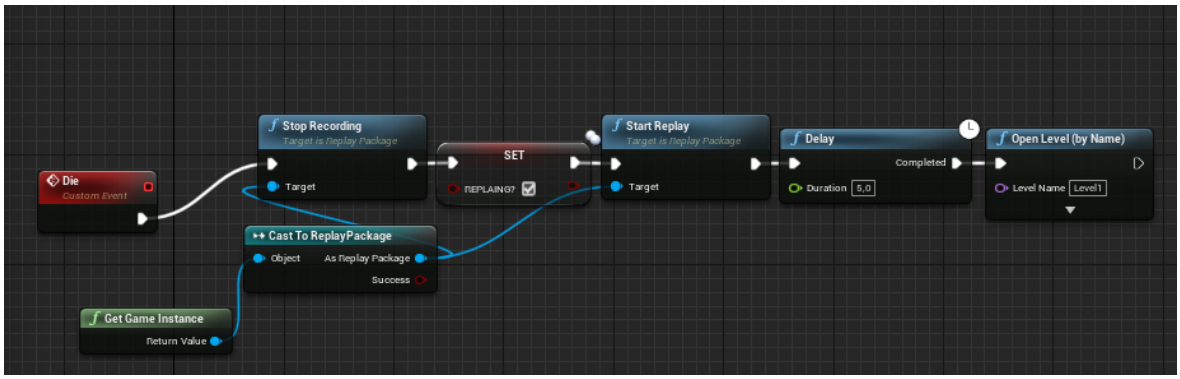
Kuva 26. Replay-järjestelmän toiminnan luominen

Pelin alkaessa BeginPlay-tapahtuma kutsuu StartRecording-funktiota, joka aloittaa uusinnan tallennuksen. Tämän jälkeen boolean-muuttujan arvo asetetaan todeksi. Replay-järjestelmän on tarkoitus lopettaa tallentaminen viiden sekunnin jälkeen ja aloittaa alusta ylikirjoittamalla vanha tallennus. Kuvassa 27 esitetään Tick-tapahtuman toiminnallisuutta. Kuvassa ensimmäiseksi tarkistetaan näyttääkö peli uusintaa, jonka jälkeen tarkistetaan tallentaako peli jo uusintaa. Jos peli ei tallenna uusintaa, kutsutaan StartRecording-funktiota. Seuraavaksi lisätään sekunti ajastimeen ja kun se saavuttaa viisi sekuntia niin kutsutaan StopRecording-funktiota, joka lopettaa tallentamisen.



Kuva 27. Ajastin ja tarkastukset Tick-tapahtumassa

Pelaajan kuoleman jälkeen toistetaan viisi viimeistä sekuntia pelitapahtumia. Kuvassa 28 on esitetty, kuinka pelaajalle toistetaan uusinta. Kun pelaaja kuolee, kutsutaan ensimmäiseksi StopRecording- funktiota, jotta toistettavaa tallennetta ei yli kirjoiteta. Seuraavaksi kutsutaan StartReplay-funktiota, joka toistaa pelaajalle viimeiset 5 sekuntia, mitä pelissä on tapahtunut. Uusinta näkyy pelaajalle sivustakatsojan perspektiivistä ja hänellä on mahdollisuus liikkua lentämällä uusinnan sisällä. Tämän jälkeen odotetaan viisi sekuntia ja siirretään pelaaja takaisin pelikentälle.



Kuva 28. Uusinnan toistaminen pelaajalle

5 Yhteenveto ja pohdinta

Työn tavoitteena oli kehittää uudelleenkäytettävä Replay-paketti, jota olisi mahdollista käyttää tulevaisuuden projekteissa. Replay-paketin oli tarkoitus tallentaa viimeisen viiden sekunnin aikana pelin tapahtumia ja tarvittaessa näyttää välittömiä uusintoja. Paketti myös täytyi olla vaivattomasti siirrettävissä projektista toiseen. Paketin kehityksessä päästiin tarvittaviin tavoitteisiin.

Työn Replay-paketti on luotu blueprint-systeemin ja C++-ohjelmointikielen avulla. Replay-paketti toimii Streamer-olioiden avulla, jotka vaativa C++-ohjelmointikieltä. Streamer-olioiden dokumentaatio on puutteellista ja vaikeutti työn tekemistä. Unreal Enginen Replay-järjestelmä toimii parhaiten koko pelin uusintojen käyttöön, joten sen takia siinä on paljon ongelmia, jos tarkoituksena on toistaa uusintoja välittömästi.

Työssä olisi voinut käyttää pelkästään C++-ohjelmointi kieltä ja optimoida peliä paremmin. Replay-paketti on siirretty ulkoiseen projektiin. Tulevaisuudessa pakettiin voitaisiin lisätä pelaajien ensimmäinen persoona tai tallentaa kokonaisia pelejä myöhemmälle käytölle. Toinen jatkokehitys mahdollisuus voisi olla Tick-funktion muuttaminen, jotta suorituskyky paranisi.

Lähteet

Epic Games. a. Functions. Unreal Engine documentation. Viitattu 23.3.2022. Saatavissa https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Blueprints/UserGuide/Functions/

Epic Games. b. Blueprint overview. Unreal Engine documentation. Viitattu 24.3.2022. Saatavissa https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Blueprints/Overview/

Epic Games. c. Balancing Blueprint and C++. Unreal Engine documentation. Viitattu 24.3.2022. Saatavissa https://docs.unrealengine.com/4.27/en-US/Resources/SampleGames/ARPG/BalancingBlueprintAndCPP/

Epic Games. d. Events. Unreal Engine documentation. Viitattu 25.3.2022. Saatavissa https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Blueprints/UserGuide/Events/

Epic Games. e. Custom Events. Unreal Engine documentation. Viitattu 25.3.2022. Saatavissa https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Blueprints/UserGuide/Events/Custom/

Epic Games. f. Networking Overview. Unreal Engine documentation. Viitattu 26.3.2022. Saatavissa https://docs.unrealengine.com/4.27/en-US/InteractiveExperiences/Networking/Overview/

Epic Games. g. Travelling in Multiplayer. Unreal Engine documentation. Viitattu 5.6.2022. saatavissa https://docs.unrealengine.com/4.27/en-US/InteractiveExperiences/Networking/Travelling/

Epic Games. h. Component Replication. Unreal Engine documentation. Viitattu 27.3.2022 saatavissa https://docs.unrealengine.com/4.26/en-US/InteractiveExperiences/Networking/Actors/Components/

Epic Games. i. RPCs. Unreal Engine documentation. Viitattu 27.3.2022 saatavissa https://docs.unrealengine.com/4.27/en-US/InteractiveExperiences/Networking/Actors/RPCs/

Epic Games. j. Game Mode and Game State. Unreal Engine documentation. Viitattu 28.3.2022. saatavissa https://docs.unrealengine.com/4.27/en-US/InteractiveExperiences/Framework/GameMode/

Epic Games. k. Advanced Sessions Plugin. Unreal Engine forums. Viitattu 30.3.2022 saatavilla https://forums.unrealengine.com/t/advanced-sessions-plugin/30020

Epic Games. l. Online Subsystem. Unreal Engine documentation. Viitattu 30.3.2022 saatavilla https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Online/

Epic Games. m. Online Subsystem Steam. Unreal Engine documentation. Viitattu 30.3.2022 saatavilla https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Online/Steam/

Epic Games. n. Replay System. Unreal Engine documentation. Viitattu 7.4.2022 saatavilla https://docs.unrealengine.com/4.26/en-US/TestingAndOptimization/ReplaySystem/

Epic Games. o. DemoNetDriver and Streamers. Unreal Engine documentation. Viitattu 7.4.2022 saatavilla https://docs.unrealengine.com/4.26/en-US/TestingAndOptimization/ReplaySystem/Streamers/

Epic Games. p. Gameplay Framework Quick Reference. Unreal Engine documentation. Viitattu 26.3.2022 saatavilla https://docs.unrealengine.com/4.27/en-US/InteractiveExperiences/Framework/QuickReference/

Epic Games. q. Blueprint Editor Reference. Unreal Engine documentation. Viitattu 19.04.2022 saatavilla https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Blueprints/Editor/

Lee, J. 2016. Learning Unreal Engine Game Development. E-kirja. Packt Publishing Ltd. Saatavilla https://books.google.fi/books?hl=fi&lr=&id=RFpLDAAAQBAJ&oi=fnd&pg=PP1&dq=what+is+unreal+engine&ots=ae3UL8xrqn&sig=3tb2cVbeeiBjQFSwhpIvwK-tMSU&redir_esc=y#v=onepage&q=what%20is%20unreal%20engine&f=false

G2A. Lockdown sees the rise of over 60s gamers, g2a data reveals. Viitattu 6.4.2022. Saatavissa https://www.g2a.co/lockdown-sees-the-rise-of-over-60s-gamers-g2a-data-reveals/

Rachel Cordone. Unreal Engine 4 Game Development quick start guide. E-Kirja. Packt Publishing Ltd. Saatavilla https://books.google.fi/books?hl=fi&lr=&id=szmbDwAAQBAJ&oi=fnd&pg=PP1&dq=introducti-on+to+unreal+engine&ots=_U9pKTKnSH&sig=sTU1aPNcbfZ1yr4FGk9R_yaILrk&redir_esc=y#v=onepage&q=introduction%20to%20unreal%20engine&f=false

Sufyan bin Uzayr. Mastering Unreal Engine. E-Kirja. CRC Press. Saatavilla https://books.google.fi/books?hl=fi&lr=&id=wGZjEAAAQBAJ&oi=fnd&pg=PP1&dq=introducti-on+to+unreal+engine&ots=rCW9f7oLNr&sig=9lzFrbse40bacFfhaeVZ_5rodBY&redir_esc=y#v=onepage&q=introduction%20to%20unreal%20engine&f=false