SAVONIA

OPINNÄYTETYÖ - **AMMATTIKORKEAKOULUTUTKINTO**
TEKNIIKAN JA LIIKENTEEN ALA

ONLINE MONINPELIN TOTEUTUS UNITY-PELI-MOOTTORILLA

TEKIJÄ: Eki Markkanen

SAVONIA-AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Tiivistelmä

Koulutusala Tekniikan ja liikenteen ala	
Koulutusohjelma/Tutkinto-ohjelma Tietotekniikan koulutusohjelma	
Työn tekijä(t) Eki Markkanen	
Työn nimi Online moninpelin toteutus Unity-pelimoottorilla	
Päiväys 11.12.2018	Sivumäärä/Liitteet 33/2
Ohjaaja(t) Tuntiopettaja Mikko Pääkkönen ja Lehtori Mikko Laasanen	
Toimeksiantaja/Yhteistyökumppani(t)	

Tiivistelmä

Opinnäytetyössä tutustutaan internetin välityksellä pelattavan moninpelin toimintaperiaatteisiin ja toteutukseen Unity-pelimoottorilla. Työn tarkoituksena on toteuttaa yksinkertainen internetin välityksellä pelattava moninpeli.

Aluksi työssä perehdytään verkkopelin toimintaperiaatteisiin. Lisäksi tutustutaan erilaisiin palvelinarkkitehtuureihin, joilla verkkopelin voi toteuttaa. Työssä myös käydään läpi protokollia, joihin verkon yli viestintä perustuu. Unity-pelimoottoriin löytyy runsaasti valmiita ratkaisuja, jotka ovat tehty helpottamaan verkkopelin kehitystä. Työssä tutustutaan neljään laajasti käytettyyn ratkaisuun. Työssä perehdytään tarkemmin Unity Networking HLAPI-rajapintaan ja sen tarjoamiin komponentteihin.

Lopputuloksena on toimiva verkkopeli, jossa kaksi pelaajaa voivat pelata vastakkain internetin välityksellä. Verkkopeli päädyttiin toteuttamaan Unityn omalla Unity Networking HLAPI-rajapinnalla.

Avainsanat
Unity, Verkkopeli, Online, Moninpeli, HLAPI

SAVONIA UNIVERSITY OF APPLIED SCIENCES

THESIS
Abstract

Field of Study Technology, Communication and Transport			
Degree Programme Degree Programme in Information Technology			
Author(s) Eki Markkanen			
Title of Thesis Making an Online Multiplayer Game with Unity Game Engine			
Date	12 December 2018	Pages/Appendices	33/2
Supervisor(s) Mr Mikko Pääkkönen, Lecturer and Mr Mikko Laasanen, Senior Lecturer			
Client Organisation /Partners			

Abstract

The aim of this thesis was to get familiar with the principles of online multiplayer games. The collected information was used to create a simple online multiplayer game. The purpose was make it with the Unity game engine.

At first, the focus was directed to the principles of online multiplayer games. Server architectures which are used to create an online game were familiarized with. After that, the interest of this thesis was pointed to the protocols on which the communication over the Internet was based on. There were lot of existing solutions available for Unity. These solutions were suitable for relieving the process of online game developing. In this thesis four widely used solutions were introduced.

As a result, a functioning multiplayer game was created where two players can play against each other via the Internet connection**.** The game ended up being made with the HLAPI-interface that was offered by Unity Networking.

Keywords
Unity, online game, HLAPI, online

SISÄLTÖ

1 JOHDANTO

Opinnäytetyön tarkoituksena on tutustua internetin välityksellä pelattavan moninpelin toteutukseen Unity-pelimoottorilla. Työssä käydään läpi erilaisia valmiita ratkaisuja, jotka ovat tehty helpottamaan verkkopelin toteutusta. Työssä vertaillaan näitä ratkaisuja eri näkökulmista ja valitaan yksi, jolla toteuttaa verkon yli pelattavan moninpelin demo kehitteillä olevaan mobiilipeliin.

Työssä myös käydään läpi verkkopelin toimintaperiaatteita. Lisäksi Tutustutaan erilaisiin palvelinarkkitehtuureihin, joihin verkkopelit turvautuvat. Tarkoituksena on saada kattava kuvaus moninpelien toiminnasta yleisesti ja toteutuksesta juuri Unity-pelimoottoria ajatellen.

Työssä toteutettava moninpeli on yksinkertainen mobiilipeli, jossa kaksi pelaajaa voi omilla puhelimillaan pelata vastakkain internetin välityksellä. Pelissä tarkoituksena on heitellä palloa kuppeihin vuorotellen pelissä asetettujen sääntöjen mukaisesti. Pelin säännöiksi voidaan määritellä, kuinka monta heittoa pelaajalla on vuorossaan ja kuinka moneen kuppiin pelaajien on osuttava. Pelaaja, joka ensimmäisenä osuu kaikkiin kuppeihin, voittaa pelin.

1.1 Moninpeli

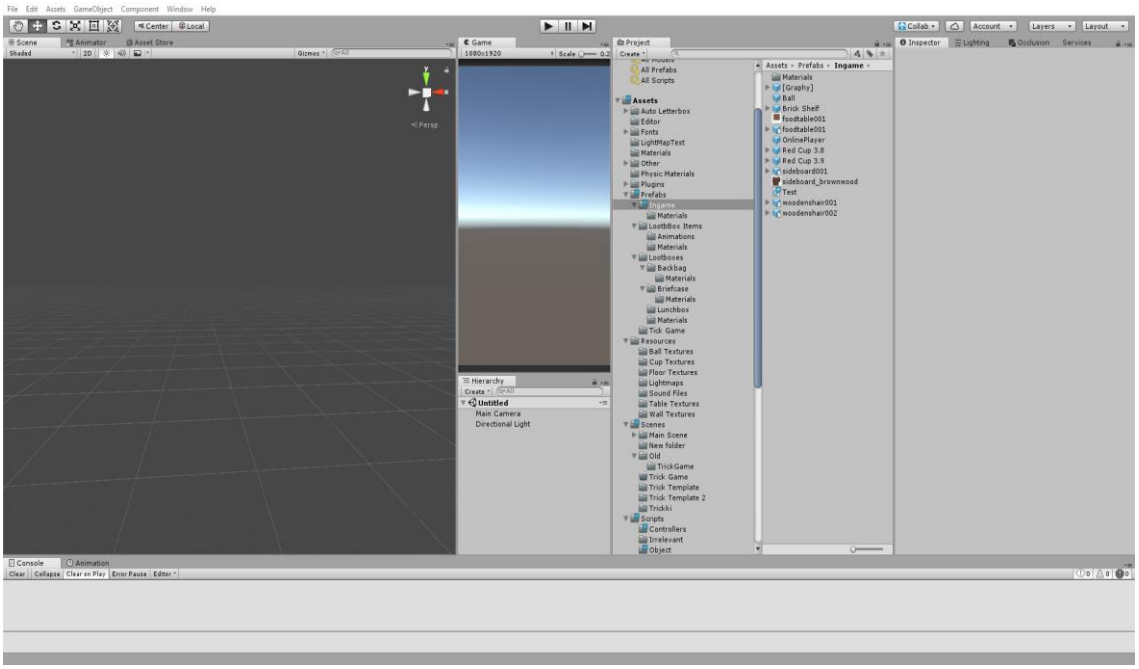
Moninpelillä tarkoitetaan peliä tai pelimuotoa, jossa pelaajat pystyvät pelaamaan yhden tai useamman pelaajan kanssa samassa pelissä. Moninpelistä riippuen pelaajat joko pelaavat samalla puolella tai toisiaan vastaan. Moninpelien ansiosta pelaajat eivät ole tekemisissä vain ennalta ohjelmoidun tekoälyn kanssa vaan pääsevät pelaamaan yhdessä muiden ihmisten kanssa.

Videopelien alkuaikoina moninpelit olivat usein samalta laitteelta pelattavia (local multiplayer), jolloin pelaajien täytyi olla samassa tilassa pelatakseen toistensa kanssa. Pelissä he joko vuorottelivat ohjaimen käyttöä tai olivat kytkeneet laitteeseen useamman ohjaimen. Samalta laitteelta pelatessa pelaajat jakoivat saman näyttöpäätteen.

Internetin yleistymisen myötä moninpelaaminen mullistui. Enää pelatakseen muiden ihmisten kanssa ei tarvinnut olla samassa tilassa, vaan pelaaminen onnistui internetin välityksellä (online multiplayer). Nykyään yhä useampi peli toimii verkon yli. Tässä työssä keskitytäänkin juuri internetin välityksellä toimivan moninpelin toimintaperiaatteisiin ja toteutukseen. (Computer Hope 2017.)

1.2 Unity

Unity on Unity Technologiesin kehittämä pelimoottori ja editori (Kuva 1.), joka mahdollistaa kaksi- ja kolmiulotteisten pelien kehityksen. Unityllä pystyy kehittämään pelejä sekä selaimelle, PC:lle, pelikonsoleille että mobiililaitteillekin. Unityn perusversio on ilmainen ja se tarjoaa hyvän vaihtoehdon pelien kehittämiseen. Unity sopii sekä ammattilaisille että aloittaville pelintekijöille. Siksi Unity onkin saavuttanut suuren suosion ja sen avulla on tehty jo puolet maailman peleistä. (Unity 2018a.)



KUVA 1 Unity editorinäkymä.

Unityn käyttäminen on ilmaista tietyillä rajoituksilla. Jos pelejä kehittävän yrityksen liikevaihto on yli $100 000 vuodessa on hankittava Unity Plus -lisenssin, joka maksaa $25 kuukaudessa. Plus-lisenssillä voi yrityksen liikevaihto olla enintään $200 000 vuodessa. Tämän jälkeen yritys joutuu ostamaan Unity Pro -lisenssin, joka on rajaton. Pro-lisenssin hinta on $125 kuukaudessa. (Unity 2018b.)

Unityssä pelintekemisessä käytetään hyödyksi komponenttisysteemiä. Erilaisille toiminnallisuuksille löytyy valmiit komponentit, jotka voi laittaa suoraan Unityn peliobjektille (GameObject). Esimerkiksi valaistukselle, fysiikoille, äänille ja animaatioille löytyy valmiit komponentit, joilla pystyy melko vaivattomasti hoitamaan kyseiset asiat.

Lisäominaisuudet, joita ei valmiilla komponenteilla voi toteuttaa, toteutetaan Unityssä C#-kielellä kirjoitettujen skriptien avulla. Erilaisilla skripteillä saadaan peliin toteutettua haluttuja toiminnallisuuksia ja ne toimivat samalla periaatteella kuin muutkin komponentit. Skriptien avulla pystytään myös vaikuttamaan peliobjektin muihin komponentteihin. Esimerkiksi peliobjektin sijainti on oma komponenttinsa (Transform), jonka arvoja muuttamalla skriptin avulla saadaan objektia liikutettua halutulla tavalla. (Unity 2018d.)

Jokaisen Unityn peliobjektiin liitetyn skriptin tulee periä MonoBehaviour-luokka (class). Tärkeimmät MonoBehaviour luokan tarjoamat funktiot ovat ’Start’ ja ’Update’ (Kuva 2.). Peleissä ruudussa tapahtuvia muutoksia sekunnissa käytetään termiä FPS (frame per second). Update-funktiota kutsutaan aina jokaisen ruudun välissä. Start-funktion suoritetaan aina ennen ensimmäistä ruutua (frame).

```
public class Test : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {

    }
}
```

KUVA 2 Unity skripti, jossa start ja update funktiot.

Asset Store toimii Unityn resurssien kauppapaikkana. Siellä on niin ilmaista kuin maksullistakin sisältöä, kuten esimerkiksi työkaluja, 3D-malleja, tekstuureita, ääniä, esimerkkejä ja muita hyödyllisiä resursseja. Nämä helpottavat pelin kehittämistä. (Unity 2018c.)

2 VERKKOPELIEN TOIMINTAPERIAATE

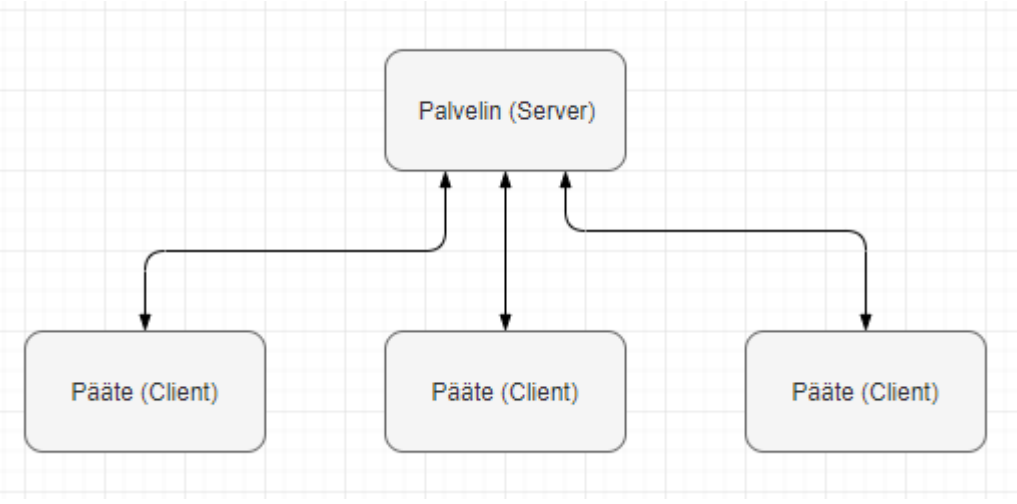
2.1 Yleisesti

Nykyään verkkopelin tekemiseen ei tarvitse syvää tietämystä internetin toimintaperiaatteista. Internetistä voi löytää lukuisia valmiita pohjia ja työkaluja, joiden avulla verkkopelin kehitys onnistuu melko vaivattomasti. Kuitenkin nämä ratkaisut ovat usein muiden tahojen ylläpitämiä, joten ne voivat olla maksullisia tai jollain muotoa rajattuja. Ymmärrys internetin toimintaperiaatteista helpottaa oikeiden työkalujen valintaa ja auttaa luomaan omia ratkaisuja.

Toimiakseen verkkopeli tarvitsee internetyhteyden. Pelaajien antamat syötteet kulkeutuvat siten internetin välityksellä muille pelaajille. Se, miten syöte kulkeutuu pelaajan laitteelta toiselle, riippuu täysin verkkopelin toteutuksesta. Verkkopelin toteutukseen taas vaikuttaa pelin tyyppi ja pelin luomat vaatimukset verkkopelille. Esimerkiksi hidastempoiset vuoropohjaiset pelit luovat täysin erilaiset vaatimukset kuin nopeaan reaktioaikaan pohjautuvat pelit.

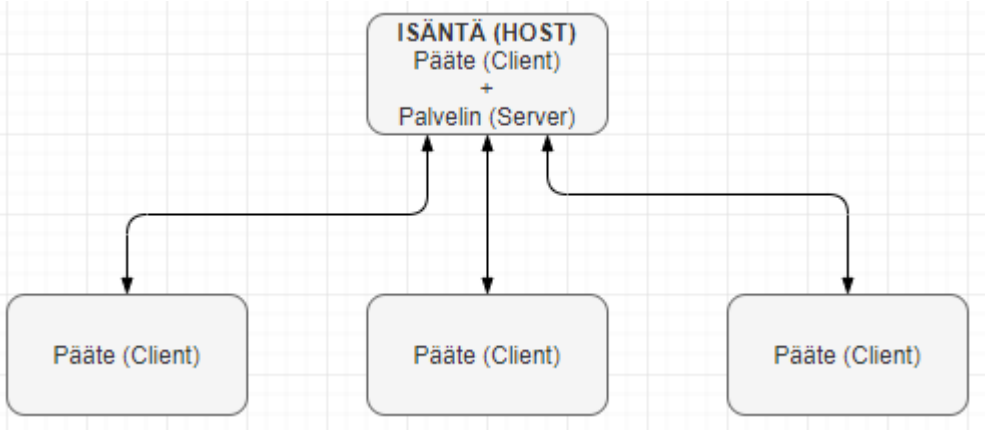
Verkkopeleissä esiintyy aina viivettä (latency). Tällä tarkoitetaan aikaa, joka kuluu datan kulkemiseen pelaajan ja palvelimen välillä. Viiveellä voidaan tarkoittaa myös datan edestakaisin (round trip delay) kulunutta aikaa. Viiveeseen vaikuttaa lukuisat eri muuttujat, kuten palvelimen fyysinen etäisyys, internetyhteyden nopeus ja kuinka pelin verkkopelilogiikka on tehty. Vähäinen viive on monissa pelityypeissä edellytys sulavalle pelikokemukselle, joten sitä pyritään vähentämään eri keinoin. (Duffy 2018.)

Tyypillisesti verkkopelit toimivat siten, että pelaajan laite toimii päätteenä (client), joka lähettää pelin kannalta tärkeää tietoa erilliselle palvelimelle (dedicated server). Palvelimelta tieto ohjataan kaikille päätteille (Kuva 3.). Tämä tieto voi olla esimerkiksi pelaajan hahmon sijainti, pelaajan hahmon suunta tai pelaajan antama syöte. Tiedon lähetys palvelimelta on ajastettua ja se tapahtuu tietyn monta kertaa sekunnissa (tick). Kun tämä tieto saapuu taas pelaajien laitteisiin, pelitilanne päivittyy tiedon mukaisesti. (Karim 2018.)



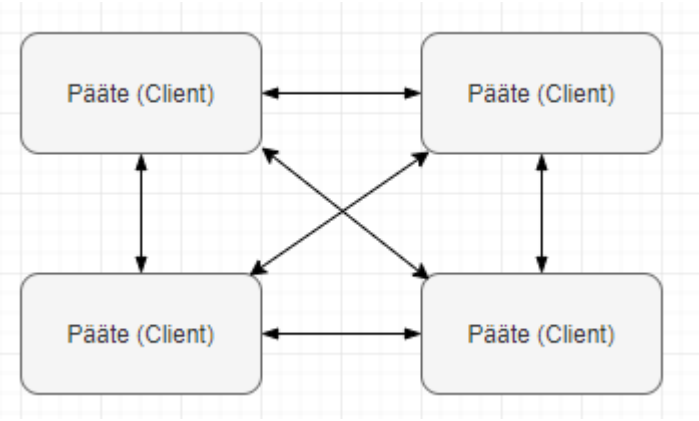
KUVA 3. Client-server ratkaisu, jossa on erillinen palvelin, johon 3 päätettä on yhdistänyt.

Verkkopeli voidaan toteuttaa myös siten, että pelaajan laite toimii palvelimena (listen server). Tällöin yksi pelaaja toimii pelin isäntänä. Kaikki verkon yli kulkeva data kulkee isäntälaitteen kautta (Kuva 4.). Tämä ratkaisu mahdollistaa verkkopelin toteutuksen ilman erillistä palvelinta, jolloin säästetään verkkopelin ylläpitokustannuksissa. Haittapuolena on pelisession riippuvuus isännän laitteesta ja sen nettiyhteydestä. Jos isännältä katkeaa yhteys internettiin, pelisessio katkeaa. Tätä ehkäisemään voi kehittää isännän siirto (Host migration) nimisen ratkaisun, jolloin isännän rooli vaihtuu yhdelle päätteistä. Isäntää vaihdetaan, kun vanha isäntä ei enää pysty peliä isännöimään. (Chris 2017.)



KUVA. 4 Yksi laite toimii sekä päätteenä että palvelimena.

Joissakin tapauksissa data voidaan lähettää päätteeltä suoraan muille päätteille ilman palvelinta (peer-to-peer). Tässä tapauksessa haluttu data lähetetään yhdeltä laitteelta kaikille muille samassa pelisessiossa oleville laitteelle (Kuva 5.). Tässäkään ratkaisussa ei tarvitse ulkopuolista palvelinta, minkä vuoksi verkkopelin ylläpito on halvempaa. Koska datan kulkemisessa esiintyy viivettä, on oma peli muita edellä. Tätä ratkaisua ei suositella käytettävän nopeatempoisiin peleihin. (Bevilacqua 2013.)



KUVA 5. Neljän päätteen peer-to-peer verkko.

Monissa tapauksissa moninpelien pelilogiikka tapahtuu suoraan palvelimella (authoritative server). Pelaajien antamat syötteet siis kulkeutuvat palvelimelle, jossa tarkastellaan, onko syöte oikeanlaista ja muokataan sitä tarvittaessa. Sitten palvelimelta lähetetään oikeanlaiseksi todettu data kaikille päätteille ja pelaajien pelitilanteet päivittyvät datan mukaisesti. Tällä pyritään ehkäisemään pelissä

huijaamista, koska pelaajat eivät pysty vaikuttamaan siihen, miten heidän dataansa tarkastellaan palvelimella. Tällaisissa tapauksissa palvelin joutuu tekemään enemmän työtä, joka voi nostaa moninpelin ylläpidon hintaa. Lisäksi pelaajan syötteeseen tulee viivettä, koska sen on kierrettävä palvelimen kautta ennen kuin se voidaan päivittää peliin (input-delay). Tämän voi estää kehittämällä ennakoinnin (client-side prediction), jolla pyritään ennakoimaan palvelimelta tuleva syöte. (Long 2016.)

2.2 Protokollat

Internetin toimintaperiaate on hyvin monimutkainen. Se pohjautuu useisiin protokolliin ja tekniikoihin, joiden avulla saadaan haluttu data kulkeutumaan oikeaan paikkaan oikeanlaisena. Internetin toimintaa voidaan kuvata OSI-mallilla (Open System Interconnection), jossa eri protokollat ja toiminnallisuuden vaiheet on jaettu seitsemään erilaiseen kerrokseen. (Kuva 6.) (Shaw 2018.)

Kaikki OSI-mallin (Kuva 6.) kerrokset ovat tärkeitä osia verkkopelin toiminnassa. Tässä yhteydessä kuitenkin paneudutaan erityisesti transport- ja network-kerrosten tärkeyteen verkkopelin toteutuksessa ja toiminnassa. Erityisesti tarkastelen Transport-kerroksen UDP- ja TCP-protokollia. Nämä protokollat ovat pääasiassa vastuussa siitä, miten data kulkeutuu palvelimen ja pelaajien välillä. (Shaw 2018.)

	OSI Layer	Protocol
Layer 7	Application	
Layer 6	Prensentation	
Layer 5	Session	
Layer 4	Transport	TCP, UDP
Layer 3	Network	IP
Layer 2	Data Link	
Layer 1	Physical	

KUVA 6. OSI-Mallin mukainen kuvaus internetin eri kerroksista.

2.2.1 IP (Internet Protocol)

IP on OSI-mallin 3. kerroksessa toimiva protokolla, jota käytetään datan kuljettamiseen verkon yli. Jokainen laite, joka on kytkettynä internettiin, omaa ainakin yhden yksilöllisen IP-osoitteen. IP-osoitteen ansiosta verkon yli kulkeva data pystytään ohjaamaan oikeaan paikkaan. Tämä data voi olla esimerkiksi nettisivu, kuva, video tai videopeleissä pelaajan antama syöte. (Rouse 2018.)

Jotta internetin välityksellä saadaan liikutettua erilaista dataa, se jaetaan palasiksi, joita kutsutaan paketeiksi (packet). Jokainen näistä paketeista sisältää sekä lähettäjän että vastaanottajan IP-osoitteen. IP-osoitteen avulla nämä paketit osaavat ohjautua oikeaan paikkaan. (Rouse 2018.)

Yksinään IP-protokolla ei riitä siihen, että saadaan data kulkemaan verkon yli. IP-protokollan rinnalla käytetään aina jotakin muuta protokollaa. Verkkopelaamisen näkökannalta käytetyin protokolla IP-

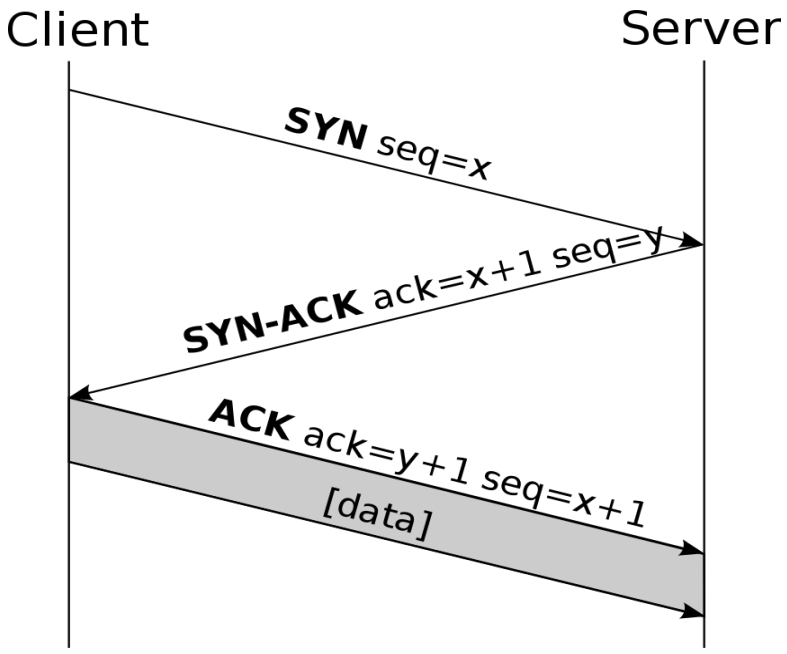
protokollan rinnalla on UDP (User Datagram Protocol). Joissakin tilanteissa myös TCP (Transmission Control Protocol) voi olla hyvä vaihtoehto IP-protokollan rinnalle. (Fiedler 2008.)

2.2.2 TCP (Transmission Control Protocol)

TCP on OSI-mallissa neljännessä kerroksessa toimiva protokolla. TCP ja IP toimivat yhdessä ja luovat selkärangan melkein kaikelle verkon yli tapahtuvalle liikenteelle (TCP/IP). TCP-protokollan avulla saadaan pidettyä yhtenäinen yhteys kahden laitteen välillä. Tämä mahdollistaa edestakaisen tiedonsiirron luotettavasti ilman, että paketteja häviää matkalla. TCP-protokollan ansiosta verkon yli liikkuvat paketit saapuvat samassa järjestyksessä perille kuin se on lähetettykin. (Unuth 2018.)

TCP-protokollan luotettavuus perustuu siihen, että se lisää lähetettyyn datapakettiin otsikon (header). Otsikko sisältää paketin kannalta tärkeää tietoa, kuten lähettäjän ja vastaanottajan porttinumeron, järjestysnumeron sekä määräajan. Lisäksi se sisältää tarkastustietoa, jolla tarkistetaan, onko lähetetty data muuttunut matkalla. Jos lähetettävä paketti saapuu perille, vastaanottaja lähettää takaisin kuittauspaketin (acknowledgment packet). Jos lähettäjä ei saa kuittausta lähetetystä paketista määräaikaan mennessä, on paketti voinut hukkua matkalla. Tässä tapauksessa paketti lähetetään uudestaan. Uutta pakettia ei lähetetä ennen kuin edellinen on saapunut onnistuneesti perille. (Unuth 2018.)

TCP-protokolla luo yhteyden kahden laitteen välille käyttäen kolmitiekättelyä (Kuva 7.). Tässä yhteyden aloittaja lähettää SYN-paketin kohdelaitteeseen. Kohdelaitteen saatua paketti se vastaa SYN/ACK-paketilla yhteyden aloittajalle. Jos aloittaja saa tämän paketin, se vastaa ACK-paketilla ja ACK-paketin saavuttua perille on luotettava yhteys muodostettu ja datan siirto voidaan aloittaa. (CS557: Basic TCP Mechanisms 2018.)



KUVA 7. TCP kolmitiekättely.

2.2.3 UDP (User Datagram Protocol)

UDP on TCP-protokollan tavoin OSI-mallin neljännessä kerroksessa toimiva protokolla. UDP-protokolla toimii IP-protokollan kanssa yhdessä. Toisin kuin TCP-protokolla, UDP on yhteyksetön. UDP-protokolla ei varmista paketin saapumista perille ja paketit saattavat saapua määränpäähän eri järjestyksessä, kuin ne on lähetetty. (Fiedler 2008.)

UDP-protokolla lisää vain pienen kerroksen IP-protokollan päälle. Lähettäessä pakettia UDP lisää pakettiin otsikon (header), joka sisältää lähettäjän ja vastaanottajan porttinumeron. IP-protokollan IP-osoitteen ansiosta paketti osaa ohjautua oikean laitteen IP-osoitteeseen. UDP-protokollan avulla lähettäjä ei kuitenkaan tiedä, onko paketti saapunut perille ja jos on, missä järjestyksessä. (UDP explained 2018.)

UDP ei ole yhtä luotettava protokolla kuin TCP. Kuitenkin UDP-protokollalla voidaan matkia TCP-protokollan luotettavuutta kehittämällä sen päälle tarkistusominasuuden, jolla voidaan tarkastaa, onko data saapunut määränpäähän. Tällöin kehittäjä saa vapaammat kädet ja datan kuljetus voidaan toteuttaa hyvin optimoidusti. Näin maksimoidaan nopeus ja saadaan tarvittava luotettavuus. (Fiedler 2008.)

2.2.4 UDP ja TCP verkkopelien kehityksessä

Erilaisia pelityyppejä luovat erilaisia vaatimukset verkkopelin toteutukselle ja valittavalle tekniikoille. Joissakin pelityypeissä on tärkeää, että data saadaan kuljetettua mahdollisimman nopeasti. Joissakin peleissä taas on tärkeämpää, että yhteys on luotettava ja jokainen paketti saapuu taatusti perille mutta nopeudella ei ole niin paljoa merkitystä. (Fiedler 2008.)

Monissa verkkopeleissä pelitilannetta ei kiinnosta kuin viimeaikaisin data. Tällöin ei tarvitse huolehtia siitä, että jokainen lähetetty data on saapunut perille. Lisäksi usein verkkopelissä on vaatimuksena mahdollisimman nopea tiedonsiirto.

UDP-on käytetyin protokolla pelien kehityksessä sen nopeuden ansiosta. UDP on TCP-protokollaan verrattuna melko yksinkertainen. Sen päälle on helppo lähteä rakentamaan omaa logiikkaa, jolla kuljettaa dataa verkon yli. Lisäksi sen päälle voi itse toteuttaa ominaisuuksia, joilla voidaan matkia TCP-protokollan luotettavuutta. (Fiedler 2008.)

TCP-protokolla ei ole niin käytetty verkkopeleissä, koska siinä on mahdollisesti suurempi viive kuin UDP-protokollassa. TCP-protokolla ei lähetä seuraavaa pakettia, ellei se ole saanut varmistuspakettia vastaanottajalta. Tällöin, jos paketti sattuu hukkumaan matkalla, lähettäjä odottaa aikamääreen verran ennen kuin lähettää saman paketin uudestaan. Tämä tuo voi tuoda huomattavaa viivettä datan kulkemiseen, mikä ei ole useimmiten verkkopelien kannalta suotavaa. (Fiedler 2008.)

UDP- ja TCP-protokollien käyttöä yhdessä ei suositella, koska ne saattavat sotkea toistensa toimintaa. Molemmat protokollat on tehty IP-protokollan päälle, jolloin näillä protokollilla lähetetyt paketit voivat vaikuttaa toisiinsa. (Fiedler 2008.)

3 VALMIIT VERKKOPELIRATKAISUT

Valmiiden ratkaisujen käyttäminen nopeuttaa verkkopelin kehitysprosessia. Näiden ansiosta ei tarvitse lähteä toteuttamaan verkkopelin monimutkaista järjestelmää aivan tyhjästä. Tässä työssä tutustuin neljään laajasti käytettyyn ratkaisuun, jotka ovat tehty juuri Unity-pelimoottorille.

Eri ratkaisut sopivat erilaisiin tilanteisiin ja pelityyppeihin. Ne myös käyttävät eri tekniikoita ja palvelinarkkitehtuureja. Tässä työssä käydään läpi Unityn tarjoama ratkaisu, Photonin Bolt- ja PUN-tuotteet sekä Forge Networking Remastered -niminen ratkaisu.

3.1 Unity Networking (Unet)

Unet on Unityn oma verkkopelaamisen mahdollistava ratkaisu. Unet on suoraan integroitu Unity-pelimoottoriin, eikä erillisiä lisäosia tarvitse ladata. Unity tarjoaa korkean tason HLAPI-rajapinnan (High Level Application Programmin Interface) sekä matalamman tason transport layer API-rajapinnan. Molemmat rajapinnat käyttävät UDP-protokollaa ja niiden ohjelmointikielenä toimii C#. (Unity 2018k.)

Unet omaa erittäin hyvän dokumentaation. Dokumentaatio kuvaa kattavasti HLAPI- ja transport layer API-rajapinnan toimintaa ja ominaisuuksia. Dokumentaatiosta löytyy video-oppaita sekä esimerkkejä, kuinka toteuttaa tiettyjä ominaisuuksia.

Unityn personal-versiolla voi olla 20 yhtäaikaista käyttäjää kerralla yhdistettynä Unityn tarjoaman välityspalvelimen kautta. Plus versio mahdollistaa 50 yhtäaikaista yhteyttä ja Pro versiolla niitä voi olla 200. Jos yhtäaikaiset pelaajat ylittävät tämän määrän, tulee tiedonsiirto maksamaan 0.49$/GB. (Unity 2018j.)

Transport layer API-rajapinta on matalamman tason rajapinta, joka antaa kehittäjälle vapaammat kädet tehdä ominaisuuksia verkkopeliin. Tämän rajapinnan avulla voidaan totuttaa vapaammin verkkopelin datan kuljetus, jolloin voidaan optimoida toteutus juuri omaa peliä ajatellen. Transport layer API-rajapinta lisää pienen kerroksen käyttöjärjestelmän soketti-pohjaisen verkkologiikan päälle. (Unity 2018k.)

HLAPI-rajapinta on korkean tason rajapinta. Tämä rajapinta on rakennettu suoraan Transport layer API-rajapinnan päälle. Kaikki yleisimmät verkkopelin kehityksessä tarvittavat ominaisuudet ovat valmiiksi tehty. HLAPI-rajapinnan avulla kehittäjän ei tarvitse miettiä matalamman tason toteutuksesta ja sen avulla saakin pelin kommunikoimaan verkon yli vaivattomasti tietämättä juurikaan verkkopelin toimintaperiaatteista.

Tämä rajapinta mahdollistaa yhden pelaajien laitteista toimivan palvelimena ja pelin isäntänä. Muut laitteet toimivat taas päätteinä. Tällöin kaikki pelissä kulkeva data kulkee isännän kautta muille päätteille. (Unity 2018f.)

HLAPI-rajapinta on avointa lähdekoodia (open source). Se tarkoittaa, että kehittäjät pystyvät vapaasti tutustumaan ja tekemään muokkauksia rajapinnan lähdekoodiin. Tämä mahdollistaa rajapinnan muokkaamisen juuri oman pelin vaatimuksiin sopivaksi. (Ólafsson 2015.)

Unity tulee korvaamaan HLAPI- ja Transport layer API-rajapinna uudella verkkorajapinnalla. Lisäksi välityspalvelin ja matchmaking-palvelu tullaan korvaamaan. Tarkoituksena on uudistaa Unet vastaamaan paremmin käyttäjän tarpeita. Uusi verkkorajapinta tulee Unityyn versiossa 2018.4. Vanhoja palveluita tuetaan vuoteen 2022 asti. (Unity 2018i.)

3.2 Photon

Photon on Exit Gamesin kehittämä alusta, joka tuottaa sovelluksia verkkopelien kehittämisen helpottamiseksi. Photonilla on useita tuotteita, jotka helpottavat verkon yli pelattavien moninpelien toteutusta. Photonin tuotteet ovat helposti integroitavissa Unity-pelimoottoriin. (Photon 2018b.)

Kaikki Photonin kehittämät tuotteet toimivat Photonin oman Photon Cloud -nimisen palvelinratkaisun kautta. Tällöin Photon huolehtii palvelimen ylläpidosta ja skaalautuvuudesta. Tämä auttaa kehittäjiä keskittymään pelkästään pelin kehitykseen, eikä tarvitse huolehtia palvelimen ylläpitoon liittyvistä asioista. Photon Cloudilla on useita palvelimia, mitkä sijaitsevat ympäri maapalloa. Näin mahdollistetaan mahdollisimman pieni viive palvelimien ja päätteiden välillä. (Photon 2018a.)

3.2.1 PUN (Photon Unity Networking)

PUN on kehitetty Unity-pelimoottoria ajatellen parantamaan Unityn omaa verkkorajapintaa. PUN tukee kaikkia Unityn tukema alustoja. Siitä löytyy valmiiksi useita verkkopeliin tarvittavia ominaisuuksia. Sillä voi käyttää sekä UDP- että TCP-protokollaa. PUN käyttää ohjelmointikielenään C#, joka on sama kieli kuin Unitylläkin ohjelmoitaessa. (Photon 2018a.)

PUN käyttää client-server-arkkitehtuuria, jolloin kaikki pelissä verkon yli liikkuva data kulkeutuu erillisen palvelimen yli. Kehittäjä voi joko itse ylläpitää palvelinta tai käyttää Photonin tarjoamaa Photon Cloud -palvelinratkaisua. Lisäksi PUN sisältää matchmaking API-rajapinnan, jolla pelaajat saadaan helposti yhdistettyä samaan peliin toistensa kanssa. (Photon 2018a.)

PUN:n hinnoittelu pohjautuu yhtaikaisten pelaajien määrään. Jos pelissä on alle kaksikymmentä yhtäaikaista pelaajaa, on tuote ilmainen. Maksamalla $95 viittä vuotta kohti voi pelissä olla sata yhtäaikaista käyttäjää. $95 kuukausimaksulla saa tuen viiteensataan pelaajaan asti. Kallein paketti on $185 kuukaudessa, jonka joutuu maksamaan jokaista tuhatta yhtäaikaista pelaajaa kohden. Lisäksi

kaikesta Photon Cloud -palvelun kautta kuljetetusta datasta joutuu maksamaan 0.05$/3GB. (Photon 2018a.)

3.2.2 Bolt

Bolt on kehitetty nopeatempoisia verkkopelejä ajatellen. Bolt on hyvin monipuolinen ja tukee useita eri palvelinarkkitehtuureja. Boltilla voi käyttää joko erillistä palvelinta tai Listen server -arkkitehtuuria, jossa palvelimena toimii pelaajan laite. Lisäksi Bolt tukee peer-to-peer-arkkitehtuuria, jossa pelaajien laitteet lähettävät suoraan dataa toisilleen. Välttyäkseen palomuurin ja nimenmuutoksen tuomilta ongelmilta kehittäjä voi käyttää Photonin tarjoamaa välityspalvelinta. (Photon 2018c.)

Bolt tukee Windows, MacOS, Linux, Android sekä IOS käyttöjärjestelmiä. Boltissa on tuki myös Xbox-konsolille. Playstationille löytyy kokeellinen tuki ja Nintendon Switch konsolille tuki on tulossa. (Photon 2018c.)

Bolt sisältää useita valmiita työkaluja ja toiminnallisuuksia, joilla pyritään minimoimaan verkkopelin kehitykseen kulutettava aika. Siitä löytyy PUN:n kaltainen pilvipohjainen matchmaking-palvelu, jolla saadaan pelaajien laitteet yhdistämään toisiinsa. (Photon 2018c.)

Boltin hinnoittelu pohjautuu PUN:n tavoin yhtaikaisten pelaajien määrään. Jos pelissä on alle kaksikymmentä yhtäaikaista pelaajaa, on tuote ilmainen. Maksamalla $95 viittä vuotta kohti voi pelissä olla sata yhtäaikaista käyttäjää. $95 kuukausimaksulla saa tuen viiteensataan pelaajaan asti. Kallein paketti on $185 kuukaudessa, jonka joutuu maksamaan jokaista tuhatta yhtäaikaista pelaajaa kohden. Lisäksi kaikesta Photon Cloud -palvelun kautta kuljetetusta datasta joutuu maksamaan 0.05$/3GB. (Photon 2018d.)

3.3 Forge Networking Remastered

Forge Networking Remastered on Bearded Man Studiosin kehittämä verkkopelikirjasto. Forgen lähdekoodi on avointa (open source), joten sitä voi vapaasti tarkastella ja muokata. Forge on kehitetty juuri Unity pelimoottorille. Forge tukee sekä TCP- että UDP-protokollaa ja sen ohjelmointikielenä toimii C#. Forge toimii kaikilla soketti-pohjaisilla alustoilla.

Forge on kehitetty helpottamaan verkkopelin toteutusta. Se sisältää paljon tärkeitä kirjastoja verkon yli kommunikaation kehittämistä helpottamaan. Forge tarvitsee erillisen palvelimen toimiakseen. Forge itse ei tuota palvelua, joka mahdollistaisi valmiin palvelinratkaisun. Forgen käyttäminen on ilmasta, eikä sillä ole rajoituksia yhtäaikaisille pelaajille. (Forge 2018.)

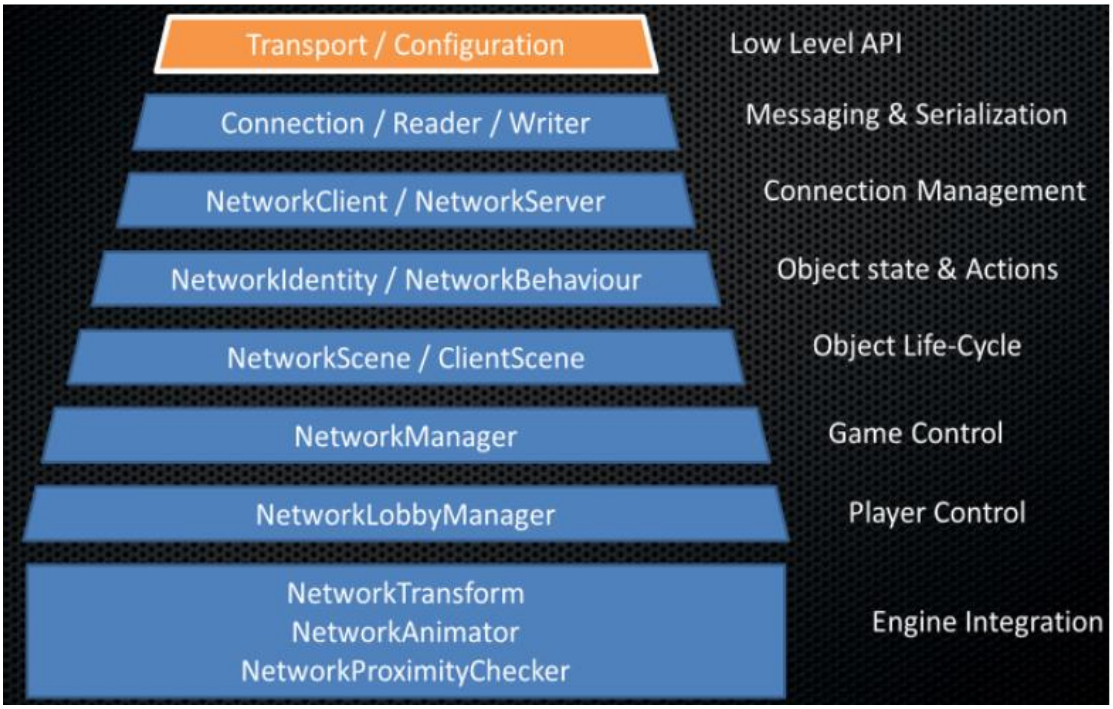
Forgella tyypillisesti käytettävä palvelinarkkitehtuuri on käyttää yhtä päätteistä palvelimena. Myös erillisen palvelimen käyttäminen on mahdollista. Forge myös tarjoaa erilliselle palvelimelle asennet-

tavan MasterServer ohjelmiston, jonka avulla voidaan listata päätteitä, isäntiä ja isännän luomia pelejä. Tämä mahdollistaa sen, että päätteet voivat yhdistää MasterServerin listaamaan peliin ilman erillistä IP-osoitteen syöttämistä. (Forge 2018.)

4 UNITY NETWORKING HLAPI-RAJAPINTA

Verkkopelin toteutus HLAPI-rajapinnalla on tehty hyvin helpoksi. Se sisältää runsaasti valmiita Unityn luomia palveluita ja komponentteja hoitamaan lähes kaiken verkkopeliin vaadittavan toiminnallisuuden. (Unity 2018f.)

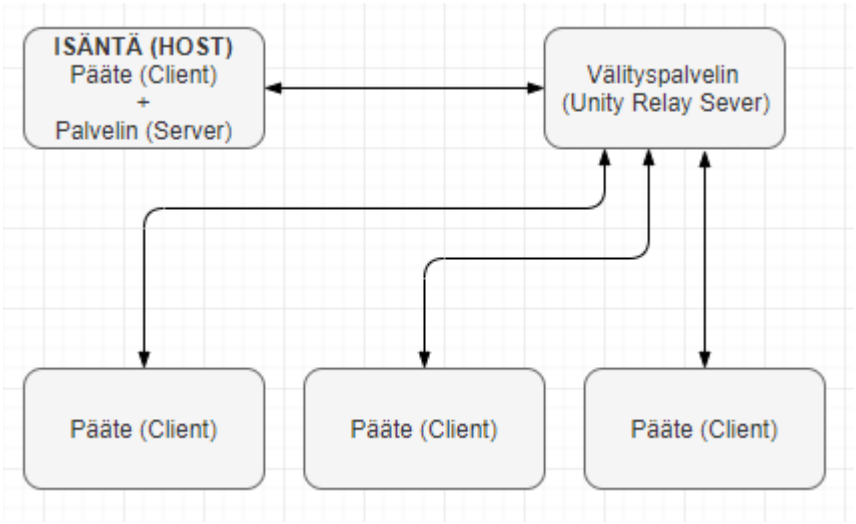
HLAPI-rajapinta on tehty matalamman tason Transport layer API-rajapinnan päälle (Kuva 8.). HLAPI-rajapinta on palvelinauktoritatiivinen järjestelmä, vaikkakin se tarjoaa mahdollisuuden yhden laitteista toimivan sekä palvelimena, että päätteenä. Tämä mahdollistaa sen, että erillistä palvelinta ei tarvita. (Unity 2018f.)



KUVA 8. Unity HLAPI-rajapinnan kerrokset. (Unity 2018f.)

4.1 Palvelut

HLAPI-rajapintaa käytettäessä voidaan käyttää Unityn tarjoamaa välityspalvelinta (Kuva 9.). Välityspalvelinta käytettäessä kaikki palvelimen ja päätteiden välillä kulkeva data kuljetetaan erillisen palvelimen kautta. Tällä vältytään palomuurin ja nimenmuutoksen tuomilta ongelmilta. (Unity 2018f.)



KUVA 9. Unity Networking toimintaperiaate.

Yksi kätevä HLAPI -rajapinnan tuoma palvelu on matchmaking-palvelu. Matchmaking-palvelun avulla voidaan joko luoda peli tai liittyä jo luotuun peliin. Tämän avulla pelaajat voivat yhdistää toistensa laitteisiin ilman erillistä IP-osoitteen syöttämistä. Matchmaking-palvelu toimii Unityn tarjoaman välityspalvelimen kautta, eikä luo suoraa yhteyttä laitteiden välille. (Unity 2018p.)

4.2 Komponentit ja ominaisuudet

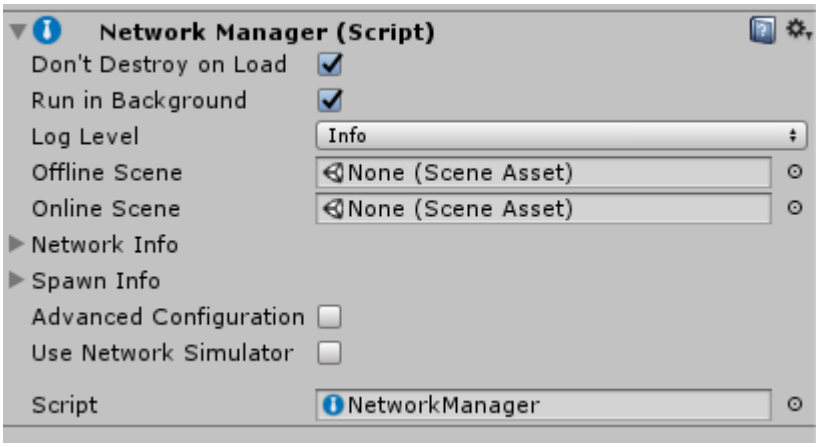
Network identity on Unityn HLAPI-rajapinna tarjoama valmis komponentti. Tämän komponentin on Unity verkkorajapinnan selkäranka. Sen avulla saadaan Unityn peliobjektit (GameObject) verkon yli toimiviksi. Network identity kompnentti sisältää peliobjektin verkkoteidot (Kuva 10.). Network Identity-komponentti liitetään peliobjektiin, joka halutaan verkon yli toimivan. Tämän komponentin avulla päätetään hallitseeko peliobjektia palvelin vai jokin päätteistä. (Unity 2018f.)

KUVA 8. Esimerkki Network Identity komponentin sisältämästä tiedosta ajon aikana.

Jotta Unityn verkko-ominaisuuksia voi käyttää omassa skriptissä, pitää luokan periä valmis NetworkBehaviour-luokka. Toimiakseen NetworkBehaviour tarvitsee Network Identity komponentin liitetyksi peliobjektiin. (Unity 2018l.)

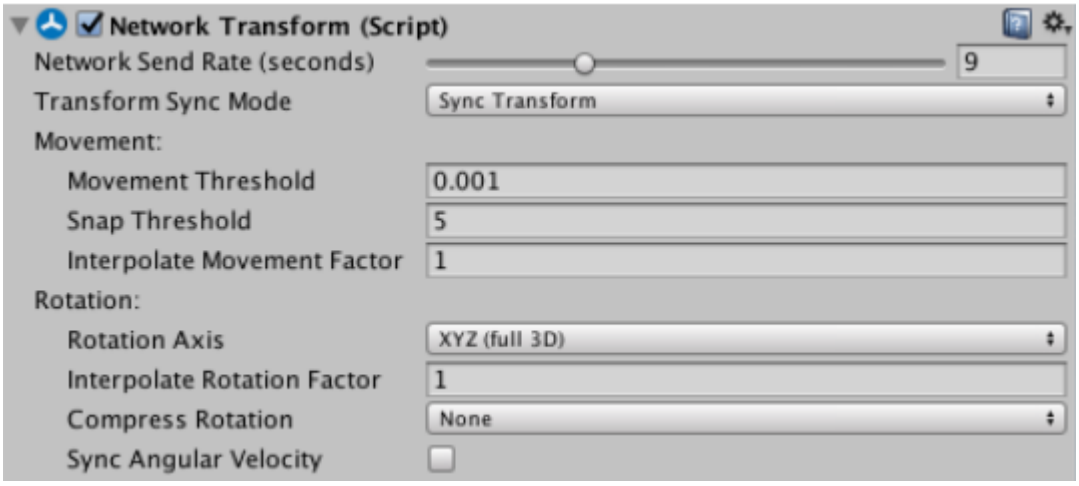
NetworkBehaviour skripti tuo lukuisia funktioita päätteelle sekä palvelimelle. Sen avulla voidaan lähettää etäkutsuja palvelimelta päätteille ja toisinpäin. Lisäksi sillä voidaan synkronoida muuttujien arvo palvelimelta päätteille. (Unity 2018l.)

NetworkManager on Unityn HLAPI-rajapinnan tarjoama valmis luokka (Kuva 11.). NetworkManager-luokkaa käytetään ohjaamaan HLAPI-rajapinnan tarjoamia toiminnallisuuksia, kuten palvelimen ja päätteen yhteyksiä. NetworkManager-luokkaa voi jatkaa omilla virtuaalifunktioilla. Virtuaalifunktioiden avulla voidaan toteuttaa toiminnallisuuksia, kun peli havaitsee tietyn tapahtuman. Esimerkiksi palvelimeen yhdistäminen tai yhteyden muodostuksen epäonnistuminen voivat olla tällaisia tapahtumia. (Unity 2018e.)



KUVA 9. Network Manager komponentti.

NetworkTransform on Unityn HLAPI-rajapinnan tarjoama valmis komponentti (Kuva 12.). Sen avulla voidaan synkronoida verkon yli Unityn peliobjektin sijainti ja kiertoliike. Peliobjektin tulee olla verkon yli luotava, jotta se voidaan synkronoida verkon yli. Lisäksi tämän peliobjektin on omattava NetworkIdentity-komponentti. (Unity 2018m.)



KUVA 10. Network Transform komponentti.

Pelaajaobjekti on pelissä objekti, joka merkattu pelaajan omaksi. Täten pelkästään sitä hallinnoiva pelaaja voi suorittaa komentoja tälle objektille. Ainoastaan pelaajaobjektissa olevista skripteistä voi lähettää komentoja, joilla kommunikoida palvelimen kanssa. (Unity 2018n.)

SyncVar attribuutin avulla voidaan synkronoida muuttujia palvelimelta muille päätelaitteille (Kuva 13.). Sallittuja muuttujia ovat C#-kielessä käytetyt perusmuuttujat, Unityn omat matemaattiset muuttujat sekä tietueet, jotka sisältävät näitä sallittuja muuttujia. Muuttujat, jotka ovat merkattu [SyncVar] attribuutilla lähetetään seuraavan ruudun (frame) aikana palvelimelta muille päätteille. (Unity 2018o.)

```
[SyncVar]
int throwCount;
```

KUVA 13. Esimerkki SyncVar muuttujasta.

4.3 Etätoiminnot (Remote Actions)

Verkkojärjestelmän yli lähetettäviä toimintoja kutsutaan yleensä etäproseduurikutsuiksi (RPC, Remote Procedure Calls). Unityssä on kahdenlaista etätoimintoa, jotka ovat Command ja ClientRPC. Command-komennot lähetetään päätteissä sijaitsevasta pelaajaobjektista palvelimelle palvelimen pelaajaobjektiin. ClientRPC-komento taas lähetetään palvelimen pelaajaobjektista päätteen pelaajaobjektiin. (Unity 2018h.)

Käyttääkseen Command-komentoa pitää funktion nimi alkaa Cmd-kirjaimilla ja olla merkittynä Command-attribuutilla. Käyttääkseen taas ClientRPC-komentoa on funktion alettava Rpc-kirjaimilla ja merkitty ClientRPC-attribuutilla. (Kuva 14.) (Unity 2018h.)

```
[ClientRpc]
public void RpcEsimerkkiFunktio()
{
    //Kun palvelimella kutsutaan tätä funktiota, niin tämä suoritetaan kaikissa päätteissä.
}
[Command]
public void CmdEsimerkkiFunktio()
{
    //Kun päätteessä kutsutaan tätä funktiota, niin tämä suoritetaan palvelimella.
}
```

KUVA 14. Esimerkki ClientRPC ja Command funktioista.

5 VERKKOPELIN TOTEUTUS

Päädyttiin toteuttamaan moninpeli Unityn omalla Unet -ratkaisulla. Valintaan päädyttiin Unetin helppokäyttöisyyden ja hinnan vuoksi. Lisäksi Unet on integroitu suoraan Unity-pelimoottoriin, joten ulkopuolisia palveluita ei tarvita. Toteutuksessa käytetään Unity HLAPI-rajapintaa, josta löytyy valmiiksi yksinkertaisen verkkopelin toteutuksen kannalta tarpeelliset toiminnallisuudet.

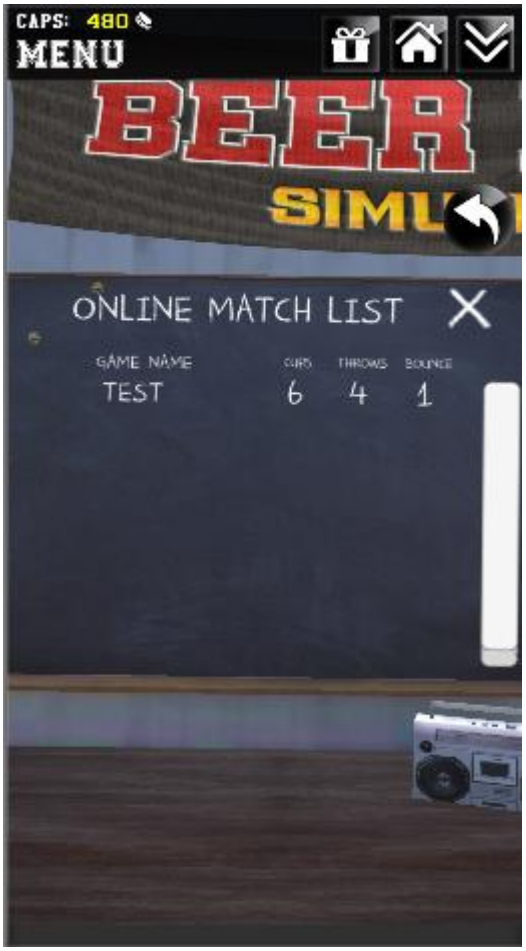
Toteutetun verkkopelin ideana on heittää vuorotellen palloa ja osua vastapuolen kuppeihin. Palloa heitetään säännöissä määrätyn verran vuorossa, jonka jälkeen vuoro vaihtuu. Pelin alussa laaditaan pelille säännöt (Kuva 15.). Pelin sääntöinä toimivat heittojen määrä vuorossa, osuttavien kuppien lukumäärä ja pompun kautta heitettäessä saatava bonus. Kumpi pelaajista osuu vastapuolen kaikkiin mukeihin ensimmäisenä, voittaa pelin.

KUVA 15. Pelin sääntöjen valinta.

Pallon heittäminen toimii pyyhkäisemällä ruutua ja pallo lentää pyyhkäisyyn suuntaan. Heiton voimakkuuteen vaikuttaa pyyhkäisyn pituus. Käyttäjän kosketusten käsittelyyn tehtiin Touch-niminen skripti. Pallo sisältää Unityn oman Rigidbody komponentin, jolla voidaan simuloida fysiikoita. Tämä mahdollistaa pallolle realistisen lentokaaren ja pomppimisen.

Pallon heittämistä varten ei tarvita kuin kaksi kolmiulotteista vektoria. Ensimmäinen tarvittava vektori on pallon sijainti heitettäessä. Tämä saadaan, kun pelaajan pyyhkäistessä ruutua otetaan viimeinen kosketuksen sijainti ruudulta ja muutetaan se peliympäristön sijainniksi. Toinen tarvittava vektori on voima, jolla palloa heitetään. Tämä saadaan, kun otetaan pyyhkäisyn ensimmäinen ja viimeinen sijainti ruudulta.

Pelissä se, joka laatii pelin säännöt ja aloittaa pelin, toimii pelin isäntänä (Host). Tällöin isäntälaite toimii pelissä palvelimena. Kun isäntä luo ottelun matchmaking-palvelu hoitaa pelin luonnin ja peli tulee muille nähtäväksi (Kuva 16.).



KUVA 16. Luodut pelit listattuna.

Matchmaking-palvelun käyttämiseksi tehtiin sille oma skripti CustomNetworkManger (Liite 1.). CustomNetworkManager-skritpin avulla voidaan luoda ja liittyä peliin, listata kaikki luodut pelit ja tietää, jos pelaajalta katkeaa yhteys.

Liityttävien pelien listaamiseksi tehtiin MatchListCanvas niminen skripti. MatchListCanvas saa CustomNetworkManager-skriptiltä listan avoimista peleistä ja listaa ne näkyville pelissä olevalle kanvakselle (Kuva 16.). Kun haluttua peliä napautetaan, kutsutaan CustomNetworkManagerista JoinGame-funktiota, jossa parametrinä liityttävän pelin tiedot. JoinGame-funktiossa matchMaker-palvelu huolehtii peliin liittymisestä.

Datan lähettämistä varten tehtiin oma skripti nimeltään OnlineGameManager (Liite 2.), joka lähettää dataa isännän ja päätteen välillä. OnlineGameManager on liitetty pelaajaobjektiin, joka luodaan CustomNetworkMangerin toimesta, kun peliin liitytään. Kun peliin on liitytty ja pelaajaobjekti luotu lähettää pääte komennon isäntälaitteelle. Kun isäntä saa tiedon pelaajan liittymisestä peliin, se lähettää päätteelle pelin säännöt ja peli alkaa.

Kun pelissä heitetään palloa, kutsutaan OnlineGameManager-skriptistä NetworkThrow-funktiota (Kuva 17.), jossa parametreinä on heiton alkusijainti- ja voimavektorit. Tässä funktiossa katsotaan, onko pelaaja isäntä vai pääte ja lähetetään tieto eteenpäin sen mukaisesti.

```
public void NetworkThrow(Vector3 throwPos, Vector3 force)
{
    if (isServer)
    {
        RpcSpawnObject(throwPos, force);
    }
    else if (!isServer && isClient)
    {
        CmdSpawnObject(throwPos, force);
    }
}
```

KUVA 17. NetworkThrow funktio.

Kun data saapuu määränpäähän, se luo pallon pelaajan laitteessa oikeaan kohtaan ja lisää sen Rigidbody komponenttiin voimaa voimavektorin verran. Tällöin pallo saadaan lentämään voiman osoittamaan suuntaan.

Pallon heittämisen olisi voinut toteuttaa myös liittämällä palloon NetworkTransform-komponentti. Tällöin, kun pelaajan laite luo pallo-objektin NetworkServer.Spawn-funktiolla, se luodaan myös verkon yli kaikille peliin yhdistäneille päätteille. NetworkTransform päivittää verkon yli pallon sijaintia ja pyörimisliikettä. Tämä kuitenkin lisäisi verkon yli lähetettävän datan määrää, joten päätettiin lähettää ainoastaan heittämisen kannalta tärkeä data ja simuloida heittoa toisessa laitteessa.

Peli toteutettiin siten, että molempien laitteet hoitavat erikseen pelin toimintaan liittyvän logiikan käsittelyn. Pelin sääntöjen ja pelin aikaisen logiikan käsittelyyn tehtiin OnlineGame niminen skripti. Tämä skripti pitää huolen heittojen laskemisesta, vuoron vaihdosta, kuppien määrän tarkastelusta ja pelin sääntöjen alustuksesta.

Toinen vaihtoehto olisi hoitaa kaikki otteluun vaikuttava pelilogiikka isäntälaitteella. Silloin isäntä kertoisi vastapuolelle, mitä tehdä ja missä tilanteessa peli on. Kuitenkin kaikki pelilogiikka oli jo valmiiksi kirjoitettu toimimaan jokaisella laitteella erikseen, joten päädyttiin toteuttamaan peli tällä tavalla.

Kun pelaaja saa osuttua kaikkiin kuppeihin, kysytään pelaajilta uusintapeliä. Jos jompikumpi pelaajista valitsee kielteisen vaihtoehdon, tuhoaa mathchmaker-palvelu pelin ja yhteys katkeaa. Tämä tapahtuu myös, jos toiselta pelaajalta katkeaa internetyhteys tai hän lopettaa pelin.

6 YHTEENVETO

Työn tavoitteena oli tutustua verkkopelin toimintaan ja toteutukseen ja toteuttaa verkon yli pelattava moninpeli. Tavoitteeseen päästiin, ja työssä käydäänkin läpi verkkopelin toimintaperiaatteet ja tekniikat. Lisäksi verkkopelin toteutus onnistui tavoitellusti ja lopputuloksena on toimiva verkon yli pelattava mobiilipeli (Kuva 18.).

Työssä verkkopelin toteutukseen valittiin käytettäväksi Unity Networking HLAPI-rajapinta, mikä ei välttämättä ollut paras valinta. Jos nyt pitäsi uudestaan valita, millä verkkopeli toteutettaisiin, päädyttäisiin Photonin PUN ratkaisuun. Tämä siksi, koska Unityn HLAPI-rajapinta vanhenee vuoteen 2022 mennessä, jolloin pelin verkkokoodi joudutaan vaihtamaan uuteen ympäristöön. Lisäksi PUN on monipuolisempi ja sen ylläpitäminen tulisi todennäköisesti myös halvemmaksi.

Kaikin puolin projekti oli hyvin opettavainen. Projektista saatiin kattava kuva verkon toiminnasta ja verkkopelien toteutuksesta. Lisäksi saatiin toimiva verkkopeli, joka on lähes valmis julkaistavaksi. Tämän opinnäytetyön opeilla on hyvä lähteä seuraavaa verkkopeliprojektia toteuttaamaan.

KUVA 18. Kuva pelitilanteesta.

LÄHTEET JA TUOTETUT AINEISTOT

Computer Hope 2017. Multiplayer [Verkkoaineoist] [Viitattu 2018-11-21] Saatavissa: https://www.computerhope.com/jargon/m/multiplay.htm

Unity 2018a. [Verkkoainesto] [Viitattu 2018-11-19] Saatavissa: https://unity3d.com/unity

Unity 2018b. [Verkkoaineisto] [Viitattu 2018-11-19] Saatavissa: https://store.unity.com/

Unity 2018c. AssetStore [Verkkoainesto] [Viitattu 2018-11-20] Saatavissa: https://www.assetstore.unity3d.com/en/

Unity 2018e. NetworkManager [Verkkoainoisto] [Viitattu 2018-11-20] Saatavissa: https://docs.unity3d.com/ScriptReference/Networking.NetworkManager.html

Unity 2018d. Scripting [Verkkoaineisto] [Viitattu 2018-11-21] Saatavissa: https://docs.unity3d.com/Manual/ScriptingSection.html

Unity 2018f. HLAPI [Verkkoaineisto] [Viitattu 2018-11-26] Saatavissa: https://docs.unity3d.com/Manual/UNetUsingHLAPI.html

Unity 2018g. Network Identity [Verkkoainoiso] [Viitattu 2018-11-26] Saatavissa: https://docs.unity3d.com/Manual/class-NetworkIdentity.html

Unity 2018h. Remote Actions [Verkkoainoist] [Viitattu 2018-11-26] Saatavissa: https://docs.unity3d.com/Manual/UNetActions.html

Unity 2018i. UNET deprecation [Verkkoainoisto] [Viitattu 2018-12-6] Saatavissa: https://support.unity3d.com/hc/en-us/articles/360001252086-UNet-Deprecation-FAQ

Unity 2018j Multiplayer [Verkkoaineisto] [Viitattu 2018-12-6] Saatavissa: https://unity3d.com/unity/features/multiplayer

Unity 2018k Transport layer API [Verkkoaineisto] [Viitattu 2018-12-9] Saatavissa: https://docs.unity3d.com/Manual/UNetUsingTransport.html

Unity 2018l NetworkBehaviour [Verkkoaineisto] [Viitattu 2018-12-9] Saatavissa: https://docs.unity3d.com/ScriptReference/Networking.NetworkBehaviour.html

Unity 2018m NetworkTransform [Verkkoaineisto] [Viitattu 2018-12-9] Saatavissa: https://docs.unity3d.com/Manual/class-NetworkTransform.html

Unity 2018n Player gameobject [Verkkoaineisto] [Viitattu 2018-12-9] Saatavissa: https://docs.unity3d.com/Manual/UNetPlayers.html

Unity 2018o SyncVar [Verkkoaineisto] [Viitattu 2018-12-9] Saatavissa: https://docs.unity3d.com/ScriptReference/Networking.SyncVarAttribute.html

Unity 2018p. Matchmaking [Verkkoainoisto] [Viitattu 2018-11-26] Saatavissa: https://docs.unity3d.com/520/Documentation/Manual/UNetMatchMaker.html

Chris 2017. [Verkkoaineisto][Viitattu 2018-12-7] Saatavissa: https://www.pcgamer.com/netcode-explained/

Long, James 2016. [Verkkoaineisto][Viitattu 2018-12-9] Saatavissa: https://jlongster.com/Making-Web-Games--5--Authoritative-Server

UDP explained 2018. [Verkkoaineisto][Viitattu 2018-12-9] Saatavissa: https://study-ccna.com/udp-explained/

CS557: Basic TCP Mechanisms. [Verkkoaineisto] [Viitattu: 2018-12-4] Saatavissa: http://www.cs.colostate.edu/~christos/cs557s12/Slides/05TCP.pdf

Unuth, Nadeem 2018. TCP (Transmission Control Protocol) Explained [Verkkoaineisto][Viitattu 2018-11-23] Saatavissa: https://www.lifewire.com/tcp-transmission-control-protocol-3426736

Bevilacqua, Bernardo 2013. Building a Peer-to-Peer Multiplayer Networked Game [Verkkoaineisto] [Viitattu 2018-11-22] Saatavissa: https://gamedevelopment.tutsplus.com/tutorials/building-a-peer-to-peer-multiplayer-networked-game--gamedev-10074

Ólafsson, Lárus 2015. Open sourcing the high level multiplayer networking layer [Verkkoaineisto] [Viitattu 2018-11-22] Saatavissa: https://blogs.unity3d.com/2015/11/19/high-level-multiplayer-networking-layer-has-been-open-sourced/

Photon 2018a. PUN [Verkkoainesto] [Viitattu 2018-12-6] Saatavissa: https://www.photonengine.com/en-US/PUN

Photon 2018b. [Verkkoainesto] [Viitattu 2018-12-6] Saatavissa: https://www.photonengine.com/en-US/Photon

Photon 2018c. Bolt [Verkkoainesto] [Viitattu 2018-12-6] Saatavissa: https://www.photonengine.com/en-US/BOLT

Photon 2018d. Bolt pricing [Verkkoainesto] [Viitattu 2018-12-6] Saatavissa: https://www.photonengine.com/en-US/BOLT/pricing

Forge 2018. User Manual [Verkkoaineisto] [Viitattu 2018-12-8] Saatavissa: http://docs.forgepowered.com/

Rouse, Margaret 2018. Internet Protocol [Verkkoaineisto] [Viitattu 2018-11-20] Saatavissa: https://searchunifiedcommunications.techtarget.com/definition/Internet-Protocol

Fiedler, Glenn 2008. UDP vs. TCP [verkkoaineisto]. [Viitattu 2018-11-19] Saatavissa: https://gafferongames.com/post/udp_vs_tcp/

Karim, Muzahidul 2018. How Online Game Works [Verkkoaineisto]. [Viitattu 2018-11-19] Saatavissa: https://viblo.asia/p/how-online-game-works-JQVGVBPMGyd

Duffy, Greg 2018. What is network latency (and how do you use a latency calculator to calculate throughput)? [Verkkoaineisto]. [Viitattu 2018-11-19] Saatavissa: https://www.sas.co.uk/blog/what-is-network-latency-how-do-you-use-a-latency-calculator-to-calculate-throughput

Shaw, Keith 2018. The OSI model explained: How to understand (and remember) the 7 layer network model. [Verkkoaineisto]. [Viitattu 2018-11-19] Saatavissa: https://www.networkworld.com/article/3239677/lan-wan/the-osi-model-explained-how-to-understand-and-remember-the-7-layer-network-model.html

LIITE 1. CustomNetworkManager.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Networking;
using UnityEngine.Networking.Types;
using UnityEngine.Networking.Match;

public class CustomNetworkManager : NetworkManager {

    public ulong netId;
    Timeri t;
             // Use this for initialization
             public void StartHosting () {
        StartMatchMaker();
        matchMaker.CreateMatch(CreateGameCanvas.Rules.name1+"-"+OnlineGame.rules.throws+"-
"+OnlineGame.rules.cups+"-"+OnlineGame.rules.bounceThrow, 2, true, "", "", "", 0, 0, OnMatchCre-
ated);
             }
    private void Start()
    {
        t = new Timeri();
    }
    void Update()
    {
        if (matchMaker != null && t.IsTime(1f))
            RefreshMatches();
    }
    public override void OnServerDisconnect (NetworkConnection conn)
    {
        base.OnServerDisconnect (conn);
        Debugger.Log("Server disconnected from match");
            OnlineGame.instance.MatchCancelled();
    }
    public override void OnClientDisconnect (NetworkConnection conn)
    {
        base.OnClientDisconnect (conn);
        Debugger.Log("Client disconnected from match");
            OnlineGame.instance.MatchCancelled();
    }
    private void OnMatchCreated(bool success, string extendedInfo, MatchInfo responseData)
    {
        StartHost(responseData);
        netId = (ulong)responseData.networkId;
        //Debugger.Log("Match created and host started");
    }
    public void RefreshMatches()
    {
        if(matchMaker == null)
        {
            StartMatchMaker();
        }
        matchMaker.ListMatches(0, 10, "", true, 0, 0, HandleListMatches);
    }
    public void CancelMatch()
    {
        if(matchMaker != null)
        {
            matchMaker.DestroyMatch((NetworkID)netId, 0, OnMatchDestroy);
        }
    }

    public void OnMatchDestroy(bool success, string extendedInfo)
    {
        Debugger.Log("Match cancelled!");
```

```
    }
    public void JoinGame(MatchInfoSnapshot match)
    {
        if (matchMaker == null)
            StartMatchMaker();
        matchMaker.JoinMatch(match.networkId, "", "", "", 0, 0, HandleJoinedMatch);

    }


     void HandleJoinedMatch(bool success, string extendedInfo, MatchInfo responseData)
    {
        StartClient(responseData);
        matchMaker.SetMatchAttributes(responseData.networkId, false, 0, (sucess, extndedInfo) =>
{
            if (!success) Debugger.Log("Unable to update match listed: " + extndedInfo);
        });
    }

    private void HandleListMatches(bool success, string extendedInfo, List<MatchInfoSnapshot> re-
sponseData)
    {
        ListMatches.SetMatchData(responseData);
        GetComponent<MatchListCanvas>().ListFoundMatches();
    }
    public void OnConnected(NetworkMessage netMsg)
    {
        //Debug.Log("Connected to server");
    }
}
```

LIITE 2. OnlineGameManager.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Networking;

public class OnlineGameManager : NetworkBehaviour {
    public GameObject Ball;
    private GameObject ball;
    public static OnlineGameManager instance;

    public int rematchCount = 0;

    // Use this for initialization
    void Start () {
                        if(instance != null)
        {
            Destroy(instance.gameObject);
        }
        instance = this;

        //Debugger.Log("Online Game Manager Creted");
        //Send info to host that player has joined match
        if (!isServer)
            CmdClientJoined();
    }
    private void OnPlayerConnected(NetworkPlayer player)
    {
       // Debugger.Log("player connected");
    }
    [Command]
    public void CmdRematch()
    {
        rematchCount++;
        if(rematchCount == 2)
        {
            rematchCount = 0;
            StartOnlineGame();
        }
    }
    public void Disconnect()
    {
        if (isServer)
            NetworkManager.singleton.StopHost();
        else
            NetworkManager.singleton.StopClient();
    }
    [Command]
    public void CmdCancelMatch()
    {
        RpcCancelMatch();
    }
    [ClientRpc]
    public void RpcCancelMatch()
    {
        Network.CloseConnection(Network.connections[0], false);
    }
    private void StartOnlineGame()
    {
        // initalize game and call client with right values
        OnlineGame.instance.StartGame(false);

        RpcHostReadyForGame(OnlineGame.rules);
    }
    [Command]
    public void CmdClientJoined()
```

```
    {
        StartOnlineGame();
    }
    [Command]
    public void CmdSpawnObject(Vector3 throwPos, Vector3 force)
    {

        ball = Instantiate(Ball);
        ball.transform.position = throwPos;
        ball.GetComponent<Ball>().isThrown = true;
        ball.GetComponent<Rigidbody>().AddForce(force);
        //NetworkServer.Spawn(ball);
        //if(isClient)
        //    Destroy(ball);
    }
    [ClientRpc]
    public void RpcHostReadyForGame(OnlineGame.Rules rules)
    {
        if (!isServer)
        {
            OnlineGame.instance.StartGame(true, rules);
        }
    }
    [ClientRpc]
    public void RpcSpawnObject(Vector3 throwPos, Vector3 force)
    {
        ball = Instantiate(Ball);

        ball.transform.position = throwPos;
        ball.GetComponent<Ball>().isThrown = true;
        ball.GetComponent<Rigidbody>().AddForce(force);
    }
    [ClientRpc]
    public void NetworkThrow(Vector3 throwPos, Vector3 force)
    {
        if (isServer)
        {
            RpcSpawnObject(throwPos, force);
        }
        else if (!isServer && isClient)
        {
            CmdSpawnObject(throwPos, force);
        }
    }
     private void OnDisconnectedFromServer(NetworkDisconnection info)
    {
        Debugger.Log("Disconnected " + info);
    }

}
```