Opinnäytetyö

Tietojenkäsittely

2019

Billy Ward

MOBIILIPELIN MUUTTAMINEN AR-MONINPELIKSI

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tietojenkäsittely

2019 | 41 sivua

Billy Ward

MOBIILIPELIN MUUTTAMINEN AR-MONINPELIKSI

Lisätyllä todellisuudella (engl. augmented reality, AR) tarkoitetaan oikean maailman ympäristön vuorovaikutuksen tehostamista tuomalla siihen digitaalista sisältöä.. Mobiililaitteiden soveltuvuus teknologialle on tehnyt siitä laajasti käytetyn työkalun mobiilisovelluksissa, ja vastaavasti mobiilipeleissä. Opinnäytetyö keskittyy AR-moninpelisovelluksen kehittämiseen, jota voidaan esittää mahdollisille asiakkaille, ja käyttää pohjana tulevaisuuden mobiilisovelluksille.

Opinnäytetyö pyrkii vastaamaan seuraavaan kysymykseen: kuinka mobiilipeli muutetaan lisätyn todellisuuden moninpeliksi?

Peli kehitetään hyödyntämällä jo olemassa olevaa prototyyppiä, joka on luotu Unity-pelimoottorilla. Opinnäytetyön reunaehtoja ovat, että moninpeliominaisuudet kehitetään Unityn UNet-työkaluilla ja että lisätyn todellisuuden ominaisuudet kehitetään Googlen ARCore-teknologialla. Tavoitteen saavuttamiseksi työssä tutustutaan tarvittaviin teknologiakomponentteihin, niiden käyttöön ja muutosprosessin vaiheisiin. Työn tuloksena pelin prototyyppi kehitettiin tukemaan moninpeliä kahden pelaajan välillä. Pelaajat pystyvät luomaan ja liittymään peleihin internetin välityksellä ja pelaamaan toisiaan vastaan oikean maailman ympäristössä.

Lukujen kaksi, kolme ja neljä lopputuloksina rajattiin välttämättömät elementit ja työkalut, jotka kehittäjän täytyy hallita tällaisen projektin suorittamiseksi. Niiden käyttötarkoitukset ja ohjeet niiden käytölle selostettiin. Opinnäytetyö myös korostaa niiden käytössä ongelmakohtia, joita voi syntyä, mikäli niistä ei ole tietoinen. Luvussa viisi kuvaillaan vaiheet muutostyön tekemiseksi em. komponenttien avulla: komponenttien asentaminen, skriptien päivittäminen, AR-alustan integroiminen ja Unity Multiplayer -palvelun käyttöönotto. Tämän opinnäytetyön tapauksessa askeleet toteutettiin toimintapeliin, mutta askeleet ovat yleistettävissä muihin pelilajeihin.

Unetin laadukkaan dokumentaation puute ja ARCoren uutuus tekevät tehokkaan työskentelemisen hankalaksi teknologioilla. Pelin jatkokehitysmahdollisuuksia ovat sen laajentaminen tukemaan enemmän pelaajia peli-istunnossa, ja merkityksen lisääminen pelaajahahmon muokkaamiseen varustetilastojen avulla.

ASIASANAT:

Mobiilipeli, Moninpeli, AR, ARCore, Unity

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Bachelors degree of information technologies

2019 | 41 pages

Billy Ward

CONVERTING A MOBILE GAME INTO AR-MULTIPLAYER

Augmented Reality (AR)'s popularity as a technology is on the rise. It provides the capability of bringing digital information into the user's environment, and thus has been able to ease working and deliver entertainment in a new manner. The suitability of mobile devices for the technology has made AR a prominent tool in mobile applications, and in mobile games correspondingly. This thesis focuses on the development of an AR-multiplayer mobile game, which can be demonstrated for potential clients and used as a foundation for future applications.

The objective of this thesis was to investigate how to convert an augmented reality mobile game into multiplayer.

The game was developed using an existing prototype created with the Unity engine. The requirements in the project were that the multiplayer features would be developed with Unity's UNet tools, and the augmented reality features with Google's ARCore technology. As a result, the prototype was developed to support competitive play between two players. Players can host and join games through the internet and play against each other in the surrounding real-world environment.

As a result of chapters two, three and four, the necessary elements and tools that are required for a project like this were outlined. Their purposes and instructions for their usage were explained. This thesis also highlights problems that might arise if one is not aware of some limitations. In chapter five the necessary steps for implementing a project like this were described: setting up the components, updating the scripts and integrating the AR-platform. In the case of this thesis, the steps were executed on an action game but can be generalized to other genres.

The lack of quality documentation with UNet and the new nature of ARCore makes it difficult to have an effective start with the technologies. Further development of the game could include expanding it to support more players in a session and enhance player character customization by adding equipment statistics.

KEYWORDS:

Mobile game, Multiplayer, AR, ARCore, Unity

SISÄLTÖ

KUVAT

KOODIESIMERKIT

KÄYTETYT LYHENTEET

SDK	Ohjelmistokehityspaketti (engl. software development kit), joka tarjoaa kokoelman työkaluja ja kirjastoja ohjelmistokehitykseen (Sandoval, 2016).
APK	Android Package Kit, pakettitiedostoformaatti, jolla jaetaan ja asennetaan mobiilisovelluksia Android-laitteille (Montegriffo, 2019).
UI	Käyttöliittymä (engl. user interface) on kaikki elementit digitaalisessa tuotteessa, joiden kanssa käyttäjä voi olla vuorovaikutuksessa (UserTesting, 2018).

1 JOHDANTO

Lisätyllä todellisuudella (engl. augmented reality, AR) tarkoitetaan oikean maailman ympäristön vuorovaikutuksen tehostamista tuomalla siihen digitaalista sisältöä. Sitä käytetään monissa erilaisissa elämäntilanteissa, kuten peleissä ja viihdeteollisuudessa. Kouluissa sillä tehostetaan opettamista. Työtilanteissa asentajat ja lääkärit kykenevät nopeuttamaan ja helpottamaan työtehtäviään sen avulla. Terveysalalla lisätty todellisuus on sallinut potilaiden kehojen ja elimien tarkkailun samalla tavalla kuin röntgenkuvaus, mutta koko värispektrillä (Kipper & Rampolla, 2012). Pelimaailmassa Nianticin kehittämä Pokemon GO on esimerkki lisättyä todellisuutta hyödyntävästä sovelluksesta, joka oli yksi suosituimmista peleistä sen julkaisun jälkeen heinäkuussa 2016. Google Play (2018) listaa vielä tänä päivänä Pokemon GO:n huippupelinä bruttotulojen perusteella sen pelikategoriassa, ja vuoden 2018 syyskuuhun mennessä sovellus oli ylittänyt kahden biljoonan myyntitulot (Blacker, 2018). CCS Insight (2018) arvioi VR- ja AR-laitteiden markkina-arvon olevan noin 1,8 miljardia dollaria vuonna 2018. ARPost (2018) listasi vuonna 2018 AR-pelien kulutuksen kolmanneksi suurimmaksi osa-alueeksi AR- ja VR-kulutuksessa 5,1 %:n osuudella.

Moninpelit ovat tällä hetkellä hyvin suosittuja. Pelatuimmista peleistä lähes kaikki sisältävät tai pohjautuvat kokonaan moninpelikomponenttiin (Statista, 2018). Pelaajien toiminnot ovat odottamattomia, mikä pitää pelin mielenkiintoisena. Joissakin peleissä pelaajia yritetään jäljitellä tietokonevastustajilla, mutta ihmisen tasoisen vastustajan luominen ei ole helppoa. Ihmisvastustajat sallivat lähes loputtomia tapoja pelin tapahtumien etenemiselle. Tämän vuoksi pelaajien on mahdollista pelata samoja tasoja peleissä kymmeniä ellei jopa satoja kertoja kyllästymättä niihin. Lisätty todellisuus ja moninpeli eivät ole myöskään tyypillinen yhdistelmä. Pokemon GO on AR-moninpeli, mutta moninpeli pohjautuu vahvasti sosiaalisiin elementteihin. Sen vastakohta olisivat ns. toiminnalliset moninpelit, joissa pelaajat ovat vuorovaikutuksessa toistensa kanssa reaaliajassa.

Työn tavoitteena on kehitettävän mobiilipelin prototyypin muuntaminen lisätyn todellisuuden toiminnalliseksi moninpeliksi ARCorella ja Unity Networkingilla (UNetilla). Tavoitteen saavuttamiseksi työssä tutustuttaan projektissa tarvittaviin komponentteihin, niiden käyttöön ja muutosprosessin vaiheisiin. ARCore on yksi uusimmista lisätyn todellisuuden sallivista tekniikoista, jonka Google on kehittänyt. Google demosi ARCoren

syksyllä 2017 ja julkaisi ensimmäisen vakaan version 1. maaliskuuta 2018. Koska ARCore on uusi tekniikka, sillä ei ole vielä kehitetty kokonaisia sovelluksia. Opinnäytetyön tutkimusmenetelmä on konstruktiivinen.

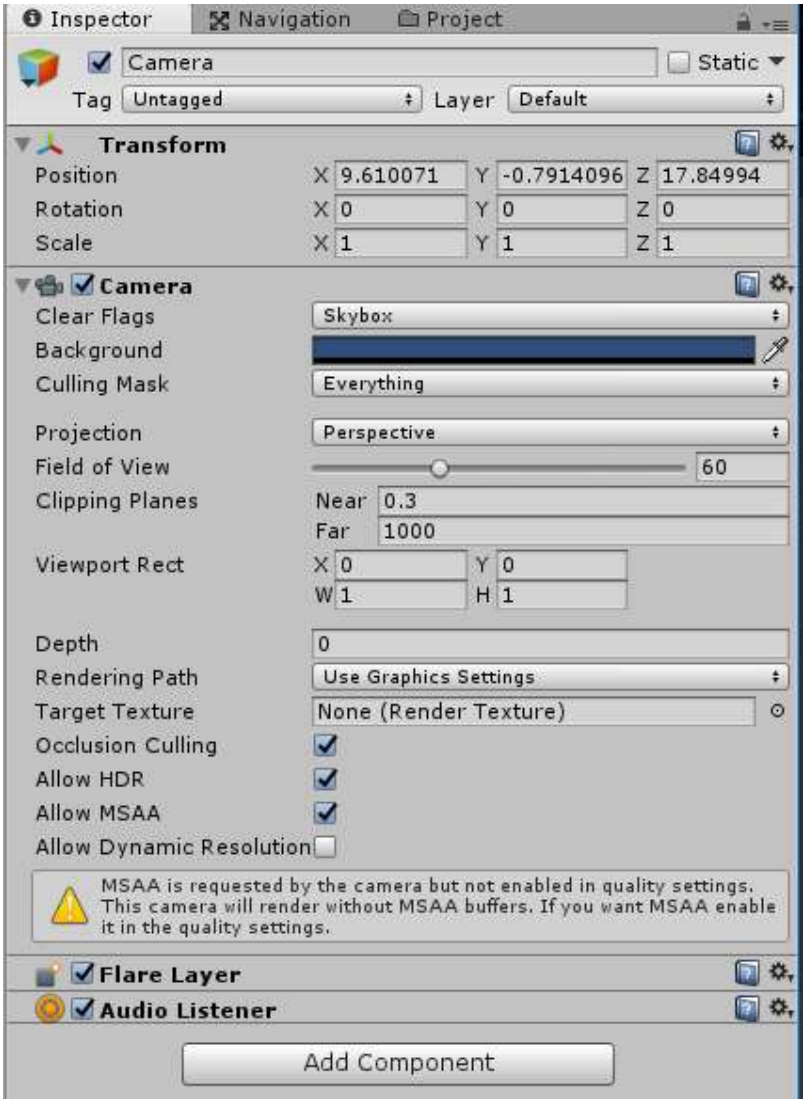
Opinnäytetyön toisessa luvussa keskityään Unityn keskeisiin elementteihin. Kolmannessa luvussa tutkitaan UNetia ja selostetaan sen komponenttien käyttötarkoituksia. Neljännessä luvussa selvitetään Unity ARCore SDK:n sisältöä ja sen projektissa käytettäviä osasia. Viidennessä luvussa käydään läpi askeleet prototyyppin muuttamiseksi lisätyn todellisuuden moninpeliksi. Muuttaminen sisältää skriptien päivittämisen, UNet-komponenttien asettelemisen, ARCoren yhteensovittamisen projektiin ja Unity Multiplayer -palvelun käyttöönoton. Kuudennessa luvussa keskitytään projektissa tapahtuneeseen kehitykseen, jatkokehitysmahdollisuuksiin, ja pohditaan opinnäytetön onnistumista.

2 UNITY-PELIMOOTTORI

2.1 Peliobjekti

Peliobjekti (engl. gameobject) on tärkein elementti Unity-pelimoottorissa. Jokainen esine tai asia pelissä/sovelluksessa on peliobjekti. Itsekseen peliobjekti ei kykene tekemään paljon mitään. Liittämällä komponentteja ja skriptejä peliobjektiin voidaan määritellä sen käyttötarkoitus: pelattava hahmo, ympäristö, erikoistehoste, ym. Peliobjekti voidaan kuvitella eräänlaisena kotelona, jolle voidaan määritellä ominaisuuksia. (Unity Technologies, 2019a.)

Peliobjekti sisältää aina Transform-komponentin, jota ei pysty poistamaan. Unity sisältää monia esivalmistettuja peliobjekteja, joita voidaan luoda pelimoottorin GameObject-valikon kautta. (Unity Technologies, 2019b.) Esivalmistetut peliobjektit sisältävät entuudestaan komponentteja niiden käyttötarkoitusten mukaan. Esimerkiksi valitsemalla listasta Camera-peliobjekti luodaan uusi peliobjekti, joka sisältää Camera-, Flare Layer- sekä Audio Listener -komponentit (Kuva 1).



Kuva 1. Camera-peliobjekti.

2.2 Komponentti

Komponentit (engl. component) tuovat peliobjekteihin toiminnallisuutta. Jokainen peliobjekti sisältää oletuksena Transform-komponentin, joka määrittelee peliobjektin sijainnin, rotaation ja skaalan pelimaailmassa. Transform-komponentti mahdollistaa myös ”vanhemmuus”-konseptin, jossa peliobjektit ovat olemassa suhteessa toisiinsa. (Unity Technologies, 2019c.)

Komponentteja pystyy liittämään peliobjekteihin komponenttivalikon tai komponenttiselaimen kautta. Komponenttiselain sisältää hakukentän, jonka avulla voi

hakea komponentteja nimen mukaan sadoista saatavilla olevista komponenteista. Peliobjektiin pystyy liittämään useita komponentteja, ja jotkin komponentit toimivatkin parhaiten yhdessä toistensa kanssa. Esimerkiksi Rigidbody- ja Collider-komponentit. Rigidbody sallii peliobjektissa fysiikat, jolloin peliobjektiin vaikuttaa painovoima sekä muut mahdolliset ympäristövaikutukset. Tällaisenaan peliobjekti ei kykene olemaan vuorovaikutuksessa ympäristönsä kanssa, se kulkisi maaston sekä muiden pelimaailman objektien läpi. Collider-komponentti sallii törmäykset muiden peliobjektien kanssa, jolloin peliobjekti voi olla olemassa maaston päällä ja vaikuttaa muihin peliobjekteihin. (Unity Technologies, 2019d.)

Komponentit ovat erittäin joustavia ja sisältävät monia muokattavia ominaisuuksia (engl. properties). Ominaisuudet voidaan jakaa kahteen päätyyppiin: arvoihin ja viittauksiin. Arvot ovat nimensä mukaisesti suoraan komponentissa muutettavia arvoja. Ne voivat olla numeerisia arvoja, nappuloita, pudotusvalikoita, liukusäätimiä, tekstin merkkijonoja, värejä ja monia muita tyyppejä. Viittaukset ovat kenttiä, jotka viittaavat jonkin tietyn tyyppiseen objektiin, kuten peliobjektiin tai komponenttiin. (Unity Technologies, 2019d.) Viittaus voidaan tehdä vetämällä haluttu peliobjekti tai komponentti viittauskenttään. Vaihtoehtoisesti kentän vierestä voidaan painaa pientä pistettä, joka aukaisee valikon projektista löytyvistä objekteista.

2.3 Prefab

Unityn prefab-järjestelmä sallii peliobjektin luomisen, muokkaamisen ja tallentamisen. Tallennettu prefab sisältää peliobjektin kaikki komponentit, komponenttien arvot, ja peliobjektin alapeliobjektit. Prefab toimii mallina, josta voi helposti luoda uusia peliobjektin ilmentymiä. (Unity Technologies, 2019e.) Prefab on ikään kuin esivalmistettu peliobjekti, mutta käyttäjän luoma.

Prefabien tarkoitus on tehdä peliobjekteista helposti uudelleenkäytettäviä. Mikäli peliobjekti tulee esiintymään pelissä moneen otteeseen, se kannattaa tallentaa prefabiksi. Tämä on parempi toteutus kuin peliobjektin kopiointi ja liittäminen, sillä prefabin päivittäminen päivittää myös sen kaikki ilmentymät. Prefabin jokaisen ilmentymän ei kuitenkaan tarvitse olla täysin samanlainen. Prefabin ilmentymää voi muokata editorissa tilanteen vaatimalla tavalla. Tyypillisiä käyttötarkoituksia prefabeille ovat ympäristölliset esineet, pelattavat ja ei-pelattavat (engl. non-player character, NPC) hahmot, ammukset ja erikoistehosteet. (Unity Technologies, 2019e.)

Prefabin voi luoda yksinkertaisesti vetämällä peliobjekti hiiren avulla hierarkianäkymästä projektinäkymään. Peliobjekti päivittyy automaattisesti prefab-ilmentymäksi. Uuden ilmentymän voi luoda vetämällä prefabin projektinäkymästä hierarkkianäkymään. Prefabien etuna on myös ilmentymien luominen suoritusaikana. Tämä on välttämätöntä ei-esiluotavien esineiden, kuten hahmojen ampumien ammuksien ja erikoistehosteiden, kannalta. (Unity Technologies, 2019e.) Ilmentymien luominen suoritusaikana täytyy tehdä skriptien avulla.

2.4 Scene

Scenet sisältävät ympäristöt, valikot ja kaikki peliobjektit, jotka halutaan esittää pelissä kerrallaan. Scene on siis kokoelma aseteltuja peliobjekteja. Peliobjektien kaltaisesti scene ei itsessään sisällä paljon toiminnallisuutta, vaan sen kaikki aktiviteetti tapahtuu sen sisältämien peliobjektien ansiosta. Yhtä sceneä voi tyypillisesti kohdella kuin yhtä tasoa pelissä. (Unity Technologies, 2019f.)

Pelille voidaan määrittää aloitus-scene, joka latautuu, kun peli käynnistetään. Pelin muiden scenejen käyttämistä varten täytyy hyödyntää SceneManager-luokkaa. Luokalla kyetään vaihtamaan pelin sceneä suorituksen aikana. Nykyisen scenen päälle pystytään myös lataamaan toinen scene, ja scenet on mahdollista yhdistää toisiinsa. Suorituksenaikaisella scenejen yhdistämisellä voidaan ladata lisää ympäristöä peliin katkaisematta pelinkulkua, joka mahdollistaa saumattoman pelikokemuksen luomisen.

3 UNITY NETWORKING

3.1 Komponentit

3.1.1 Network Manager

Network Manager on komponentti, jonka tehtävä on hallinnoida verkko-ominaisuuksia moninpelissä (Unity Technologies, 2018g). Tärkeimmät ominaisuudet verkkotoiminnoille määritellään tässä komponentissa, ja monet toimenpiteet suoritetaan sen kautta. Se sisältää monia muuttujia, jotka toimivat asetuksina peli-istunnoille. Niillä voidaan määritellä istunnon vähimmäis- ja enimmäislukemat pelaajamäärille. Pelaajien enimmäismäärä yhdelle yhteykselle voidaan myös määritellä. Muuttujilla asetetaan myös verkkoyhteyksien tekniset ominaisuudet, kuten IP-osoitteet ja käytettävät verkkoportit.

Komponentti on vastuussa kaikkien verkostoitujen peliobjektien luomisessa, joten ne kaikki rekisteröidään sen sisälle. Pelaajahahmon peliobjekti rekisteröidään erikseen, ja komponentissa voidaan määritellä, tuleeko sen luoda pelaajahahmot automaattisesti, sekä miten ne luodaan. Perimällä Network Managerin luokkaa on mahdollista mukauttaa sen asetuksia ja käyttäytymistä enemmän luomalla räätälöity Network Manager. Se sisältää koukkuja, joilla tiettyjä vaiheita suoritetuissa toimenpiteissä voidaan muokata koodissa (Koodiesimerkki 1).

Koodiesimerkki 1. Koukkua hyödyntämällä pelaajan yhteyden katkeamisesta voidaan kerätä mahdollista virhetietoa (Unity Technologies, 2018g).

```
public override void OnServerDisconnect(NetworkConnection conn) {

    NetworkServer.DestroyPlayersForConnection(conn);

    if (conn.lastError != NetworkError.Ok) {

        if (LogFilter.logError) { Debug.LogError("ServerDisconnected due to error: " + conn.lastError); }

    }

    Debug.Log("A client disconnected from the server: " + conn);

}
```

3.1.2 Network Lobby Manager

Network Lobby Manager -komponentti on Network Managerin erikoistunut muoto. Sen päätarkoitus on mahdollistaa aulan luominen moninpeliin, jossa pelaajat sijaitsevat ennen varsinaiseen peliin siirtymistä. Komponentti sisältää lisäasetuksia, joilla voidaan määritellä lobby scene, ja pelaajaobjekti, joka on ainutlaatuinen aulaympäristölle. (Unity Technologies, 2019h.) Aulassa käytettävän pelaajaobjektin täytyy sisältää Network Lobby Player -komponentti, joka on vastuussa pelaajan valmiustilasta sekä muista mahdollisista määrittelytiedoista. Komponentti ilmoittaa valmiustilamuutoksista Network Lobby Managerille. Kun kaikkien pelaajien valmiustila on ”valmis”, Network Lobby Manager siirtää pelaajat automaattisesti play sceneen (Unity Technologies, 2019h).

Network Lobby Manager sisältää lisäksi muita pieniä lisäominaisuuksia, joita ei löydy Network Managerista. Se mahdollistaa ”sohvamoninpelin” (engl. couch multiplayer), jossa monta pelaajaa pystyy liittymään peliin yhden sovelluksen ilmentymän kautta. Sen voi myös määritellä estämään pelaajien liittymisen keskeneräisiin peleihin. (Unity Technologies, 2019h.)

3.1.3 Network Identity

Network Identity on UNetin yleisimmässä käytössä oleva komponentti. Jokaisen verkostoidun peliobjektin täytyy sisältää tämä komponentti. Network Identity antaa peliobjektille ainutlaatuisen identiteetin, jolla verkkojärjestelmä tulee tietoiseksi peliobjektin läsnäololle (Unity Technologies, 2019i). Identiteetin avulla Network Manager tunnistaa saman peliobjektin jokaisen pelaajan sovelluksessa.

Ainoastaan palvelin pystyy luomaan peliobjekteja, jotka sisältävät Network Identity -komponentin. Verkostoituja peliobjekteja voi luoda kahdella menetelmällä. Peliobjektit voi tallentaa osaksi sceneä etukäteen, esim. ympäristölliset peliobjektit, joiden sijaintia täytyy päivittää, kuten liikkuva alusta. Pelaajan liittyessä peliin scene-pohjaiset peliobjektit laitetaan pois päältä ja palvelin kertoo sovellukselle peliobjektien viimeisimmät tiedot (Unity Technologies, 2019i). Toinen menetelmä on luoda verkostoidut peliobjektit suorituksenaikaisesti. Kaikki peliobjektit, jotka halutaan luoda näin, täytyy rekisteröidä Network Manageriin etukäteen. Peliobjekteista täytyy olla

olemassa prefabit ja niiden täytyy sisältää Network Identity -komponentit (Unity Technologies, 2019i).

Network Identity ei itse sisällä paljon muokattavia ominaisuuksia. Se sisältää kaksi boolean-tyyppistä nappulaa: ”Server Only” ja ”Local Player Authority”. Server Only -asetus määrittelee, mikäli peliobjekti luodaan ainoastaan palvelimen puolella. Local Player Authority -asetus päättää onko peliobjektin omistavalla pelaajalla hallinta kyseisestä peliobjektista. (Unity Technologies, 2019i.) Tyypillinen peliobjekti, jossa asetus on päällä, on pelaajaobjekti, sillä jokaisen pelaajan halutaan olevan hallussa omasta pelaajaobjektistaan.

Tärkeä asia ottaa huomioon komponenttia käyttäessä on, että peliobjekti voi sisältää ainoastaan yhden Network Identity -komponentin, ja sen täytyy sijaita peliobjektin juuressa. Peliobjektin alapeliobjektit eivät siis voi sisältää Network Identity -komponentteja.

3.1.4 Network Transform

Network Transform on komponentti, jonka tehtävä on synkronoida peliobjektien sijainti ja rotaatio pelaajien sovellusten välillä. Network Transform toimii ainoastaan peliobjekteissa, jotka omaavat Network Identity -komponentin. Mikäli komponentin kiinnittää peliobjektiin, joka ei omaa Network Identity -komponenttia, niin kyseinen komponentti lisätään samalla automaattisesti. Jotta Network Transform -komponentti toimisi oikein, verkostoidut peliobjektit täytyy luoda palvelimen kautta. (Unity Technologies, 2019j.)

Komponentti ottaa huomioon peliobjektin valtuuden, joten peliobjektit, jotka omaavat paikalliset valtuudet päivittävät niiden tiedot käyttäjältä palvelimelle, joka lähettää ne eteenpäin muille käyttäjille. Muut peliobjektit päivittävät niiden sijainnin palvelimelta käyttäjille. (Unity Technologies, 2019j.)

3.2 Skriptit

3.2.1 NetworkBehaviour

NetworkBehaviour-skriptit toimivat peliobjekteissa, jotka omaavat Network Identity -komponentin. Skriptit toimivat samantapaisesti kuin MonoBehaviour-skriptit, mutta tuovat skripteihin välttämättömiä ominaisuuksia, joita tarvitaan nettipeleissä. NetworkBehaviour ei ole komponentti, jonka voi kiinnittää suoraan peliobjektiin. (Unity Technologies, 2019k.) Yksinkertaisin tapa luoda NetworkBehaviour-skripti on luoda Unityn Editorissa projektinäkymässä uusi skriptitiedosto ja sitten muokata skripti perimään NetworkBehaviour-luokkaa MonoBehaviour-luokan sijasta.

NetworkBehaviour sallii uusien attribuuttien käyttämisen skripteissä: SyncVar, ClientRpc ja Command (Unity Technologies, 2019k). Attribuutteja käytetään merkkaamaan verkostoituja toimintoja ja ominaisuuksia. SyncVar-attribuutti sallii muuttujien synkronoimisen palvelimelta käyttäjille. Synkronointi ei toimi käyttäjältä palvelimen suuntaan. Synkronoitavat muuttujat voivat olla mitä tahansa perustyyppiä. (Unity Technologies, 2019k.)

NetworkBehaviour tuo skripteihin myös uusia muuttujia, joita voi käyttää tarkistuksiin ja kontrollin lisäämiseen koodissa. Tärkeimpiä muuttujia ovat ”isLocalPlayer”, ”isServer” ja ”isClient”, jotka ovat kaikki boolean-tyyppisiä muuttujia. (Unity Technologies, 2019k.) Niitä voi käyttää koodissa tarkistuksiin, joilla varmistetaan, että vain palvelin tai käyttäjä voi ajaa tiettyä koodia. Esimerkiksi pelaajaobjektin skriptit hyödyntävät tyypillisesti isLocalPlayer-muuttujaa, jotta ainoastaan paikallinen pelaaja voi hallita omaa pelaajaobjektiaan.

Skripteissä voidaan myös käyttää hyödyllisiä koukkuja tietyille tapahtumille. ”OnStartServer”- ja ”OnStartClient”-koukut ovat molemmat funktioita, jotka ajetaan, kun verkostoitu peliobjekti luodaan. OnStartServer ajetaan ainoastaan palvelimen puolella ja OnStartClient ajetaan käyttäjien puolella. On otettava huomioon, että asetelmissa, joissa yksi käyttäjä toimii myös palvelimena, molemmat koukut tullaan ajamaan samassa peliobjektissa. (Unity Technologies, 2019k.)

3.2.2 NetworkServer

NetworkServer on skripti, joka ylläpitää käyttäjäyhteyksiä ja lisää verkkotoimintoihin pelinkaltaisia ominaisuuksia (Unity Technologies, 2019l). Network Manager hyödyntää NetworkServeriä sen toimintoihin, ja tämän vuoksi NetworkServeriä ei yleensä käytetä suoraan. Se kuitenkin sisältää joitakin funktioita, joita voi ajaa ainoastaan sen kautta ja ovat välttämättömiä nettipeleissä. Tärkein funktio saattaa olla Spawn()-funktio, jolla palvelin luo uusia verkostoituja peliobjekteja. Skripti on kokonaisuudessaan staattinen (engl. static), joten sitä voi kutsua helposti koodissa eikä sitä tarvitse erikseen valmistella.

3.3 Etätoiminnot

Etätoiminnot (engl. remote actions) sallivat toimintojen suorittamisen verkon välityksellä. Etätoimintoja on kahta eri tyyppiä: ClientRpc ja Command (Unity Technologies, 2019m). Etätoimintoja hyödynnetään NetworkBehaviour skripteissä vastaavilla ClientRpc- ja Command-attribuuteilla.

Etätoimintojen funktiot eivät hyväksy parametreina minkä tahansa tyyppisiä muuttujia. Hyväksyttyjen tyyppien lista on kuitenkin laaja: perustyypit, perustyyppien taulukot, Unityn sisäänrakennetut matemaattiset tyypit (Vector3, Quaternion, jne.), Network Identity -komponentit ja peliobjektit, jotka omaavat Network Identity -komponentin. (Unity Technologies, 2019m.)

3.3.1 ClientRpc

ClientRpc-etätoiminnot toimivat palvelimen puolelta käyttäjien suuntaan. Mikä tahansa peliobjekti, joka omaa Network Identity -komponentin voi ajaa ClientRpc-toimintoja. ClientRpc-funktion voi merkitä ClientRpc-attribuutilla ja lisäämällä funktion nimeen etuliitteen ”Rpc” (Koodiesimerkki 2). Käytännössä tämä tarkoittaa sitä, että kun palvelin kutsuu funktiota, kaikki käyttäjät ajavat kyseisen funktion. Ainoastaan palvelin voi kutsua ClientRpc-funktioita. (Unity Technologies, 2019m.)

Koodiesimerkki 2. Merkitsemällä funktio ClientRpc-etätoiminnoksi se ajetaan jokaisella käyttäjällä, kun palvelin kutsuu sitä. Esimerkin funktio tulostaa vahingon määrän jokaiselle käyttäjälle. (Unity Technologies, 2019m.)

```
class Player : NetworkBehaviour
{

    [SyncVar]
    int health;

    [ClientRpc]
    void RpcDamage(int amount)
    {
        Debug.Log("Took damage:" + amount);
    }

    public void TakeDamage(int amount)
    {
        if (!isServer)
            return;

        health -= amount;
        RpcDamage(amount);
    }
}
```

3.3.2 Command

Command-etätoiminnot toimivat käyttäjien puolelta palvelimen suuntaan (Unity Technologies, 2019m). Käyttäjä ei itse aja funktiota, vaan se pyytää palvelinta suorittamaan funktion sen puolesta. Command-funktio määritellään käyttämällä Command-attribuuttia ja lisäämällä funktion nimen etuliitteeksi ”Cmd”. Turvallisuussyistä Command-etätoimintoja voi ajaa ainoastaan käyttäjän omasta pelaajaobjektista. Koska Command-etätoiminnot ajetaan palvelimen puolella, niitä voidaan hyödyntää ajamaan funktioita, joita käyttäjä ei tavallisesti pystyisi suorittamaan, kuten uusien verkostoitujen peliobjektien luominen (Koodiesimerkki 3).

Koodiesimerkki 3. Command-etätoiminnolla käyttäjä pyytää palvelinta ajamaan funktion sen puolesta. Esimerkin funktio luo uuden ammutun ammuksen, jota käyttäjä ei itse pystyisi luomaan. (Unity Technologies, 2019m.)

```
[Command]
void CmdDoFire(float lifeTime)
{
    GameObject bullet = (GameObject)Instantiate(
        bulletPrefab,
        transform.position + transform.right,
        Quaternion.identity);

    var bullet2D = bullet.GetComponent<Rigidbody2D>();
    bullet2D.velocity = transform.right * bulletSpeed;
    Destroy(bullet, lifeTime);

    NetworkServer.Spawn(bullet);
}

void Update()
{
    if (!isLocalPlayer)
        return;

    if (Input.GetKeyDown(KeyCode.Space))
    {
        CmdDoFire(3.0f);
    }

}
```

3.4 Unity Multiplayer Services

Unity Multiplayer Services (UMS) on Unityn tarjoama palvelu, jonka on tarkoitus helpottaa nettipelien järjestämistä sovelluksissa. Se antaa käyttäjien noutaa listan aktiivisista nettipeleistä Unityn palvelimilta, luoda nettipelejä, jotka näkyvät muille käyttäjille, sekä liittyä nettipeleihin. (Unity Technologies, 2019n.) Ilman tätä palvelua käyttäjät voivat liittyä toistensa peleihin ainoastaan IP-osoitteiden kautta. Tämä ei

tarkoita sitä, että UMS on välttämätön moninpeleille; se on vain yksi ratkaisu nettipelien julkiseen listaamiseen.

UMS sisältää ”Live”-tilan, joka on suunniteltu peleille, jotka aiotaan julkaista markkinoille. Palvelun hinta määräytyy pelaajamäärän ja nettikaistakäytön mukaan. Tämän lisäksi Unity sallii pienen määrän pelaajia palvelussa ilman maksullista ohjelmaa. Sallittu määrä riippuu kehittäjien Unity-lisenssistä: personalissa (ilmainen) määrä on 20, plussassa 50 ja prossa 200.

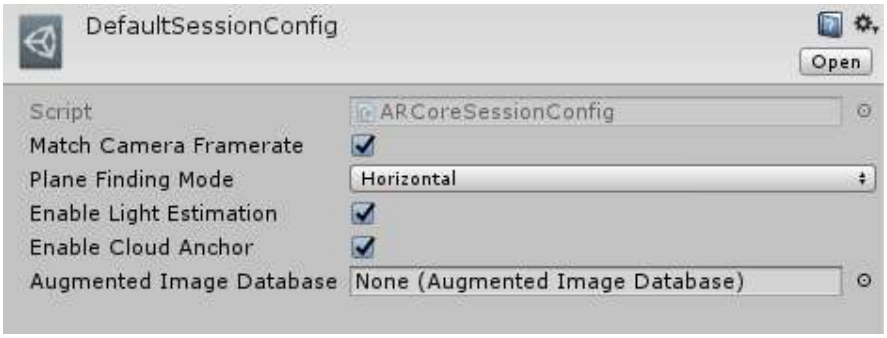
4 UNITYN ARCORE SDK

4.1 Prefabit

4.1.1 ARCore Device

ARCore Device on ARCoren toiminnan ydin. Se sisältää ARCoreSession-komponentin, jonka tehtävä on hallita ARCore-istuntoa. Kun prefab on aktiivinen scenessä, se luo uuden ARCore-istunnon automaattisesti. Prefab sisältää alapeliobjektina First Person Camera -peliobjektin, joka sisältää Camera-komponentin. Peliobjekti sisältää ARCoreBackgrounRender-komponentin, joka heijastaa laitteen kameran videokuvan pelin kameran taustakuvaksi. (Google, 2019a.)

ARCoreSession-komponentti sisältää Session Config -asetuksen, joka on pakollista määrittää (Google, 2019a). Oletuksena tämä kenttä viittaa DefaultSessionConfig-tiedostoon, joka tulee SDK:n mukana. Sen sijaan, että kentän tiedostoa päivittäisi uudella asetustiedostolla, kannattaa muokata jo olemassa olevaa oletustiedostoa. Sen löytää helposti kirjoittamalla projektinäkymän hakukenttään ”DefaultSessionConfig”. Tiedosto sisältää muutamia määriteltäviä asetuksia, kuten pitäisikö sen etsiä pintoja horisontaalisesti tai vertikaalisesti, vai sekä että (Kuva 2).



Kuva 2. DefaultSessionConfig-tiedosto.

Vain yksi ARCoreDevice-peliobjekti saa olla olemassa scenessä. Samalla kuin peliobjekti tuhotaan, poistetaan myös nykyinen ARCore-istunto (Google, 2019a).

4.1.2 Environmental Light

Environmental Light -prefabin tarkoitus on muokata valaistusta pelattavan pelin ympäristön valaistuksen näköiseksi. Prefabin käyttö ei ole pakollista lisätyn todellisuuden luomiseksi, mutta se auttaa tekemään siitä todenmukaisempaa.

Prefab sisältää EnvironmentalLight-komponentin, jonka tehtävä on päivittää globaalia _GlobalLightEstimation-muuttujaa. Mikäli prefabia hyödyntää, täytyy jokaisen pelissä esiintyvän peliobjektin shader päivittää käyttämään SDK:n mukana tulevia shader-tiedostoja: ”DiffuseWithLightEstimation” tai ”SpecularWithLightEstimation”. Huomioitavaa näitä shader-tiedostoja käyttäessä on, että mikäli EnvironmentalLight-komponenttia ei ole ehditty ajamaan pelissä, peliobjektit, jotka hyödyntävät tiedostoja, esiintyvät pelissä täysin mustina, kunnes komponenttia ajetaan ensimmäisen kerran. (Google, 2019b.)

4.2 Luokat

4.2.1 Session

Session-luokka esittää ARCore-istuntoa. Se sisältää tietoa ARCoren tilasta ja hallitsee ankkureiden ja pintojen seurantaa (Google, 2019c). Luokka on kokonaisuudessaan staattinen, joten sitä voi kutsua missä tahansa koodissa. Se sisältää muutamia funktioita, joita voi hyödyntää muualla koodissa. CheckApkAvailability- ja RequestApkInstallation-funktioilla voidaan tarkistaa, onko ARCore-APK asennettu laitteelle ja tarvittaessa asentaa se. Session-luokalla voidaan myös luoda ankkureita CreateAnchor-funktiolla. Funktio ottaa syötteenä Pose-tyypin, joka määrittelee ankkurin sijainnin Unityn maailmassa, ja Trackable-tyypin, joka määrittelee seuratun elementin, johon kiinnittää ankkuri. Seurattavaa elementtiä ei ole pakko määrittää ankkurille.

4.2.2 Anchor

Anchor-luokka esittää ankkureita ARCoressa. Ankkurit toimivat ns. yhteytenä ARCoren seurattavien elementtien ja pelin objektien välillä (Google, 2019d). Ankkurit ovat tarpeellisia, koska seurattavien elementtien sijainti saattaa muuttua sen mukaan, kun

ARCore oppii ympäristöstä enemmän (Google, 2019e). Ilman ankkureita, pelin objektit saattavat alkaa ”liukumaan” pois niiden tarkoitetulta sijainniltaan. Määrittelemällä ankkuri varmistetaan, että objektit pysyvät paikallaan suhteessa ympäristöön.

Anchor-luokka ei itse sisällä erityisiä funktioita, ja omaa vain yhden muuttujan, joka ilmoittaa ankkurin sen hetkisen seurantatilan (Google, 2019d). Niitä voidaan luoda mm. Session-luokan kautta.

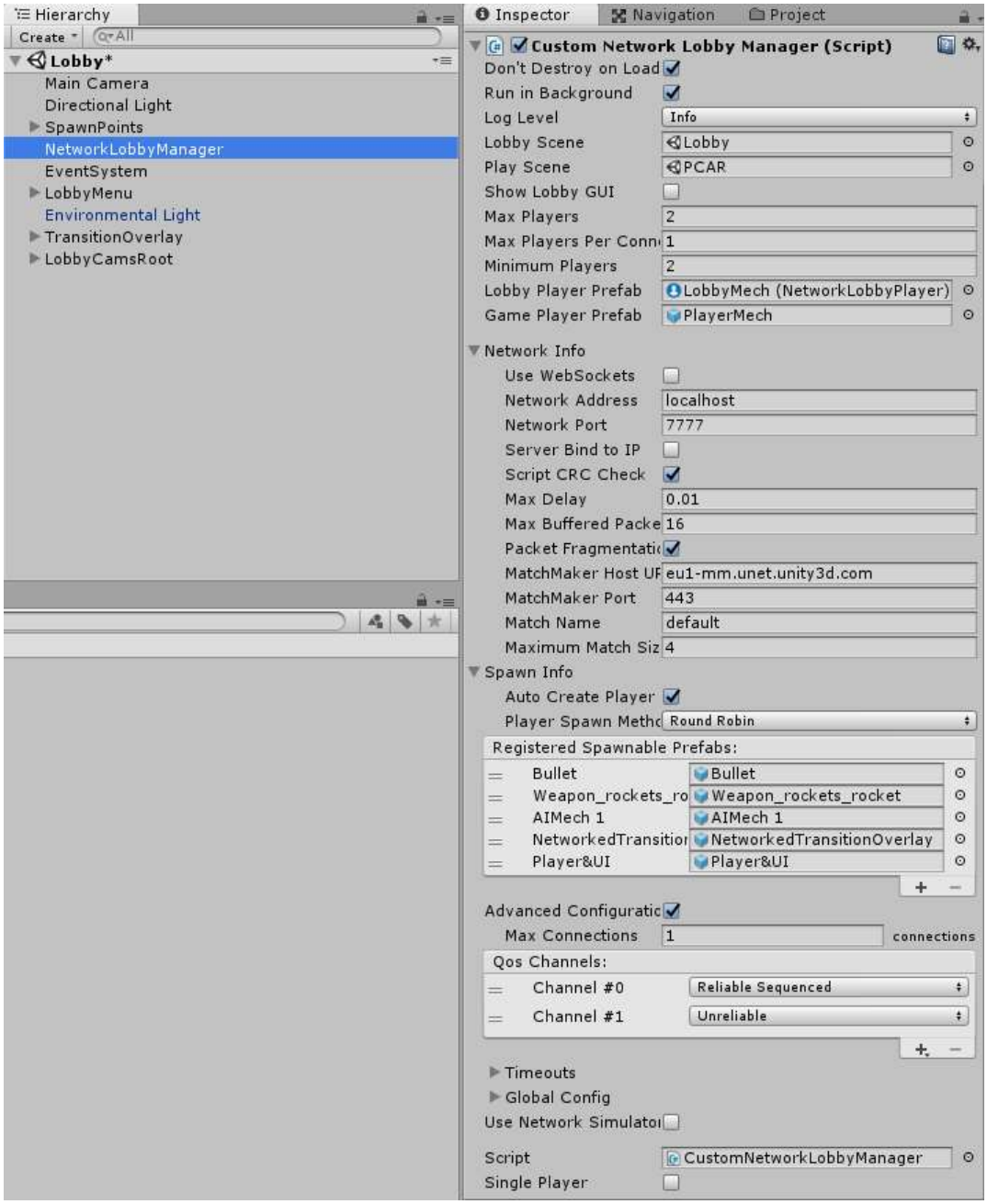
5 PROTOTYYPIN KEHITTÄMINEN

5.1 NetworkLobbyManagerin asentaminen

Koska NetworkManager on nettipelin keskeisin komponentti, se päätettiin asentaa ensimmäisenä projektiin. Pelissä tulisi olemaan aula, joten käytettiin komponentin muunnelmaa, NetworkLobbyManageria(NLM). Jotta luokan monia callback-metodeja voitaisiin hyödyntää, täytyi luoda uusi luokka, joka perii NLM-luokan. Uuden luokan nimeksi annettiin ”CustomNetworkLobbyManager” merkitsemään, että se sisältää muokattuja ominaisuuksia. Uuden luokan tekemisen etuna oli myös uusien muuttujien ja metodien luominen.

Olemassa olevaan aulaan luotiin uusi peliobjekti nimeltä ”NetworkLobbyManager”, johon kiinnitettiin räätälöity NLM-luokka (Kuva 1). Unityn inspectorin kautta täytyi käydä läpi komponentin asetukset. ”Don’t destroy on Load”-toiminto kytkettiin päälle, sillä peliobjekti ja komponentti haluttiin säilyttää, vaikka vaihdettaisiin sceneä. Lobby-sceneksi asetettiin nykyinen aula ja play-sceneksi ”PCAR”-niminen scene, joka on projektissa käytettävä scene pelaamista varten.

NLM-komponenttiin asetettiin oletusarvoja pelaajien enimmäis- ja vähimmäismäärille, mutta näitä arvoja muutetaan sovelluksessa peliasetusten mukaan. Koska aulassa esitetään pelaajat fyysisesti, lisättiin LobbyMech-peliobjekti ”lobby player prefab”-asetukseen. ”Game player prefab”-asetukseen lisättiin PlayerMech-peliobjekti, joka on pelaajan hallitsema objekti pelissä. Kaikki elementit, jotka tultaisiin luomaan palvelimen puolella kaikkien käyttäjien nähtäväksi täytyi myös lisätä komponenttiin. Tähän sisältyivät pelaajien luomat ammukset ja raketit. Komponenttiin lisättiin ”Single Player”-muuttuja, jolla voitaisiin hallita toiminnallisuutta sen mukaan pelaako yksin vai nettipelissä.



Kuva 3. Valittu NetworkLobbyManager Lobby-scenessä (vasemmalla) esittää asetetut komponentin asetukset (oikealla).

Seuraavaksi luotiin ohjaimet, joilla käyttäjä voi luoda tai liittyä nettipeleihin. Koska napeilla kutsuttaisiin metodeja koodissa, luotiin uusi NetworkingButtons-luokan, johon sisällytettiin kaikki napeilla käytettävät metodit. Kyseisen luokan voisi sitten tarpeen mukaan liittää peliobjekteihin ja kutsua vain haluttuja metodeja. Luokkaan lisättiin

metodit CallFunctionName, RoutineStartHost, RoutineStartClient (Koodiesimerkki 2). Coroutine-metodit (suomeksi sivurutiini) kutsutaan CallFunctionName-metodin kautta. Tämä menetelmä tehtiin Transitioner-luokkaa varten, jolla käyttäjän ruutu pimennetään ja tuodaan takaisin esiin. Ajamalla metodit sivurutiineina niiden ajo voidaan pysäyttää pimentymisten ajaksi. Kun ruutu on pimentynyt sovellus piilottaa ja tuo esiin oikeat valikkoelementit ja ajaa pelin luomisen tai liittymisen metodin NLM-komponentin kautta.

```
0 references
public void CallFunctionName(string functionName) {
    StartCoroutine(functionName);
}

0 references
public IEnumerator RoutineStartHost() {
    yield return StartCoroutine(Transitioner.FadeOut(1, 0));
    GameObject.Find("LobbyMenu").transform.Find("PostGameMenu").GetComponent<CanvasGroup>().alpha = 1;
    GameObject.Find("LobbyMenu").transform.Find("PostGameMenu").GetComponent<CanvasGroup>().blocksRaycasts = true;
    GameObject.Find("LobbyMenu").transform.Find("LANMenu").GetComponent<CanvasGroup>().alpha = 0;
    GameObject.Find("LobbyMenu").transform.Find("LANMenu").GetComponent<CanvasGroup>().blocksRaycasts = false;
    lobbyCams.SetActive(true);
    nLM.SetSinglePlayer(false);
        NetworkServer.dontListen = false;
        NetworkLobbyManager.singleton.StartHost();
    StartCoroutine(Transitioner.FadeIn(1, 0));
}

0 references
public IEnumerator RoutineStartClient() {
    SuicideText.singleton.RunRoutine("ConnectingText");
    string IP = GameObject.Find("IPField").GetComponent<InputField>().text;
    NetworkLobbyManager.singleton.networkAddress = IP;
    PlayerPrefs.SetString("lastIP", IP);
    yield return StartCoroutine(Transitioner.FadeOut(1, 0));
    GameObject.Find("LobbyMenu").transform.Find("PostGameMenu").GetComponent<CanvasGroup>().alpha = 1;
    GameObject.Find("LobbyMenu").transform.Find("PostGameMenu").GetComponent<CanvasGroup>().blocksRaycasts = true;
    GameObject.Find("LobbyMenu").transform.Find("LANMenu").GetComponent<CanvasGroup>().alpha = 0;
    GameObject.Find("LobbyMenu").transform.Find("LANMenu").GetComponent<CanvasGroup>().blocksRaycasts = false;
    lobbyCams.SetActive(true);
    nLM.SetSinglePlayer(false);
    NetworkLobbyManager.singleton.StartClient();
    yield return new WaitForSeconds(0.5f);
    StartCoroutine(Transitioner.FadeIn(1, 0));
}
```

Koodiesimerkki 4. Metodit pelien luomiselle ja niihin liittymiselle.

Aulassa oli jo ”LobbyMenu”-niminen peliobjekti, joka sisälsi canvas-komponentin. Tämä peliobjekti sisältää alapeliobjekteja, jotka toimivat näkyminä eri alavalikoille. Peliobjektiin lisättiin uusi alapeliobjekti nimeltä ”LANMenu”, jonka alle luotiin tarvittavat elementit pelien hallinnalle (Kuva 2). Näkymään sisällytettiin nappeja pelin luomiselle ja liittymiselle. Koska peliin liittyminen vaatisi IP-osoitteen oli tarpeellista lisätä tekstikenttä näkymään, johon käyttäjä voi kirjoittaa osoitteen johon yhdistää. NetworkingButtons-luokka liitettiin LobbyMenu-peliobjektiin, ja siihen viittattiin LANmenun napeista. Napit asetettiin kutsumaan CallFunctionName-metodia, mutta niiden tarkoitusten mukaisten vastaavilla string-parametreilla.

Kuva 4. LANMenu näkymä sovelluksessa.

Koska aulassa esitetään pelaajat visuaalisesti, täytyi vielä päivittää NLM-komponenttiin liitettyä LobbyMech-peliobjektia. Peliobjekti vaatii NetworkLobbyPlayer(NLP)-komponentin valmiustilaa varten, joten se lisättiin peliobjektiin. NetworkIdentity-komponentti täytyi myös lisätä, koska peliobjekti luodaan palvelimen toimesta.

Valmiustilan päivittämistä varten käyttäjä tarvitsee ohjaimet, joten peliobjektiin luotiin luokka nimeltä ”LobbyPlayerControls”. Luokka sisältää SetReadyState-metodin, joka hyödyntää NLP-komponenttia valmiustilan ilmoittamiseksi. Metodin sisällä kutsutaan myös muita metodeja, joiden avulla varmistetaan, että nykyinen valmiustila havainnollistetaan sovelluksessa oikein. Peliobjektiin lisättiin ala-peliobjekti, joka sisältää canvas-komponentin, ja sen ”Render Mode”-asetus määriteltiin ”World Space”-tilaan. Näin sen sisältö esitetään pelaajaobjektien edessä. Sitten siihen liitettiin nappi, jolla kutsutaan SetReadyState-metodia ja tekstielementtejä, joilla havainnollistetaan valmiustila.

Viimeiseksi, jotta pelaajat voidaan esittää visuaalisesti, täytyi luoda aloituspisteet, joihin käyttäjien peliobjektit luotaisiin. Aulaan lisättiin peliobjekteja, joihin sisällytettiin NetworkStartPosition-komponentit. Ongelmien välttämiseksi peliin liittymisessä ja poistumisessa haluttiin pitää kirjaa varatuista aloituspisteistä, jonka takia aloituspisteille luotiin LobbyStartPosition-luokka. Luokka sisältää boolean-tyyppisen occupied-muuttujan, sekä occupier-muuttujan, joka kertoo kuka sijaitsee aloituspisteessä.

Räätälöityjen luokkien hyödyntämiseksi täytyi käyttää NLM-komponentin OnLobbyServerCreateLobbyPlayer-callback metodia. Metodilla voidaan hallita käyttäytymistä, kun palvelin luo peliobjektin käyttäjälle aulassa. Kun käyttäjä liittyy peliin, palvelin käy jokaisen aloituspisteen läpi, kunnes se löytää vapaan pisteen. Kun vapaa piste on löydetty pelaajaobjekti luodaan, ja aloituspisteen varaustiedot päivitetään. Kun nämä muutokset oli tehty, nettipeli voitiin luoda ja siihen kyettiin liittymään alavalikon nappien kanssa. Pelaajaobjektit ilmestyvät näkymään, ja kun kaikkien pelaajien valmiustila on ”Ready”, palvelin siirtää kaikki pelaajat PCAR-sceneen (Kuva 3).

Kuva 5. Nettipelin aulanäkymä ennen pelin aloittamista.

5.2 Skriptien päivittäminen moninpeliä varten

Monia peliobjekteja täytyi päivittää sisältämään UNet-komponentteja, ennen kuin sovellus pystyi toimimaan nettipelinä. Komponenttien lisäämisen yhteydessä oli tarpeellista päivittää peliobjektien skriptejä ja lisätä uusia skriptejä.

Käyttäjän ohjaama peliobjekti oli yksi oleellisimmista päivitettävistä peliobjekteista. Projektissa oli jo hahmon hallitsemista varten skripti, jonka nimi oli ”MechController”. Skripti sisältää jo etukäteen käyttäjälle konkreettista toiminnallisuutta, kuten hahmon liikuttamista, ruutukosketusten prosessointia, sekä hahmon eri osasten kääntämistä. Peliobjekti sisältää skriptejä myös alemmilla peliobjekteilla, jotka ovat vastuussa hyökkäystoiminnoista, kuten ammusten ja rakettien ampumisesta. Skriptit täytyi päivittää peliobjektissa sellaisiksi, että vain paikallinen käyttäjä olisi hallussa omasta

peliobjektista. Skriptit päivitettiin perimään NetworkBehaviour-luokkaa, ja niiden FixedUpdate-metodeihin lisättiin tarkistuksia isLocalPlayer-muuttujalla (Koodiesimerkki 5). Koska MechController-skriptissä ajettiin animaatioita toimintojen mukaan, animaatioiden funktiot täytyi päivittää käyttämään ClientRpc- ja Command-attribuutteja (Koodiesimerkki 6). Ilman näitä muutoksia animaatiot näkyisivät vain paikalliselle käyttäjälle. Lisäämällä NetworkTransform-komponentti peliobjektiin sen sijainti päivittyisi jatkuvasti palvelimella, joten liikkumiseen liittyviä muutoksia ei tarvinnut tehdä skriptiin.

Koodiesimerkki 5. isLocalPlayer-muuttujalla varmistetaan, että ainoastaan peliobjektin omistaja voi ajaa tiettyä koodia. Esimerkissä ainoastaan funktion alkuosa.

```
protected virtual void FixedUpdate() {

    if (!isLocalPlayer) {
        return;
    }

    leftStickDirection = jsMovement.InputDirection;

    if(leftStickDirection.magnitude != 0){
```

Koodiesimerkki 6. Käyttäjä pyytää palvelinta ajamaan CmdSetAnimation-funktion, minkä seurauksena palvelin pyytää kaikkia käyttäjiä ajamaan RpcSetAnimation-funktion. Lopputuloksena on saman animaation ilmeneminen kaikille käyttäjillä.

```
[Command]
3 references
protected void CmdSetAnimation(string animator, string trigger) {
    RpcSetAnimation(animator, trigger);
}

[ClientRpc]
3 references
protected void RpcSetAnimation(string animator, string trigger) {
    animators[animator].SetTrigger(trigger);
}
```

Alapeliobjektien skriptejä oli myös tarpeellista päivittää käyttämään ClientRpc- ja Command-attribuutteja, koska ammukset täytyisi luoda palvelimella. Skripteissä ei kuitenkaan pystynyt suoraan käyttämään näitä attribuutteja, koska ne eivät sijainneet samalla tasolla NetworkIdentity-komponentin kanssa. Tämän takia pelaajaobjektin

ylimmälle tasolle luotiin ”PlayerNetworkingHelper”-skripti, jonka tehtävänä oli hoitaa alapeliobjektien luokkien toimintoja nettipelissä. Se muodostuu kahdesta Dictionary-kokoelmasta, jotka pitävät kirjaa käyttäjien aseista, sekä funktioista joilla aseet lisätään kyseisiin kokoelmiin. Kaikki funktiot alapeliobjektien luokissa, jotka vaativat attribuuttien lisäämistä, sisällytettiin myös PlayerNetworkingHelper-skriptiin päivitettyinä.

Koska ammukset luodaan palvelimella, ne täytyi myös päivittää perimään NetworkBehaviour-luokkaa. Projektissa on kahden tyyppisiä ammuksia: luoteja ja raketteja. Kummallakin on oma skripti, BulletController ja RocketController, mutta ne perivät samaa luokkaa: ShootableController. Päivittämällä ShootableController perimään NetworkBehaviour-luokkaa voidaan BulletController- ja RocketController-skripteihin lisätä etätoimintoja. Molemmissa skripteissä täytyi muuttaa törmäyksen havaitseminen toimimaan ainoastaan palvelimen puolella. Mikäli tarkistus tehdään kaikilla käyttäjillä, sama koodi ajetaan monta kertaa. Esimerkiksi yhden luodin energiavähennys saatetaan ajaa useampaan otteeseen. Skriptien OnCollisionEnter-funktioihin lisättiin tarkistuksia isServer-muuttujilla, varmistaen, että koodi ajetaan ainoastaan palvelimella (Koodiesimerkki 7). Skriptien tuhoamisfunktiot päivitettiin myös käyttämään ClientRpc-muuttujaa, jotta peliobjektit tuhottaisiin kaikkien käyttäjien puolella (Koodiesimerkki 8).

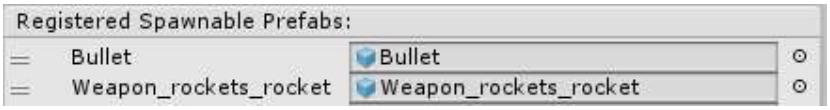
Koodiesimerkki 7. isServer-muuttuja varmistaa, että vain palvelin voi ajaa koodia.

```
0 references
void OnCollisionEnter(Collision other) {
    if (isServer) {
        if (!other.gameObject.Equals(sourceMech)) {
            if (other.gameObject.GetComponent<TargetHealth>()) {
                other.gameObject.GetComponent<TargetHealth>().DeductHealth(damage, sourceMech);
            }
            RpcDestroyRocket();
        }
    }
}
```

Koodiesimerkki 8. Funktiota kutsuessa jokainen käyttäjä ajaa sen sisällön. Funktio luo erikoistehosteen, jonka jälkeen se poistaa raketin.

```
[ClientRpc]
1 reference
void RpcDestroyRocket() {
    Instantiate(explosion, transform.position, Quaternion.identity, GameObject.Find("Fx").transform);
    Destroy(this.gameObject);
}
```

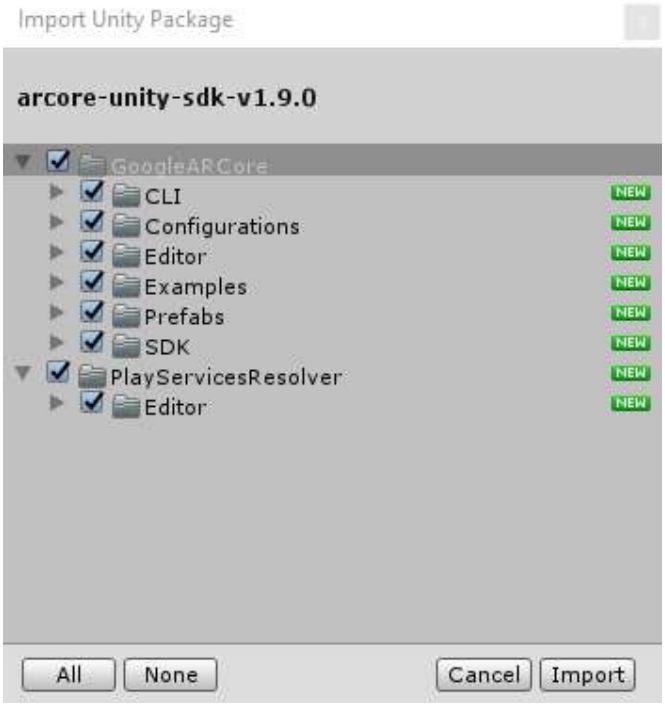
Luoti- ja rakettipeliobjekteihin lisättiin Network Identity -komponentit, jotta etätoiminnot toimisivat. Network Transform -komponentteja ei lisätty, vaikka ne ovat liikkuvia objekteja. Se ei ole tarpeellista, koska objektien elinaika on hyvin lyhyt, ja ne liikkuvat yksisuuntaisesti. Viimeiseksi peliobjektit rekisteröitiin NLM-komponenttiin, jotta palvelin pystyisi luoda niitä verkostoituina peliobjekteina (Kuva 6).



Kuva 6. Luoti- ja rakettipeliobjekti reskiteröitiin NLM-komponenttiin.

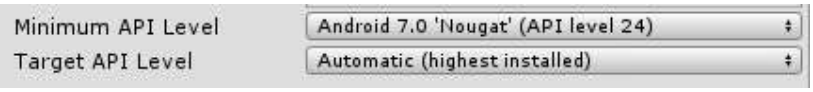
5.3 ARCoren yhteensovittaminen

Kun moninpelitoiminnot oli kehitetty, oli aika siirtyä ARCore integroimiseen projektiin. ARCore asennettiin projektiin lataamalla sen Unity SDK Google ARCoren nettisivulta. Ajamalla ladattu tiedosto aloitettiin asennusprosessi Unityssa auki olevaan projektiin. Unity aukaisi ikkunan, jossa voitiin hallita tarkemmin projektiin tuotavia elementtejä (Kuva 7). Kaikki paketissa olevat elementit asennettiin projektiin.



Kuva 7. Asennettavien elementtien hallintaikkuna.

SDK:n asennuttua valittiin File-valikosta ”Build Settings”, joka aukaisi ikkunan, josta voitiin siirtyä sovelluksen asetuksiin painamalla ”Player Settings”-nappia. Asetuksissa määriteltiin ”Other settings”-osiossa API-alarajatasoksi Android 7.0 (Kuva 8), ja ”XR Settings”-osiossa kytkettiin päälle ARCore-tuki (Kuva 9).

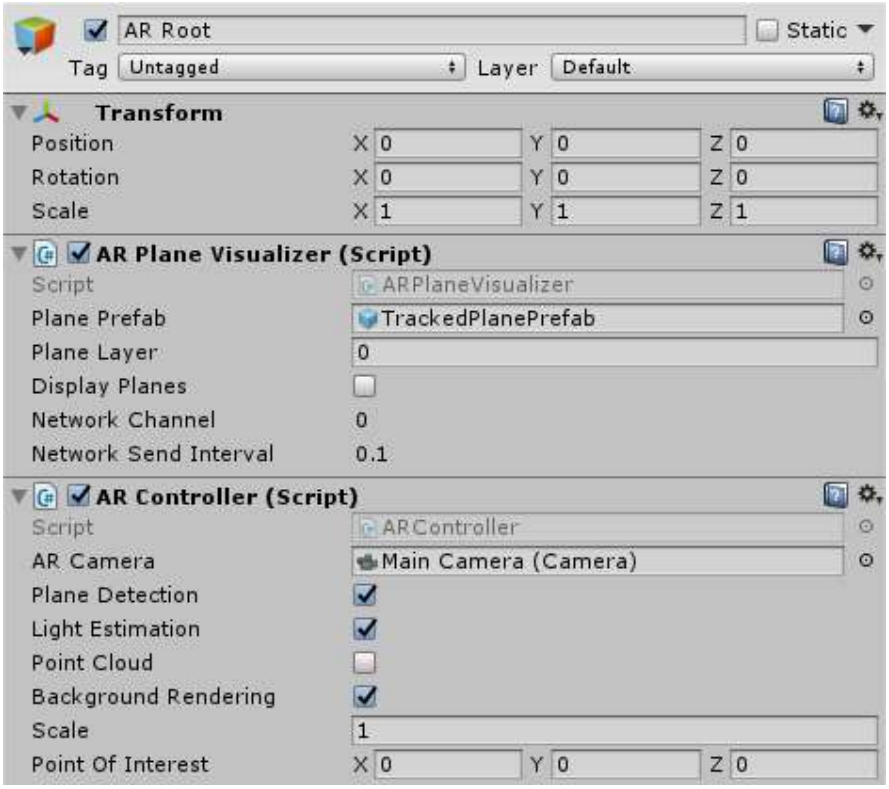


Kuva 8. API-asetukset sovellukselle.

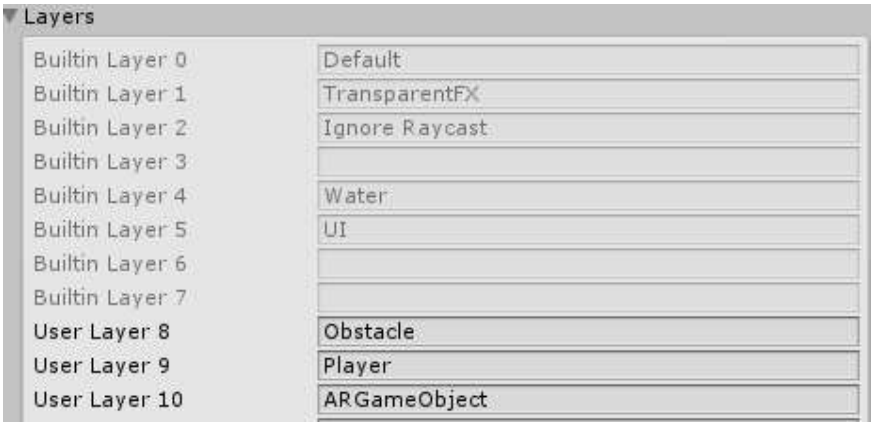
Kuva 9. ARCore-tuki kytketty päälle sovelluksen asetuksissa.

Projektissa hyödynnetään ARInterface-kirjastoa, jonka Unity Technologies on itse kehittänyt. Kirjaston käytön etuna on mm. havaittujen pintojen visualisointi automaattisesti, ja mahdollisuus testata AR-ympäristöä Unityn Editorissa. Opinnäytetyön kirjoistusajankohtana kirjasto on vanhentunut ja sen käyttäminen ei ole enää suotavaa. Unity Technologies on korvannut kirjaston ”AR Foundation”-paketilla, joka sisältää monia kirjaston tarjoamia ominaisuuksia.

Projektin PCAR-sceneen luotiin uusi peliobjekti nimeltä ”AR Root”, joka toimisi ARCoren ytimenä. Peliobjektiin asetettiin seuraavat ARInterface-kirjaston skriptit: AR Controller ja AR Plane Visualizer (Kuva 10). AR Plane Visualizer -skriptissä määriteltiin prefab, jota käytetään pintojen visualisointiin. Skripti käyttää visualisointiin ”ARGameObject”-kerrosta, joten kyseinen kerros täytyi määritellä projektin ”Layers”-asetuksissa (Kuva 11). Peliobjektin alapeliobjektiksi lisättiin Camera-peliobjekti, ja siihen viitattiin AR Controller -skriptissä. Skriptissä kytkettiin päälle Plane Detection, Light Estimation ja Background Rendering -asetukset. AR Root -peliobjektin alapeliobjektiksi lisättiin vielä ARCore SDK:n Environmental Light -prefab.



Kuva 10. AR Root -peliobjektiin lisättiin AR Controller ja AR Plane Visualizer -skriptit.



Kuva 11. ARGameObject-kerros lisättiin Layers-asetuksiin.

AR Plane Visualizer ei oletuksena sisällä asetusta visualisoinnin tilan vaihtamiseksi, joten sitä hieman räätälöitiin. Skriptiin lisättiin boolean-tyyppinen DisplayPlanes-muuttuja, jonka arvoa vaihtamalla havaittujen pintojen visualisointia voidaan hallita (Koodiesimerkki 9).

Koodiesimerkki 9. Vaihtamalla DisplayPlanes-muuttujan arvoa voidaan vaihtaa visualisoinnin tilaa.

```
public bool displayPlanes;
8 references
public bool DisplayPlanes {
    get { return displayPlanes;}
    set {
        if (value == displayPlanes) {
            return;
        }
        displayPlanes = value;
        if (displayPlanes) {
            foreach (GameObject go in m_Planes.Values) {
                go.SetActive(true);
            }
        }
        else {
            foreach (GameObject go in m_Planes.Values) {
                go.SetActive(false);
            }
        }
    }
}
```

Koska ARCoressa käyttäjien täytyy valita pinta, johon kiinnittää peliobjektit, moninpeliin täytyi asettaa tiettyjä askelia pelin alkamiseksi. Network Lobby Manager luo pelaajaobjektit automaattisesti pelin siirtyessä PCAR-sceneen. Pelaajien valitessa pintaa on siis mahdollista, että he havaitsevat pelaajaobjektit satunnaisesti ympäristössä, mikä voi aiheuttaa hämmennystä. Scenen käynnistyessä halutaan siis laittaa pelaajaobjektit pois päältä pinnan valitsemisen ajaksi. PCAR-scenessä on SceneController-peliobjekti, joka sisältää ARMechMaker-skriptin. Skriptin tehtävä on pelin aloittamisprosessin hallinnointi. Skriptiin lisättiin DeactivatePlayers-funktio, jota kutsutaan scenen käynnistyessä (Koodiesimerkki 10).

Koodiesimerkki 10. DeactivatePlayers-funktio odottaa 1,5 sekuntia, jonka jälkeen se laittaa jokaisen pelaajaobjektin pois päältä.

```
2 references
IEnumerator DeactivatePlayers() {
    yield return new WaitForSeconds(1.5f);
    foreach (NetworkConnection conn in NetworkServer.connections) {
        conn.playerControllers[0].gameObject.GetComponent<MechController>().ChangeActive(false);
    }
}
```

Askelien seuraamiseksi luotiin NetworkGameStatus-enumeraattori (engl. enumerator) (Koodiesimerkki 11). ARMechMaker-skriptiin lisättiin vastaava NetworkGameStatus-tyyppinen muuttuja. Samalla luotiin SetStatus- ja RpcSetStatus-funktiot, jotta tilaa voidaan päivittää (Koodiesimerkki 12). Koska jokainen askel määritellään enumeraattorissa, voidaan käyttäjälle esittää ohjeita askeleen mukaan. Sceneen luotiin uusi tekstielementti ja siihen uusi ”AssistAnnouncementController”-skripti. Skriptin Update-funktioon lisättiin jokainen NetworkGameStatus-tila, joka esittää tilan vastaavan tekstin käyttäjälle (Koodiesimerkki 13).

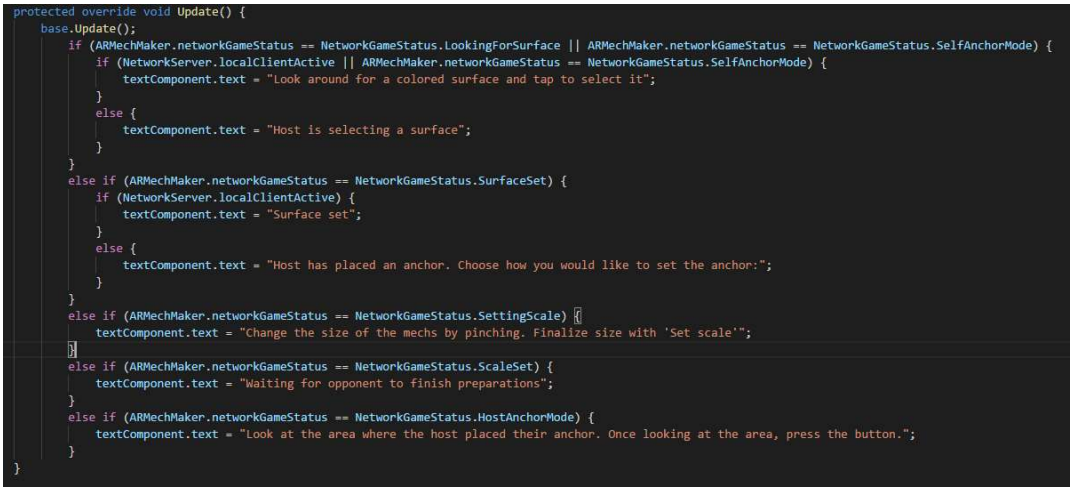
Koodiesimerkki 11. Enumeraattorilla voidaan helposti määrittää tila koodissa.

```
28 references
public enum NetworkGameStatus {
    3 references
    LookingForSurface = 0,
    4 references
    SurfaceSet = 1,
    0 references
    AnchorCreated = 2,
    5 references
    SettingScale = 3,
    2 references
    ScaleSet = 4,
    2 references
    HostAnchorMode = 5,
    4 references
    SelfAnchorMode = 6,
    1 reference
    Starting = 7,
    4 references
    Ongoing = 8,
    0 references
    Finished = 9,
}
```

Koodiesimerkki 12. SetStatus- ja RpcSetStatus-funktioilla päivitetään pelin tilaa.

```
[ClientRpc]
3 references
void RpcSetStatus(NetworkGameStatus newStatus) {
    if (newStatus == NetworkGameStatus.SurfaceSet) {
        if (!isServer) {
            networkGameStatus = newStatus;
            GameObject.Find("ScriptDelayer").transform.Find("TempUI/AnchorModeSelection").gameObject.SetActive(true);
        }
    }
    else {
        networkGameStatus = newStatus;
    }
}

0 references
public void SetStatus(NetworkGameStatus newStatus) {
    networkGameStatus = newStatus;
}
```

Koodiesimerkki 13. Käyttäjälle esitetään ohjetekstiä NetworkGameStatus-tilan mukaan.



```
protected override void Update() {
    base.Update();
        if (ARMechMaker.networkGameStatus == NetworkGameStatus.LookingForSurface || ARMechMaker.networkGameStatus == NetworkGameStatus.SelfAnchorMode) {
            if (NetworkServer.localClientActive || ARMechMaker.networkGameStatus == NetworkGameStatus.SelfAnchorMode) {
                textComponent.text = "Look around for a colored surface and tap to select it";
            }
            else {
                textComponent.text = "Host is selecting a surface";
            }
        }
        else if (ARMechMaker.networkGameStatus == NetworkGameStatus.SurfaceSet) {
            if (NetworkServer.localClientActive) {
                textComponent.text = "Surface set";
            }
            else {
                textComponent.text = "Host has placed an anchor. Choose how you would like to set the anchor:";
            }
        }
        else if (ARMechMaker.networkGameStatus == NetworkGameStatus.SettingScale) {
            textComponent.text = "Change the size of the mechs by pinching. Finalize size with 'Set scale'";
        }
        else if (ARMechMaker.networkGameStatus == NetworkGameStatus.ScaleSet) {
            textComponent.text = "Waiting for opponent to finish preparations";
        }
        else if (ARMechMaker.networkGameStatus == NetworkGameStatus.HostAnchorMode) {
            textComponent.text = "Look at the area where the host placed their anchor. Once looking at the area, press the button.";
        }
}
```

ARMechMaker-skriptiin lisättiin ohjaimet pinnan valitsemiselle. Ohjaimet lisättiin skriptin Update-funktioon (Koodiesimerkki 14). Pinta haluttiin valita vain kerran, minkä vuoksi skriptiin lisättiin boolean-tyyppinen firstClick-muuttuja, jolla varmistetaan, että pinta valitaan vain kerran. Klikkauksen tai sormenpainalluksen yhteydessä Camera-peliobjektista ammutaan raycast käyttämällä Physics-luokkaa. Raycast ampuu kamerasta viivan painalluksen suuntaan, ja mikäli viiva osuu pintaan, peliobjektit siirretään osuttuun sijaintiin. Osuma käsitellään hieman eri tavalla palvelimen ja muiden käyttäjien välillä. Palvelimen puolella kaikki peliobjektit siirretään valittuun pisteeseen, ja käyttäjille ilmoitetaan, että peli-istunnon isäntä on valinnut pinnan. Samaa menetelmää ei voi käyttää muiden käyttäjien suhteen, sillä verkostoitujen peliobjektien sijainti täytyy olla sama kaikkien käyttäjien välillä. Sen sijaan muilla käyttäjillä lasketaan isännän valitseman sijainnin ja käyttäjän valitseman sijainnin ero. Ero lisätään käyttäjän Camera-peliobjektin sijaintiin, joka luo illuusion peliobjektien siirtymisestä valittuun pisteeseen.

Koodiesimerkki 14. ARMechMaker-skriptin Update-funktiossa käsitellään pinnan valitseminen lisätyssä todellisuudessa.

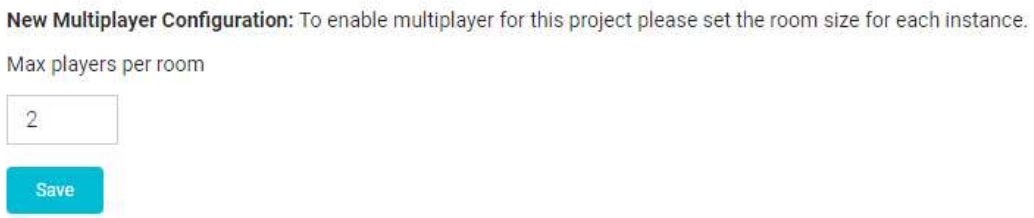
```
void Update () {
    if (!isActiveAndEnabled || (!isServer && networkGameStatus != NetworkGameStatus.SelfAnchorMode))
        return;

    if (Input.GetMouseButton (0)) {
        if (firstClick) {
            var camera = GetCamera();
            Ray ray = camera.ScreenPointToRay (Input.mousePosition);
            int layerMask = 1 << LayerMask.NameToLayer ("ARGameObject"); // Planes are in layer ARGameObject
            RaycastHit rayHit;
            Vector3 pos = Camera.main.ScreenToViewportPoint(new Vector3 (Input.mousePosition.x, Input.mousePosition.y, 0));
            if (Physics.Raycast (ray, out rayHit, float.MaxValue, layerMask)) {
                Vector3 position = rayHit.transform.position;
                Anchor anchor = Session.CreateAnchor(new Pose(position, rayHit.transform.rotation));
                anchor.transform.position = rayHit.point;
                anchor.transform.rotation = rayHit.transform.rotation;

                if (isServer) {
                    StartCoroutine("HostCloudAnchor", anchor);
                    CreateMechSet (new Vector3 (rayHit.point.x, rayHit.point.y, rayHit.point.z), anchor);
                    MovePlayersToSpawnPositions();
                    ParentPlayers(anchor);
                    RpcSetStatus(NetworkGameStatus.SurfaceSet);
                    networkGameStatus = NetworkGameStatus.SettingScale;
                }
                else {
                    //CalibrateClient(rayHit.point);
                    CalibrateClient(anchor);
                }

                this.enabled = false;
            }
            firstClick = false;
        }
```

Viimeiseksi peliin haluttiin lisätä mahdollisuus muuttaa peliobjektien suuruutta. Riippuen ympäristöstä pelin peliobjektit voivat olla liian suuria tai liian pieniä. Tämän toiminnallisuuden toteuttamiseksi hyödynnettiin ARCore-tutoriaalia Google Developers Codelabs (2019) -nettisivulta. Tutoriaalin yksi ladattava tiedosto on GlobalScalable-skripti, joka toteuttaa tämän toiminnon. Skripti lisättiin SceneController-peliobjektiin, ja siinä kytkettiin ”Handle Scale Input” -asetus päälle. Se myös lisättiin AR Root -peliobjektiin, jossa sen ”Adjust Scale” -toiminto kytkettiin päälle. Samalla skriptiä muokattiin hieman viittaamaan ARController-skriptin scale-muuttujaan.

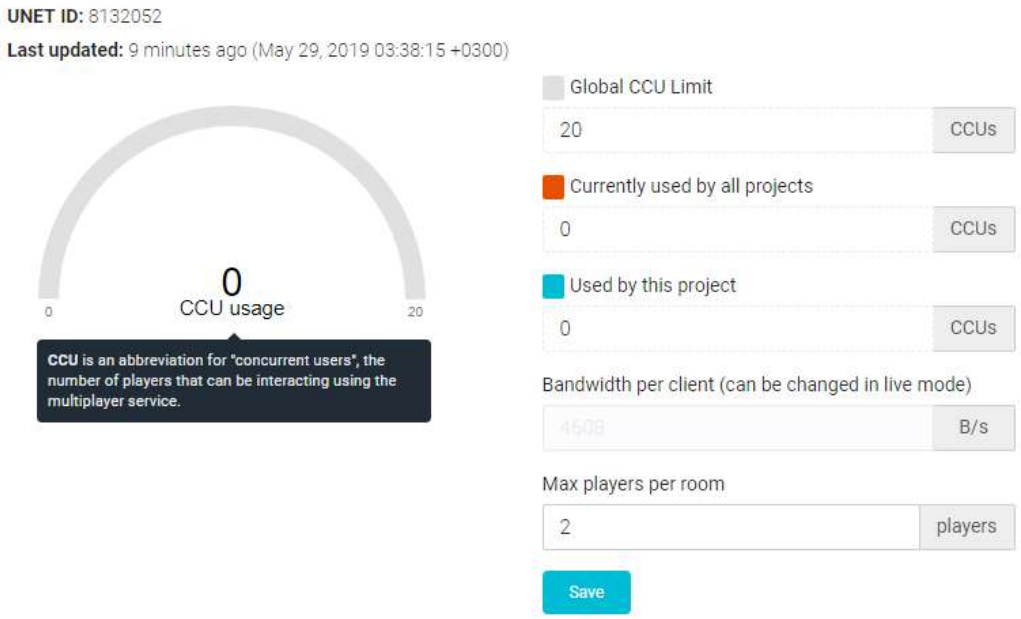
5.4 Unity Multiplayer -palvelun käyttöönotto

Lopuksi projektissa otettiin käyttöön Unity Multiplayer -palvelu, jotta käyttäjät voisivat helposti liittyä toistensa peli-istuntoihin. Aukaisemalla Unity Editorissa Services-ikkunan, päästään käsiksi Unityn tarjoamien palveluiden asetuksiin. Valitsemalla ikkunassa ”Multiplayer”, ja painamalla ”Go to dashboard”-nappia, päästään määrittelemään palvelun asetukset Unityn kehittäjäsivuilla. Määrittelemällä enimmäismäärän peli-

istunnoille, palvelu aktivoidaan projektissa (Kuva 12). Kehittäjäsivuilla voi myös tarkkailla palvelun käyttäjämäärän enimmäismäärää ja sen hetkistä käyttäjälukemaa palvelussa (Kuva 13).



Kuva 12. Peli-istunnoille määritellään enimmäismäärä pelaajia, mikä käynnistää Unity Multuplayer -palvelun projektissa.



Kuva 13. Kehittäjäsivuilla nähdään palvelun rajoitukset ja käyttäjämäärät.

Kun palvelu oli otettu käyttöön projektissa, peliin täytyi lisätä tapa käyttäjille nähdä saatavilla olevat peli-istunnot, sekä luoda niitä. Lobby-sceneen lisättiin uusi UI-peliobjekti, ”MatchmakingMenu”, joka toimisi valikkona peli-istunnoiden hallinnalle. Päävalikkoon lisättiin nappi, joka aukaisee uuden valikon. Samalla nappi ajaa StartMatchMaker-funktion NLM-komponentissa, joka yhdistää pelin palveluun.

Peli-istunnoiden noutamiseen luotiin MatchLister-skripti. Skriptiin lisättiin ListMatches-funktio, joka suorittaa peli-istunnoiden noutamisen NLM-komponentin kautta (Koodiesimerkki 15). Funktion suorittamisen jälkeen ajetaan OnMatchList-funktio (Koodiesimerkki 16). OnMatchList-funktio muotoilee jokaisesta noudetusta peli-istunnosta peliobjektin, joka lisätään listaan UI-elementtinä (Kuva 14). Mikäli peli-istuntoja ei löydy tai tapahtuu virhe niitä noutaessa, käyttäjälle näytetään tilannetta vastaava viesti. Funktiot on suunniteltu hakemaan kymmenen peli-istuntoa kerrallaan.

Koodiesimerkki 15. Funktio suorittaa haun Unity Multiplayer -palveluun.

```
2 references
public void ListMatches() {
    NetworkLobbyManager.singleton.matchMaker.ListMatches(pageNumber, 10, "", false, 0, 0, OnMatchList);
}
```

Koodiesimerkki 16. Funktio luo listan noudetuista peli-istunnoista ja näyttää käyttäjälle mahdollista virhetietoa.

```
public void OnMatchList(bool success, string extendedInfo, List<MatchInfoSnapshot> matches) {
    if (success) {
        if (matches.Count >= 1) {
            foreach (MatchInfoSnapshot match in matches) {
                GameObject listing = Instantiate(matchTemplate, matchLister.transform);
                listing.transform.Find("NameBox/Name").GetComponent<Text>().text = match.name;
                listing.transform.Find("PlayersBox/Players").GetComponent<Text>().text = match.currentSize + " / " + match.maxSize;
                if (!match.isPrivate) {
                    listing.transform.Find("PasswordBox/PasswordRequired").GetComponent<Image>().enabled = false;
                }
                listing.transform.Find("JoinButton").GetComponent<NetworkingButtons>().SetMatchInfo(match);
                matchesList.Add(listing);
                listing.SetActive(true);
            }
            if (matches.Count >= 10) {
                pageNumber++;
                ListMatches();
            }
            else {
                refreshButton.interactable = true;
                listInfoObject.GetComponent<Text>().text = "-- End of list --";
                listInfoObject.transform.SetAsLastSibling();
            }
        }
        else if (matches.Count == 0 && pageNumber == 0){
            listInfoObject.GetComponent<Text>().text = "-- No games available. -- \n -- Refresh list later or create a new game.  --";
            refreshButton.interactable = true;
        }
        listInfoObject.SetActive(true);
    }
    if (!success) {
        listInfoObject.GetComponent<Text>().text = "-- Could not contact matchmaking service. -- \n -- Make sure you are connected to the internet.  --";
        listInfoObject.SetActive(true);
        refreshButton.interactable = true;
    }
}
```

Kuva 14. Noudettu peli-istuntojen lista Unity Multiplayer -palvelusta.

MatchmakingMenu-valikkoon lisättiin alavalikko, joka aukaistaan painamalla ”Create game”-nappia (Kuva 15). NetworkingButtons-luokkaan lisättiin funktioita peli-istuntojen luomiselle ja niihin liittymiseen. CreateMatch- ja OnMatchCreate-funktioilla peli-istunto lisätään palveluun ja käynnistetään peli-istunto (Koodiesimerkki 17). CreateMatch-funktio lukee käyttäjän määrittelemät asetukset alavalikosta ja luo niistä lähetettävän

pyynnön palveluun. Jos pyyntö onnistuu, OnMatchCreate-funktio käynnistää peli-istunnon. Jos ei, funktio käsittelee virhetilan.

Kuva 15. Aukaistusta alavalikosta käyttäjä voi määritellä asetukset uudelle peli-istunnolle.

Koodiesimerkki 17. CreateMatch- ja OnMatchCreate-funktiot.

```
public void CreateMatch() {
    GetComponent<Button>().interactable = false;
    string name = transform.Find("../NameGroup/InputField").GetComponent<InputField>().text;
    int size = Int32.Parse(transform.Find("../SizeGroup/Dropdown/Label").GetComponent<Text>().text);
    string password = transform.Find("../PasswordGroup/InputField").GetComponent<InputField>().text;
    NetworkLobbyManager.singleton.matchSize = (uint)size;
    NetworkLobbyManager.singleton.matchMaker.CreateMatch(name, (uint)size, true, password, "", "", 0, 0, OnMatchCreate);
}

1 reference
void OnMatchCreate(bool success, string extendedInfo, MatchInfo matchInfo) {
    if (success) {
        NetworkLobbyManager.singleton.matchInfo = matchInfo;
        StartCoroutine("MatchRoutineStartHost", matchInfo);
    }
    else {
        if (extendedInfo.Contains("Cannot resolve destination host")) {
            Debug.Log(extendedInfo);
            transform.Find("../../ErrorText").gameObject.SetActive(true);
            GetComponent<Button>().interactable = true;
        }
    }
}
```

Vastaavasti JoinMatch-, JoinMatchWithPassword- ja OnMatchJoin-funktiot käsittelevät peli-istuntoon liittymisen. JoinMatch-funktio lähettää pyynnön peli-istuntoon liittymiseksi tai aukaisee salasanaikkunan, jos peli-istunnolle on asetettu salasana (Koodiesimerkki 18). JoinMatchWithPassword-funktio lukee käyttäjän syöttämän salasanan ja lähettää pyynnön liittymiselle. OnMatchJoin-funktio vastaanottaa peli-istunnon tiedot ja lisää ne NLM-komponenttiin, jonka jälkeen se liittyy peli-istuntoon (Koodiesimerkki 19). Mikäli liittymisessä on ongelmia, se käsittelee virhetilanteet ilmoittamalla niistä käyttäjälle.

Koodiesimerkki 18. JoinMatch- ja JoinMatchWithPassword-funktiot.

```
0 references
public void JoinMatch() {
    if (matchInfoSnapshot.isPrivate) {
        passwordOverlay.transform.Find("EnterPasswordPanel/JoinGameButton").GetComponent<NetworkingButtons>().matchInfoSnapshot = matchInfoSnapshot;
        passwordOverlay.SetActive(true);
    }
    else {
        GetComponent<Button>().interactable = false;
        NetworkLobbyManager.singleton.matchMaker.JoinMatch(matchInfoSnapshot.networkId, "", "", "", 0, 0, OnMatchJoin);
    }
}

0 references
public void JoinMatchWithPassword() {
    string password = transform.Find("../InputField").GetComponent<InputField>().text;
    GetComponent<Button>().interactable = false;
    NetworkLobbyManager.singleton.matchMaker.JoinMatch(matchInfoSnapshot.networkId, password, "", "", 0, 0, OnMatchJoin);
}
```

Koodiesimerkki 19. OnMatchJoin-funktio.

```
2 references
void OnMatchJoin(bool success, string extendedInfo, MatchInfo matchInfo) {
    if (success) {
        Transform error = transform.Find("../ErrorText");
        if (error != null) {
            error.gameObject.SetActive(false);
        }

        NetworkLobbyManager.singleton.matchInfo = matchInfo;
        StartCoroutine("MatchRoutineStartClient", matchInfo);
    }
    else if (!success) {
        Debug.Log(extendedInfo);
        if (extendedInfo.Contains("serverId=InvalidFormatException")) {
            transform.Find("../ErrorText").gameObject.SetActive(true);
        }
        else if (extendedInfo.Contains("Match is likely unavailable")) {
            transform.Find("Text").GetComponent<Text>().text = "Match unavailable";
            return;
        }
        GetComponent<Button>().interactable = true;
    }
}
```

6 JOHTOPÄÄTÖKSET

Opinnäytetyön tavoitteena oli muuttaa kehitettävän mobiilipelin prototyyppi lisätyn todellisuuden moninpeliksi käyttämällä Unity Networkingia ja ARCorea. Tavoitteen saavuttamiseksi työssä tutustuttiin tarvittaviin teknologiakomponentteihin, niiden käyttöön ja muutosprosessin vaiheisiin. Tavoite saavutettiin, ja tuloksena oli peli, jossa pelaajat voivat luoda ja liittyä peli-istuntoihin pelaamaan toisiaan vastaan lisätyssä todellisuudessa.

Luvuissa kaksi, kolme ja neljä käytiin läpi välttämättömät elementit ja työkalut, jotka kehittäjän täytyy hallita tämän tyyppisen projektin suorittamiseksi. Niitä tarvitaan myös muiden kaltaisten mobiilipelien muuttamisessa AR-moninpeliksi. Elementtien ja työkalujen käyttötarkoitukset ja käyttöohjeet selostettiin. Opinnäytetyö korostaa niiden käytössä rajoituksia, jotka voivat aiheuttaa ongelmia, jos niistä ei olla tietoisia. Network Identity -komponentti on esimerkki rajoituksesta; peliobjekti voi sisältää ainoastaan yhden tällaisen komponentin, mukaan lukien sen alapeliobjektit. Rajoituksesta ei ole Unityn omassa dokumentaatiossa mainintaa, jonka vuoksi on todennäköisempää, että työkalua käyttää väärällä tavalla.

Luvussa viisi kuvaillaan vaiheet muutostyön tekemiseksi em. komponenttien avulla: komponenttien asentaminen, skriptien päivittäminen, AR-alustan integroiminen ja Unity Multiplayer -palvelun käyttöönotto. Opinnäytetyön aiheena on toimintapeli, mutta edellä mainitut vaiheet ovat muutoksitta yleistettävissä muihin pelilajeihin, kuten strategiapeleihin tai pulmapeleihin. Network Managerin asentaminen on välttämätöntä moninpelille, ja vaikka pelissä ei olisi konkreettista pelaajahahmoa, pelaajille täytyy luoda skripti, joka käsittelee pelaajien toimintoja. Tuloksia on mahdollista hyödyntää erikseen kummin päin tahansa: pelin voi muuntaa moninpeliksi ilman lisättyä todellisuutta tai lisätyksi todellisuudeksi ilman moninpeliä.

Opinnäytetyön lopputulokset ovat yleistettävissä muihin mobiilipeleihin. Pelissä on monia mahdollisuuksia jatkokehitykselle. Nykyisessä muodossaan peli olisi melko yksinkertaista laajentaa tukemaan isompaa pelaajamäärää yhdessä peli-istunnossa. Pelaajahahmojen räätälöintiominaisuutta voisi tehdä merkitsevämmäksi antamalla muokattaville osille erilaisia peliin vaikuttavia ominaisuuksia. Esimerkiksi jotkin osat voisivat antaa käyttäjälle enemmän energiapisteitä, ja toiset voisivat aiheuttaa enemmän vahinkoa vastustajiin.

Unity oli minulle entuudestaan tuttu tekniikka, mutta tämä oli ensimmäinen kerta, kun työskentelin sen UNet-komponenteilla. Komponenttien opiskelemiseen käytin Unityn omaa ohjekirjaa sekä Scripting APIa. Työn edetessä kuitenkin ilmeni, että komponenttien dokumentaatio oli hyvin puutteellista. Tämä johti välillä kummalliseen ja odottamattomaan komponenttien käyttäytymiseen sovelluksessa. Näiden odottamattomien käyttäytymisten selvittelyyn kului työaikaa, kun jouduin etsimään selityksiä muilta Unityn käyttäjiltä ja kehittäjiltä. Unity-kohtaisten ongelmien selvittämiseen kannattaa hyödyntää ”Unity Answers”- ja ”Unity Forums”-nettisivustoja. Apua saattaa löytää myös ”StackOverflow”-yhteisöstä. Aloittaessani opinnäytetyötä ARCore oli uusi asia eikä siitä ollut vielä monia esimerkkejä eikä tutoriaaleja. Tämä hankaloitti sen opettelemista. Kokonaisuudessaan dokumentaation puute opinnäytetyön tekniikoissa johti yritys ja erehdys -kehitysmenetelmään, joka ei ollut tehokasta.

LÄHTEET

ARPost, 2018. *Key Statistics for the Virtual and Augmented Reality Industry in 2018.* [Online]
Available at: https://arpost.co/2018/05/23/key-statistics-virtual-and-augmented-reality-industry-2018/
[Haettu 1 11 2018].

Blacker, A., 2018. [Online]
Available at: https://blog.apptopia.com/pok%C3%A9mon-go-catches-2-billion-since-launch#blogSubscribe
[Haettu 1 11 2018].

CCS Insight, 2018. *Virtual Reality and Augmented Reality Device Market Worth $1.8 Billion in 2018.* [Online]
Available at: https://www.ccsinsight.com/press/company-news/3451-virtual-reality-and-augmented-reality-device-market-worth-18-billion-in-2018
[Haettu 1 11 2018].

Google Developers Codelabs, 2019. *Introduction to ARCore in Unity.* [Online]
Available at: https://codelabs.developers.google.com/codelabs/arcore-intro/
[Haettu 29 5 2019].

Google, 2018. [Online]
Available at: https://play.google.com/store/apps/top/category/GAME
[Haettu 1 11 2018].

Google, 2019a. *ARCore Device prefab.* [Online]
Available at: https://developers.google.com/ar/reference/unity/prefab/ARCore_Device
[Haettu 26 5 2019].

Google, 2019b. *Environmental Light prefab.* [Online]
Available at:
https://developers.google.com/ar/reference/unity/prefab/Environmental_Light
[Haettu 26 5 2019].

Google, 2019c. *GoogleARCore.Session.* [Online]
Available at:
https://developers.google.com/ar/reference/unity/class/GoogleARCore/Session
[Haettu 26 5 2019].

Google, 2019d. *GoogleARCore.Anchor.* [Online]
Available at:
https://developers.google.com/ar/reference/unity/class/GoogleARCore/Anchor
[Haettu 26 5 2019].

Google, 2019e. *Fundamental concepts.* [Online]
Available at: https://developers.google.com/ar/discover/concepts
[Haettu 26 5 2019].

Kipper, G. & Rampolla, J., 2012. *Augmented Reality: An Emerging Technologies to AR.* s.l.:William Andrew.

Montegriffo, N., 2019. *What is an APK file and how do you install one?.* [Online]
Available at: https://www.androidpit.com/android-for-beginners-what-is-an-apk-file
[Haettu 26 5 2019].

Sandoval, K., 2016. *What is the Difference Between an API and an SDK?.* [Online]
Available at: https://nordicapis.com/what-is-the-difference-between-an-api-and-an-sdk/
[Haettu 26 5 2019].

Statista, 2018. *Most played PC games on gaming platform Raptr in November 2015, by share of playing time.* [Online]
Available at: https://www.statista.com/statistics/251222/most-played-pc-games/
[Haettu 1 11 2018].

Unity Technologies, 2018g. *Using the Network Manager.* [Online]
Available at: https://docs.unity3d.com/Manual/UNetManager.html
[Haettu 4 2 2018].

Unity Technologies, 2019a. *GameObjects.* [Online]
Available at: https://docs.unity3d.com/Manual/GameObjects.html
[Haettu 16 5 2019].

Unity Technologies, 2019b. *GameObject.* [Online]
Available at: https://docs.unity3d.com/Manual/class-GameObject.html
[Haettu 16 5 2019].

Unity Technologies, 2019c. *Introduction to components.* [Online]
Available at: https://docs.unity3d.com/Manual/Components.html
[Haettu 16 5 2019].

Unity Technologies, 2019d. *Using Components.* [Online]
Available at: https://docs.unity3d.com/Manual/UsingComponents.html
[Haettu 16 5 2019].

Unity Technologies, 2019e. *Prefabs.* [Online]
Available at: https://docs.unity3d.com/Manual/Prefabs.html
[Haettu 16 5 2019].

Unity Technologies, 2019f. *Scenes.* [Online]
Available at: https://docs.unity3d.com/Manual/CreatingScenes.html
[Haettu 16 5 2019].

Unity Technologies, 2019h. *Network Lobby Manager.* [Online]
Available at: https://docs.unity3d.com/Manual/class-NetworkLobbyManager.html
[Haettu 19 5 2019].

Unity Technologies, 2019i. *Network Identity.* [Online]
Available at: https://docs.unity3d.com/Manual/class-NetworkIdentity.html
[Haettu 19 5 2019].

Unity Technologies, 2019j. *Network Transform.* [Online]
Available at: https://docs.unity3d.com/Manual/class-NetworkTransform.html
[Haettu 19 5 2019].

Unity Technologies, 2019k. *NetworkBehaviour.* [Online]
Available at: https://docs.unity3d.com/Manual/class-NetworkBehaviour.html
[Haettu 19 5 2019].

Unity Technologies, 2019l. *NetworkServer.* [Online]
Available at:
https://docs.unity3d.com/540/Documentation/ScriptReference/Networking.NetworkServer.html
[Haettu 19 5 2019].

Unity Technologies, 2019m. *Remote Actions.* [Online]
Available at: https://docs.unity3d.com/Manual/UNetActions.html
[Haettu 19 5 2019].

Unity Technologies, 2019n. *Matchmaker.* [Online]
Available at:
https://docs.unity3d.com/520/Documentation/Manual/UNetMatchMaker.html
[Haettu 19 5 2019].

UserTesting, 2018. *UI vs. UX: What's the difference between user interface and user experience?.* [Online]
Available at: https://www.usertesting.com/blog/ui-vs-ux/
[Haettu 29 5 2019].