Osaamista
ja oivallusta
tulevaisuuden
tekemiseen

Henri Hakala

Monen käyttäjän persistentti lisätty todellisuus

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikan koulutusohjelma

Insinöörityö

10.9.2019

Tekijä Otsikko Sivumäärä Aika	Henri Hakala Monen käyttäjän persistentti lisätty todellisuus 28 sivua 10.9.2019
Tutkinto	Insinööri (AMK)
Tutkinto-ohjelma	Tietotekniikan koulutusohjelma
Ammatillinen pääaine	Ohjelmistotuotanto
Ohjaajat	Janne Salonen

Insinöörityön tavoitteena on tutkia lisätyn todellisuuden tilannetta nykyhetkellä ja siinä olevia persistenssin mahdollistavia kokonaisuuksia. Lisäksi myös vilkaistaan mistä lisätty todellisuus on kehittynyt ja mihin suuntaan se on menossa tulevaisuudessa.

Tarkoituksena on myös toteuttaa Proof Of Concept tyypinen monen käyttäjän persistentti lisätyn todellisuuden sovellus jossa tutkitaan mitä haasteita ja mahdollisuuksia lisätty todellisuus tuo. Sovelluksen tarkoitus on auttaa sisätiloissa navigoinnissa virtuaalisen avustajan avulla joka osaa ohjata sovelluksen käyttäjät käytäviä pitkin ennaltamääriteltyihin kohtiin. Sovelluksen lopullinen toteutus tehtiin IPad pro tabletille ARKitin osoittautuessa parhaaksi ratkaisuksi persistenssin toteuttamisessa.

Työ ei käy yksityiskohtaisesti käytettyjä työkaluja tai teknologioita läpi joten työn lukeminen edellyttää perusymmärrystä peliteknologioista sekä lisätyn todellisuuden konsepteista.

Avainsanat	Lisätty todellisuus, moninpeli, unity

Author	Henri Hakala
Title	Persistent multi-user augmented reality
Number of Pages	28 pages
Date	10 September 2019
Degree	Bachelor of Engineering
Degree Programme	Degree Program in Information Technology
Professional Major	Software development
Instructors	Janne Salonen

The goal of this thesis is to study the state of AR (Augmented Reality) at present and to find out the technologies and frameworks that make persistence in it possible.
In addition we also take a small peek at where AR has come from and where its headed in the future.

We'll also implement a Proof Of Concept persistent multi-user AR application in which we investigate the challenges and possibilities that augmented reality brings. The main intention of this application is to help users navigate indoors with the help of a virtual asstant who can help the users navigate along corridors to predefined spots. The final implementation of this application was made on the IPad pro tablet due to ARKit proving to be the best solution to implement persistence in AR.

The thesis doesn't go into great detail about the tools and technologies used and assumes the reader has basic understanding of game technologies and augmented reality concepts.

Keywords	Augmented reality, multiplayer, unity

Sisällys

Lyhenteet

AR	Augmented reality. Teknologia jolla voidaan lisätä virtuaalista sisältöä oikeaan maailmaan.
VR	Virtual reality. Keinotekoinen virtuaalinen todellisuus.
XR	Yleinen termi AR/VR teknologioille.
ARKit	Applen rajapinta IOS laitteiden AR ominaisuuksille.
ARCore	Googlen rajapinta android laitteiden AR ominaisuuksille.
Persistenssi	AR kontekstissa tarkoittaa sisällön pysymistä samoissa paikoissa sessioiden välillä.
Shader	Pieni koodinpätkä joka määrittelee miten geometria piirretään ruudulle
Relokalisointi	Yrityksen sijoittaa muistissa olevan pistepilvi kameran kuvan pistepilven mukaan.

1 Johdanto

Lisätty todellisuus on viime vuosina kehittynyt hurjaa tahtia. Monet maailman suurimmista teknologiayrityksistä kuten Microsoft/Apple/Google ovat alkaneet panostaa lisätyn todellisuuden kehittämiseen. Tästä huolimatta teknologia on vielä suhteellisen nuori joten haasteita löytyy niin laitteiston kuten rajapintojen puolelta.

Tarkoituksena tässä insinöörityössä on selvittää persistentin lisätyn todellisuuden mahdollisuuksia ja potentiaalisia rajoitteita. Lisäksi tarkoituksena on toteuttaa proof of concept tyyppinen moninpelattava persistentti lisätyn todellisuuden sovellus. Lopputilanne johon yritetään päästä on että kaikki käyttäjät näkisivat jonkunlaisen tunnisteen toisten pelaajien ympärillä oman mobiililaitteensa kameran läpi. Lisäksi sovellukseen yritetään toteuttaa sisätiloijen navigoinnissa auttava humanoidi hahmo joka ymmärtää fyysistä todellisuutta ja voi auttaa käyttäjiä navigoimaan ympäri ennaltamääriteltyä tilaa.

Sovellus toteutetaan IPad tabletille hyödyntämällä Unity pelimoottoria ja Applen kehittämää ARKit rajapintaa.

2 Lisätty todellisuus

Lisätty todellisuus eli Augmented Reality on tietokoneella generoidun sisällön tuomista oikeaan maailmaan. Yleensä älypuhelimen tai erillisten laitteiden avulla. Lisätty todellisuus eroaa virtuaalisesta todellisuudesta siten että virtuaalisessa todellisuudessa ei ole fyysistä maailmaa mukana mitenkään. Lisätyn todellisuuden käyttötapaukset voi käytännössä jakaa kahteen luokkaan. Viihdekäyttöön tarkoitettuihin sovelluksiin ja teollisuus- ja yrityskäyttöön tarkoitettuihin sovelluksiin.

Viihdekäyttöön suunnatut lisätyn todellisuuden sovellukset toimivat yleensä suoraan älypuhelimella ja eivät vaadi mitään muuta kuin kohtuullisen uuden puhelimen. Muutama vuosi takaperin suosiota kerännyt Pokémon Go on todella hyvä esimerkki lisätyn todellisuuden pelillistämisestä. Pelaamiseen tarvitaan vain android/ios älypuhelin ja

nettiyhteys. Fyysisen lokaation lisäksi joillakin laitteilla pystyi puhelimen kameran avulla tähtäämään ja nappaamaan näitä virtuaalisia

Teollisuuskäytön sovellukset taas vaativat useasti kalliita erillisiä laitteita kuten Microsoftin Hololens älylaseja, Kinect liikkeentunnistuskameraa, tai Magic Leap älylaseja. Nämä laitteet pystyvät tosin saamaan paljon paremman kuvan ympäristöstään kuin pelkkä älypuhelimen kamera, joten niille kehitetyt sovellukset ovat yleisesti todella paljon monipuolisempia. Yksi hyvä esimerkki teollisuuskäytössä olevasta lisätyn todellisuuden sovelluksesta on käytössä Toms River Municipal Utilities Authority nimisellä yrityksellä New Jerseyssä [1]. Yritys tekee julkisten palvelujen huoltotöitä jotka useasti vaativat maanalaisten linjojen tietoja. Aikaisemmin tiedot jouduttiin pyytämään manuaalisesti mm. puhelimen välityksellä, mutta lisätyn todellisuuden sovelluksen käyttöönoton jälkeen tiedot saatiin lähes reaaliajassa. Töiden nopeuttamisen lisäksi tämä myös estää osan potentiaalisista vahingoista, koska tietämys ympäristöstä on paljon kattavampi kuin muilla keinoilla.

2.1 Lisätty todellisuus ennen

Lisätyn todellisuuden alku on vielä epäselvää monelle sillä se vaihtaa muotoaan vielä nykypäivänäkin. Jotkut pitävän Ivan Sutherlandin vuonna 1968 kehittämää Sword Of Damocles virtuaalisen todellisuuden laitetta ensimmäisenä lisätyn todellisuuden laitteena koska siinä käytetty näyttö oli osittain läpinäkyvä. Toiset taas pitävät sen virallisena alkuna sitä kun Hirokazu Kato vuonna 1999 julkaisi ARToolKit kirjastonsa jonka ominaisuudet vastaavat nykypäivän lisätyn todellisuuden sovelluksia [2].

Ensimmäinen suuri kaupallinen lisätyn todellisuuden sovellus tuli kuitenkin vasta vuonna 2008 [16]. Sovellus oli mainos uudelle Mini Cabrio henkilöautolle. Se toimi selaimessa ja vaati webkameran toimiakseen. Käyttäjän mennessä mainoksessa mainitulle internetsivulle ja näyttämällä lehdessä olevaa mainosta kameralle sen päälle ilmestyi malli mainostetusta autosta. Malli seurasi mainossivun liikkeitä ja sitä oli mahdollista pyöritellä ympäriinsä joka mahdollisti mallin tutkimisen kameran läpi monesta eri suunnasta.

2.2 Lisätty todellisuus nykyhetkellä

Nykyhetken suurimmat lisätyn todellisuuden laitteet ovat joko älypuhelimia tai tabletteja. Googlen ARCore toimii Android käyttöjärjestelmän päällä kun taas Applen ARKit toimi vain IOS laitteilla. Molemmat näistä perustuvat kännykän kameran avulla sisällön projisointiin oikeaan maailmaan joko ennaltamääritellyn merkin päälle tai sovelluskehyksen tunnistaman pinnan päälle. Tämän lisäksi mm. Instagram käyttää lisättyä todellisuutta mahdollistaakseen jotkin filttereistään ja parantaakseen osan laatua [15].

Teollisuus ja bisnespuolella toimii Microsoftin kehittämä Hololens lisätyn/sekoitetun todellisuuden älylasit. Hololens lasit eivät tarvitse mitään ulkopuolista laitetta toimiakseen vaan ne ovat käytettävissä ilman ulkoista prosessointiyksikköä. Yksi suurimmista käyttökohteista teollisuudessa ovat opetus ja koulutustarkoitukset. Esimerkiksi AirBus on kehittänyt tuleville miehistönsä jäsenille sovelluksen jolla on mahdollista kouluttaa ilman fyysistä laitteistoa [17] . Vaikka näille laseille löytyy muutamia viihdekäyttöön tarkoitettuja sovelluksia, ei niiden suosio ole korkean hinnan takia juuri noussut keskiverto kuluttajien kesken.

2.3 Lisätty todellisuus tulevaisuudessa

Lisätty todellisuus on kehittynyt viimeisen viiden vuoden aikana ekspotentiaalisesti. Kehitys on vienyt meidät karkeista ja kömpelöistä merkkipohjaisista sovelluksista pistepilviä rakentaviin, ympäröivästä tilasta tiedossa oleviin sovelluksiin. Tämän takia pitkälle tulevaisuuteen katsominen on lisätyn todellisuuden saralla vaikeaa. Varsinkin koneoppimisen vuoksi joka tuo lisätyn todellisuuden sovelluksiin mm. esineiden ja ihmisten tunnistamisominaisuuksia.

Apple julkisti omassa *Apple Worldwide Developers Conference (WWDC) 2019* konfferenssissaan ARKit 3 sovellusrajapinnan joka tuo uutuutena ihmisten siluetin tunnistuksen joka mahdollistaa lisätyn todellisuuden sisällön peittämisen jos kameran ja sisällön välissä on henkilö [14].

2.4 Persistenssi lisätyssä todellisuudessa

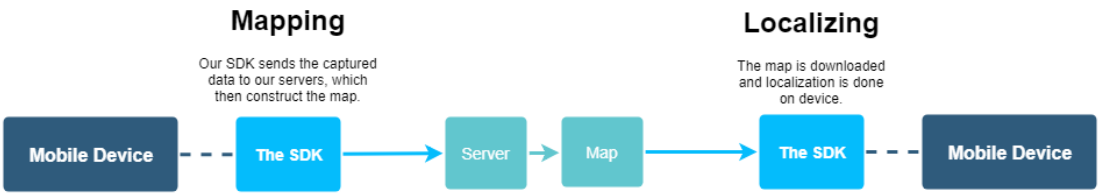
Lisätyn todellisuuden sovellusten pitää jotenkin rakentaa oma käsityksensä ympäröivästä maailmasta. Tämä voi yksinkertaisimmillaan olla vain kuva tai QR-koodi jonka sovellus tunnistaa ja käyttää sitä ankkuripisteenä virtuaalisessa maailmassa. Ongelmaksi tässä muodostuu että sovellus on tällöin rajoitettu vain sille alueelle jossa se voi nähdä kuvan, joten sovelluksen käytettävissä oleva alue on suhteellisen pieni. Lisäksi puhelimen kameran laatu määrittää kuinka kaukana kuvasta voidaan olla jotta kuvan seuraaminen vielä onnistuu.
Edistyneempi tapa pitää lukua maailmasta on pistepilvien avulla. Niihin perustuva seuranta vaatii yleensä ensin alueen kartoituksen laitteella jolla pystyy joko ottamaan yksittäisiä kuvia tai nauhoittamaan videota. Materiaalista tuotetaan pistepilvi ja tuotettua pistepilveä verrataan sovelluksen käynnissä ollessa kameran tuottamaan pistepilveen jonka avulla sovellus rakentaa oman käsityksensä siitä missä kohtaa fyysistä tilaa ollaan.

Projektin aikana testattiin kolmea erilaista pistepilviin perustuvaa persistenssiä joista lopuksi valittiin ARKitin world map johtuen Immersalin Early access tilasta, lisensseistä ja suhteellisen kankeasta ympäristön skannaamisesta. Googlen cloud ankkureissa taas oli liikaa rajotteita. Lisäksi koska kaikki kolme teknologiaa käyttävät hyvin paljon kameraa oli laitevalintana IPad vaikutti myös koska se sisältää ainakin kirjoitushetkellä yhden markkinoiden parhaista kameroista lisätyn todellisuuden sovelluksiin. Kaikki kolme teknologiaa toimivat Unity pelimoottorin päällä.

2.4.1 Immersal AR Cloud SDK

Immersal [3] on suomalainen startup yritys jonka päätuotteena on pilvessä toimiva visuaalisen sijainnin ja kartoituksen pilviratkaisu (Immersal AR Cloud SDK). Projektin aloittamisen kohdalla tämä tuote oli vielä Early Access vaiheessa joten testaus ja mielipiteet perustuvat sillä hetkellä olemassa olevaan teknologiademoon.

Immersal AR Cloud SDK vaatii toimiakseen joko ARKit tai ARcore valmiin laitteen. Ratkaisun teknisestä puolesta ei vielä tässä vaiheessa ole kovin kattavaa dokumentaatiota mutta Immersal developer portaalissa on selitettynä karkeasti sen toiminta [4] .



Kuva 1 Immersal AR Cloud SDK ratkaisun karkea toimintaperiaate.

Ympäristön kartoitus tapahtuu Immersal SDK:ssa videon sijaan kuvia ottamalla. Early access vaiheessa tähän ei ollut vielä mitään erinäistä sovellusta joten se piti itse toteuttaa lähdekoodin avulla. Tämä tosin mahdollisti itse kuvankaappaus logiikan kustomoinnin. Kuvien ottamisen jälkeen kuvat lähetetään rajapinnan kautta Immersalin pilvipalveluun jossa ne analysoidaan ja niistä rakennetaan pistepilvi. Jos pistepilven rakentaminen onnistuu sen voi ladata joko Immersalin developer portaalista tai rajapinnan kautta omiin sovelluksiin. Sovelluksen puolella sitä verrataan ARKitin tapaan ympäristöön ja yritetään sovittaa muistissa oleva malliin.

Jos maailman kartoitus onnistuu hyvin niin maailma pysyy todella hyvin paikoillaan. Paljon paremmin kuin ARcoren ja ARKitin vastaavilla ratkaisuilla. Lisäksi maailman relokalisaatio eli paikoilleen asettelu sujuu nopeammin.

2.4.2 ARKit World Map

ARKit käyttää *visual-inertial odometry* (VIO) nimistä tekniikkaa maailman tunnistamisessa [5]. VIO rakentuu ARKitin tapauksessa kahdesta eri osasta: mobiililaitteiden antureista ja kuvantunnistuksesta. Näistä kuitenkin ensisijainen tapa on kuvantunnistus, jolla voidaan kerätä videokuvasta ominaisuuskohtia (Feature Point). Nämä ominaisuuskohdat kartoittuvat mobiililaitteen kameran videon kautta ympäristön uniikkeihin kohtiin kuten pöydänkulmiin, nurkkiin, tai muuten uniikkeihin muotoihin. Ominaisuuskohdat voidaan ensimmäisen kartoituskerran jälkeen tallentaa ja seuraavalla käynnistyskerralla lukea uudestaan. Näitä kutsutaan ARKitissä world mapeiksi [6]. Seuraavalla käynnistyskerralla tallennettu world map voidaan lukea muistiin ja verrata kameran reaaliaikaisiin ominaisuuspisteisiin. Tätä avustaa laitteiden kiihtyvyysantureista saatu data. Koska kuvan pisteiden tunnistaminen ei onnistu aina jokaisella kerralla käyttää ARKit kiihtyvyysantureista saatua dataa arvioidakseen mihin

kohtaan kameraa on liikutettu. Tästä kuitenkin kertyy ajan mittaan kasvava virhemarginaali joten sitä ei voi käyttää pääasiallisena tapana arvioida käyttäjän asentoa virtuaalisessa maailmassa.

2.4.3 ARCore world anchor

Googlen ARCore world anchorit toimivat melkein samalla periaatteella kuin Applen world mapit. Skannataan alueen ominaisuuskohdat ja laitetaan ne talteen. Se kuitenkin eroaa world mapeista siinä että rakennetut pistepilvet ladataan googlen palvelimille ja jaetaan sieltä suoraan rajapinnan kautta muille käyttäjille. Tämä helpottaa sovellusten tuottamista koska verrattava pilvi on automaattisesti sama käyttäjien kesken ja sitä voidaan myös jakaa lennosta. World anchorien käyttö vaatii myös rekisteröitymisen ja rajapinnan käyttöavaimen googlelta. Rajapinnan käyttö on myös maksullista käyttömäärän ylittäessä tietyn rajan.

3 Persistentti AR moninpelisovellus sisätiloissa navigointiin

Tarkoituksena on toteuttaa proof of concept tyylinen monen pelaajan lisätyn todellisuuden sovellus käyttäen Unity pelimoottoria. Sovelluksessa pelaajat näkevät saman maailman, toisensa, ja sisätilojen navigoinnissa auttavan virtuaalisen avustajan mobiililaittensa kameran läpi. Sovellus toteutetaan IPad pro tabletille mutta käytännössä sen pitäisi toimia millä tahansa ARKittiä tukevalla laitteella.

Unity projekti pitää sisällään kaksi eri näkymää (scene). Bootstrap näkymästä jonka työnä on kerätä sovelluksen käyttävän käyttäjän nimi, alustaa moninpelijärjestelmä ja ladata itse päänäkymä. Päänäkymä taas pitää sisällään kaiken sovelluksen toimintalogiikan kuten maailman relokalisoinnin, pelaajien synkronoinnin sekä virtuaalisen avustajan tekoälyn.

3.1 Ympäristön luominen

Ympäristön luominen aloitetaan ensin kartoittamalla haluttu tila johon sovelluksen halutaan sijoittuvan. Tähän ei ole mitään valmista ratkaisua vaan Unityn mukana tuleva

ARKit lisännäinen tarjoaa valmiita skripti joilla voi rakentaa itselleen sopivan kartoitussovelluksen. Oletuksena esimerkkiskripti tallentaa pistepilven vain serialisoituna world map objektina, mutta koska projektin myöhemmässä vaiheessa pistepilvi pitää saada myös 3D-mallinnusohjelmaan, pitää pisteiden keräyksen aikana ne tallentaa myös karteesisen koordinaatiston koordinaatteina (XYZ muoto). Kun pistepilvi on rakennettu se voidaan ladata IPadilta koordinaatit sisältävän tiedoston kanssa ja viedä projektikansioon.

3.1.1 Kartoitus

Pistepilven luomiseen fyysisestä tilasta ei ole valmista työkalua, se pitää rakentaa itse lähdekoodista. Tästä on onneksi muutama hyvä esimerkki ja oman työkalun rakennus ei ole mahdottoman vaikeata. ARKit unitylisännäisen mukana tulee esimerkki joka rakennettuna visualisoi luotua pistepilveä. Tämän esimerkin päälle oli helppo rakentaa oma työkalu tilojen kartoitusta varten. Kun tiloja on kartoitettu tarpeeksi, voi sen hetkisen world map objectin serialisoida binäärimuotoon seuraavalla tavalla:

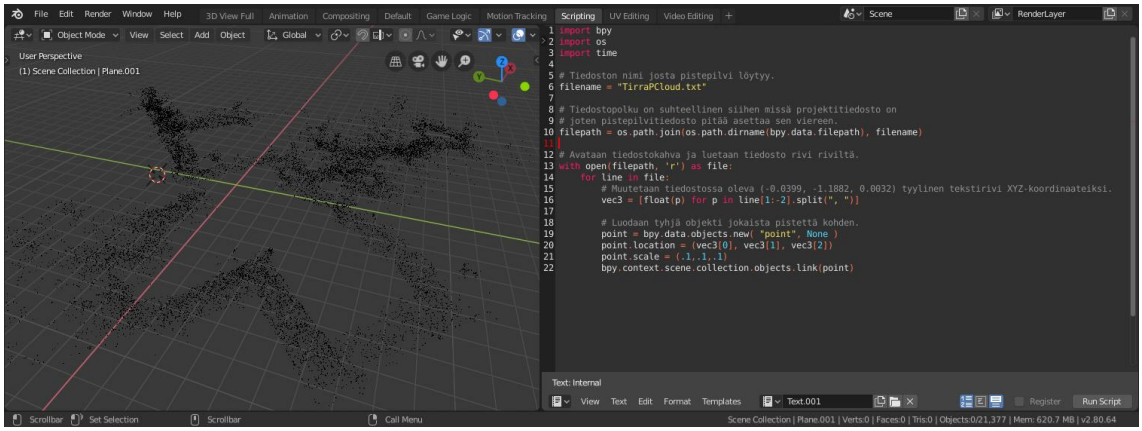
```
public void SaveWorldMap(){
        // GetARWorldMapAsync suorittaa parametrinä annetun funktion sen hetkistä world mappia käyttäen.
        ARSubsystemManager.sessionSubsystem.GetARWorldMapAsync(SaveMap);
}


 private void SaveMap(ARWorldMapRequestStatus status, ARWorldMap map){

        // Kartan tallennuslokaatio. Kartan perään lisätään sen hetken päivämäärä ja aika niiden
        erottelemiseksi.
        string path = Path.Combine(Application.persistentDataPath, "ARWorldMap" +
        DateTime.Now.ToShortTimeString());

        // World mapin serialisointi binäärimuotoon.
        Unity.Collections.NativeArray<byte> data = map.Serialize(Unity.Collections.Allocator.Temp);

        // Datan kirjoitus tiedostoon.
        FileStream file = File.Open(path, FileMode.Create);
        using (BinaryWriter writer = new BinaryWriter(file)) {
            writer.Write(data.ToArray());
        }

        // Resurssien vapauttaminen
        data.Dispose();
        map.Dispose();
}
```

Pisteiden tallentaminen XYZ-muodossa oli sen sijaan hieman haastavampaa. ARWorldMap objektista ei ole mitään suoraa tapaa saada pisteiden sijainteja koordinaatteina, joten tähän piti kehittää oma logiikka hoitamaan niiden keräys. Unity tarjoaa onneksi ARFoundation lisännäisen joka antaa työkaluja AR-projektien

toteuttamisen helpottamiseksi ja samalla mahdollistaa ARCore ja ARKit sovellusten toteuttamisen samassa projektissa. Sen tarjoama **ARPointCloudManager** luokka mahdollistaa pistepilven rakentumisen seuraamisen ja pisteiden tallentamisen sitä mukaa kun pistepilvi päivittyy. Valitettavasti pisteitä ei ole mitenkään eroteltu toisistaan joten samassa kohtaa oleva piste saattaa joutua tallennetuksi monta kertaa. Tätä yritettiin estää tallentamalla pisteet vain tietyin väliajoin mutta siitä huolimatta duplikaatteja ei saatu ihan kokonaan vältettyä.

3.1.2 Mallintaminen

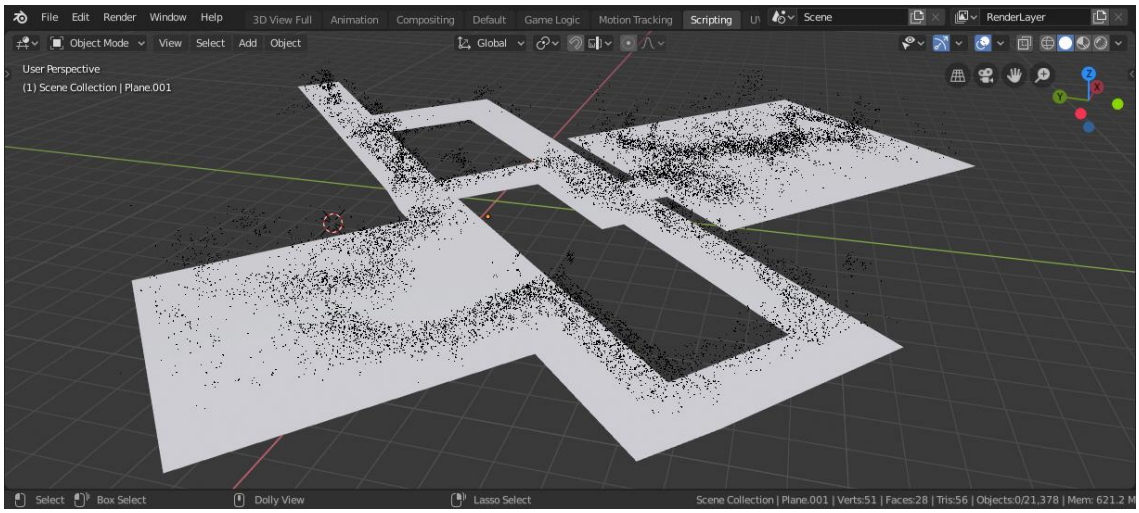
Myöhemmässä vaiheessa virtuaalinen avustaja tarvitsee jonkunlaisen käsityksen ympäristö muodosta. Tämän toteuttaminen onnistuu todella helposti aikasemmassa vaiheessa kerätyn pistepilven pohjalta. Koska pistepilvi on tekstimuodossa sitä ei ihan suoraan saa mihinkään mallinnusohjelmaan. Tämäkään ei sinällään ole ongelma sillä melkein kaikki suurimmat mallinnusohjelmat tukevat jotakin skriptikieltä, joilla pystyy tekstitiedoston muuttamaan pistepilven tyyliseksi objektiksi. Tässä projektissa siihen käytettiin Blenderiä joka käyttää skriptikielenään pythonia.



Kuva 2 Pistepilvi Blenderissä ja sen luonut python skripti.

Ennen tiedoston viemistä Blenderiin sitä piti karsia huomattavasti. Alkuperäisessä pistepilvessä oli yli 150k pistettä ja sen prosessoimiseen olisi mennyt todella pitkä aika. Lisäksi koska pisteet ovat yksittäisiä objekteja niiden liikuttelu on todella raskasta, jopa 20k pisteeseen rajatussa pilvessä. Kun pisteet on visualisoitu Blenderiin on niiden

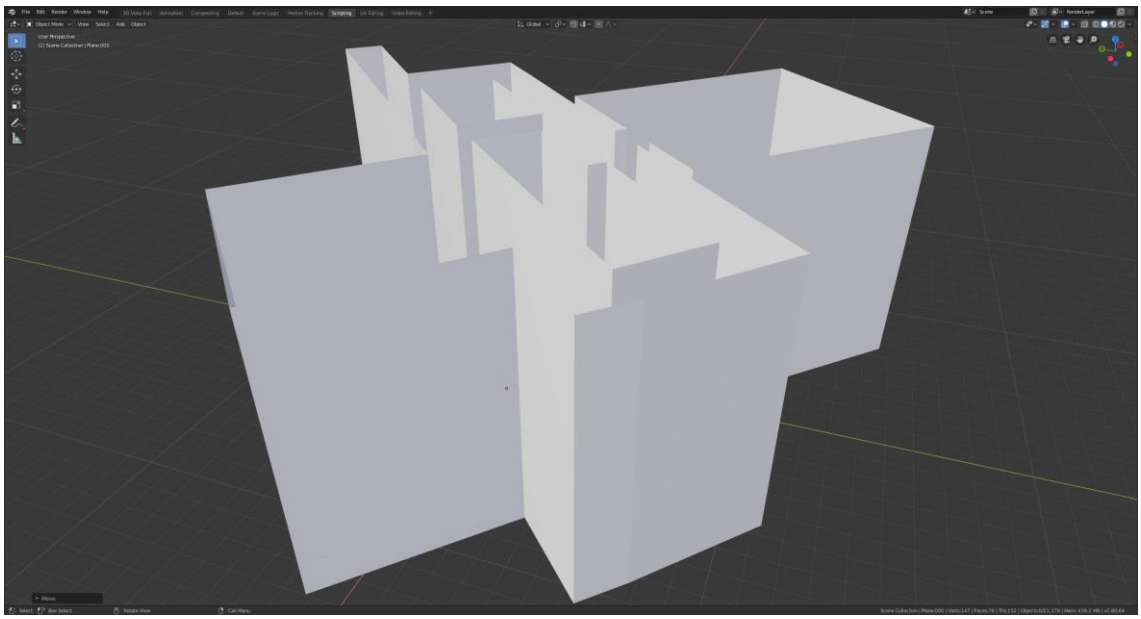
pohjalta helppo rakentaa tilan pohjapiirros. Aivan täydellistä siitä ei saa, johtuen pistepilven vääristymisestä tilan ääripäissä mutta lopputulos on silti erittäin tyydyttävä.



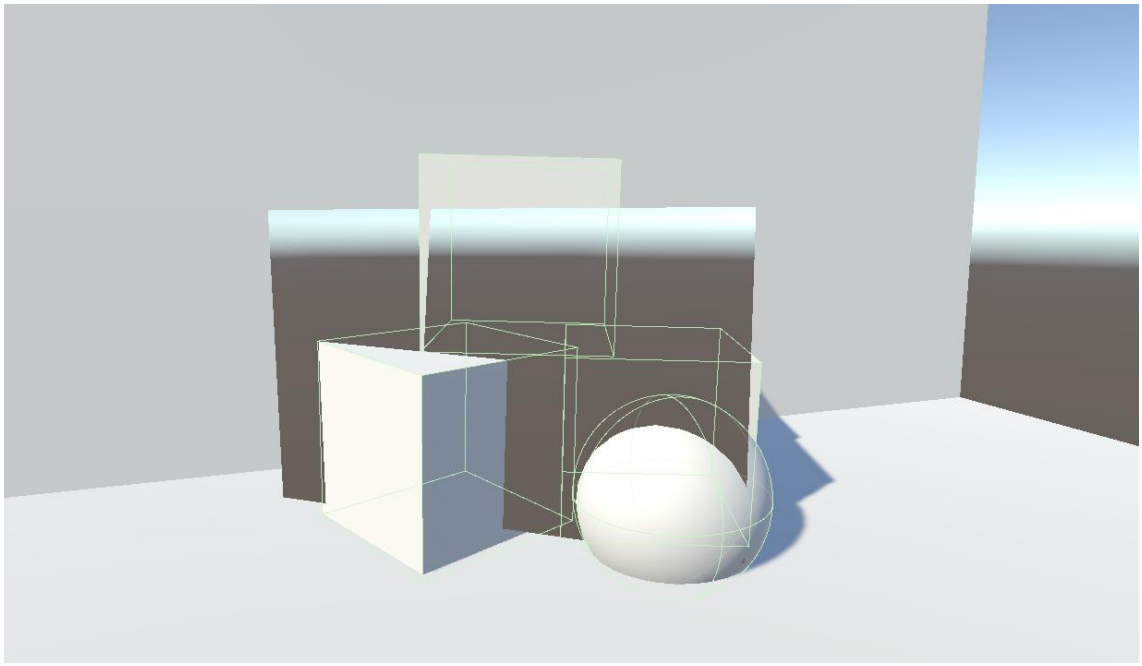
Kuva 3 Pistepilven päälle mallinnettu tilan pohjapiirros

3.1.3 Peiteseinät

Koska oikeassa elämässä ei näe seinien läpi niin on myös kohtuullista ettei lisätyn todellisuuden sisältö näy oikeiden seinien lävitse. Ratkaisu tähän oli mallintaa myös tilan seinät pohjapiirroksen päälle ja asettaa niihin shader ohjelma joka vain kirjoittaa Z-bufferiin ennen muiden ohjektien piirtämistä piirtämättä ruudulle mitään. Tällä tavalla kaikki seinien takana olevat objektit saadaan piilotettua. Koska tilaa on todella vaikea mallintaa täsmälleen saman muotoiseksi ja kokoiseksi kun fyysistä tilaa eivät seinät ole täsmälleen oikeilla paikoilla ja tästä syystä välillä saattaa nähdä sisältöä minkä pitäisi olla kulman takana.



Kuva 4 Peiteseinät rakennettu pohjapiirroksen päälle



Kuva 5 Esimerkki syvyysbufferiin kirjoittavasta shader ohjelmasta. Neliskulmaisen peitepinnan takana näkyy sen osittain peittämien objektien ääriviivat.

Syvyystestausta käytetään normaalisti kaikissa piirettävissä objekteissa jotta jo piirrettyä pikseliä ei piirrettäisi kovin monta kertaa uudestaan. Kun objektia aletaan piirtämään verrataan sen kaikkia pikseleitä syvyysbufferiin. Jos syvyysbufferissa oleva arvo on lähempänä kameraa kuin sen hetkisen pikselin niin piirto keskeytetään [7].

3.2 Moninpeliosuus

Moninpelien toteuttamiseen unityssä on muutama varteenotettava vaihtoehto. Tässä projektissa käytetään Photon Unity Networking moninpelimoottoria, mutta tutkinnan alla oli myös kaksi muuta potentiaalista vaihtoehtoa:

- Unityn tarjoama Unet rajapinta. Tämä on kuitenkin vanhentunutta teknologiaa ja se ollaan tulevaisuudessa poistamassa käytöstä [8] joten siihen ei paneuduta tässä kohtaa enempää.

- DarkRift2 [9]. Darkrift2 on kehys omien moninpelilogiikoiden toteuttamiseen. Se ei itse suoranaisesti hoida muuta kuin datan välityksen ja antaa käyttäjän määrittää molemmissa päissä (Palvelimen ja käyttäjän) tapauskohtaisesti miten loogiset kokonaisuudet kuten huoneet ja pelilogiikka toteutetaan. Palvelinpuoli toimii .dll pluginien kautta ja ne ovat ihan erillään unity puolen toteutuksesta.

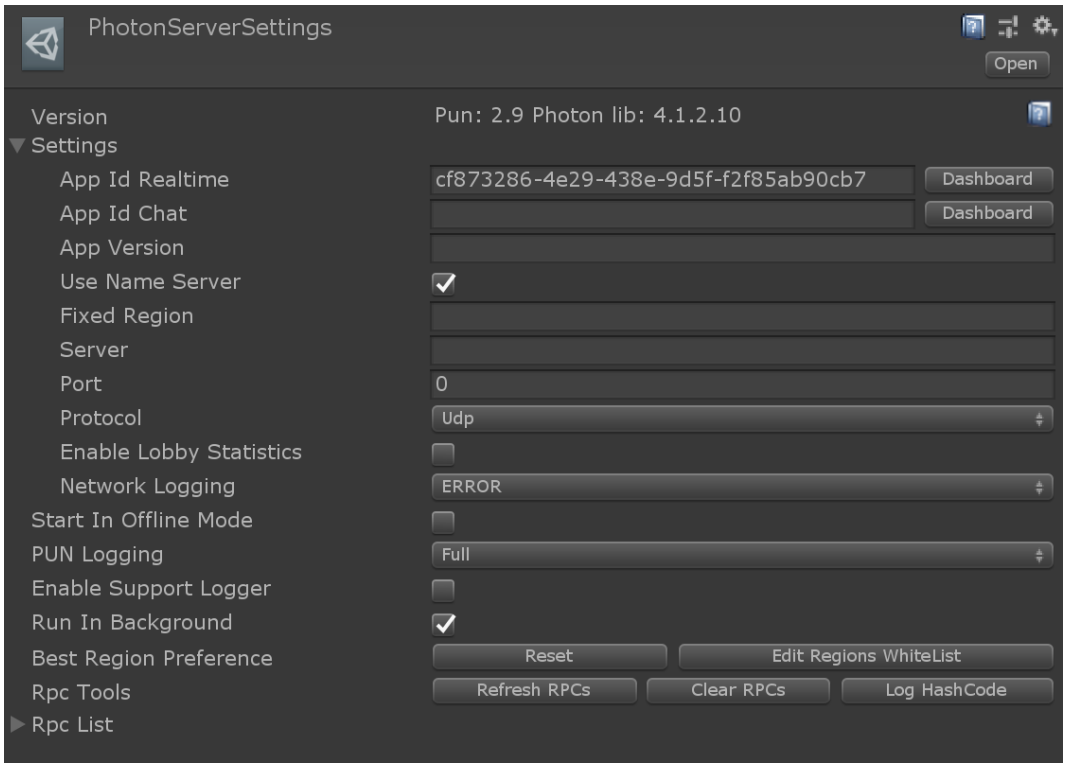
- Photon Unity Networking (PUN) [10]. PUN tarjoaa täyellisen ratkaisun moninpelien toteuttamiselle. Se hoitaa kaiken datan lähettämisestä huoneiden luomiseen automaattisesti taustalla yksinkertaisen rajapinnan kautta.

3.2.1 Photon Unity Networking

Palvelimilla ei PUN:ssa varsinaisesti mitään muuta virkaa kuin huoneiden ylläpito ja datan välitys. Kaikki prosessointi hoidetaan käyttäjien omilla koneilla ns. Master client tyyppisellä toteutuksella, missä yksi pelaajista hoitaa aina palvelimen virkaa. Tämä mahdollistaa palvelinkoodin kirjoittamisen normaalin pelikoodin yhteyteen mikä selkeyttää ja nopeuttaa pelien toteutusta. PUN palvelimien ylläpitoa ei myöskään tarvitse hoitaa itse, vaan Photon tarjoaa mahdollisuuden käyttää heidän ylläpitämiä palvelimiaan joiden ilmaisversiossa rajoituksena on 20 samanaikaisen käyttäjän raja.

3.2.2 Moninpelilogiikan alustus

Jotta PUN osaisi ohjata liikenteen oikeiden käyttäjien välillä pitää palvelimelle yhdistäminen ja huoneesen liittyminen hoitaa ensin. Yhteyden asetukset ovat tallennettuna PhotonServerSettings ScriptableObjectiin.



Kuva 6 Kaikki moninpeliyhteyteen liittyvät tiedot löytyvät samasta tiedostosta.

Yleisesti tarvittaisiin palvelimen domain tai IP-osoite mutta jos käytössä on photonin ylläpitämät palvelimet tarvitaan vain app-id jonka sai hankittua photonin verkkosivuilta rekisteröitymisen jälkeen.
Photon tarjoaa mahdollisuuden hoitaa itse näkymien hallinnoinnin mutta helpoin tapa on antaa photonin aina synkronoida kaikkien käyttäjien näkymä master clientin näkymään. Kokonaisuudessaan yhteyden alustus tapahtuu seuraavalla tavalla.

```
public class TirraARLauncher : MonoBehaviourPunCallbacks
{
    void Start()
    {
        PhotonNetwork.AutomaticallySyncScene = true;
    }

    // Tätä metodia kutsutaan tämän skriptin ulkopuolelta kun halutaan aloittaa
    palvelimeen yhdistäminen.
    public void Connect()
    {
        PhotonNetwork.ConnectUsingSettings();
    }

    // Kun palvelimeen on yhdistetty Photon kutsuu tätä metodia automaattisesti jos
    luokka perii MonoBehaviourPunCallbacks luokan.
    public override void OnConnectedToMaster()
    {
        // Liitytään ensimmäiseen vapaaseen huoneeseen.
        // Koska emme ole määritelleet huoneille mitään rajoituksia kaikki käyttäjät
       liittyvät aina samaan huoneeseen.
        PhotonNetwork.JoinRandomRoom();
    }

     // Jos huoneeseen liittyminen epäonnistuu tarkoittaa se sitä että huoneita ei ole.
     Tässä tapauksessa haluamme luoda uuden huoneen. Huoneen luonnin jälkeen PUN
     automaattisesti liittyy vastaluotuun huoneeseen.
    public override void OnJoinRandomFailed(short returnCode, string message)
    {
        // MaxPlayers = 0 tarkoittaa loputonta määrää käyttäjiä.
        PhotonNetwork.CreateRoom(null, new RoomOptions { MaxPlayers = 0 });
    }

    // Kun huoneeseen liittyminen onnistuu PUN kutsuu tämän metodin.
    public override void OnJoinedRoom()
    {
       // Näkymä 0 on tämän sovelluksen tapauksessa aina bootstrap näkymä.
       // Näkymä 1 on päänäkymä joten jos huoneessa on sinne liittyessä yksi käyttäjä
       meinaa se että palvelimen näkymä pitää vaihtaa.
       // Tämä tapahtuu automaattisesti kaikille tämän jälkeen palvelimelle liittyville
       käyttäjille
        if (PhotonNetwork.CurrentRoom.PlayerCount == 1) {
            PhotonNetwork.LoadLevel(1);
        }
    }

}
```

3.3 Maailman relokalisointi

ARKit ja ARFoundation yhdessä tekevät maailman relokalisoinnista melkein automaattista. Näkymään missä relokalisointi tapahtuu piti lisätä **ARSessionOrigin** ja **ARSession** komponentit. Niiden avulla ARKit osaa asettaa pelaajan kameran oikeaan paikkaan virtuaaliympäristössä. WorldMap objektin lataus pitää kuitenkin hoitaa itse. Tämä

tapahtuu koodinpätkällä joka ajetaan heti päänäkymän käynnistyessä. Ensimmäinen askel on ladata binäärimuodossa oleva tiedosto muistiin ja muuttaa se tavumuotoon.

```
ARWorldMap worldMap;
List<byte> worldMapBytes = new List<byte>();

using (MemoryStream stream = new MemoryStream(worldmap.bytes)) {
    int bytesPerFrame = 1024 * 10;
    long bytesRemaining = stream.Length;

        // Muutetaan raaka binääridata listaksi tavuja.
    using (BinaryReader binaryReader = new BinaryReader(stream)) {
        while (bytesRemaining > 0) {
            var bytes = binaryReader.ReadBytes(bytesPerFrame);
            worldMapBytes.AddRange(bytes);
            bytesRemaining -= bytesPerFrame;
            yield return null;
        }
    }
}
```

Kun tiedosto on muistissa muutetaan se unityn NativeArray mutoon. NativeArray puskuriluokka on tarkoitettu datan nopeaan käsittelyyn ja se mahdollistaa jakamisen vapaammin muiden prosessien tai natiivi/alemman abstraktiotason kielien kanssa [11]. Tämä vaihe oli pakko tehdä koska unityn ARKit lisännäinen keskustelee natiivin ARKit toteutuksen kanssa.

```
NativeArray<byte> worldMapData = new NativeArray<byte>(worldMapBytes.Count,
Allocator.Temp);
worldMapData.CopyFrom(worldMapBytes.ToArray());
```

Tässä vaiheessa on jäljellä enään kartan luominen datasta. Tähänkin ARKit tarjoaa valmiin funktion.

```
using (NativeArray<byte> worldMapData = new NativeArray<byte>(worldMapBytes.Count,
Allocator.Temp)) {
    worldMapData.CopyFrom(worldMapBytes.ToArray());

    // Yritetään kääntää binäärimuodossa oleva data ARWorldMap objektiksi.
    if (ARWorldMap.TryDeserialize(worldMapData, out worldMap)) {
        Debug.Log("ARWorldMap succesfully deserialized.");
    } else {
        Debug.LogError("Failed to deserialize ARWorldMap");
    }

}
```

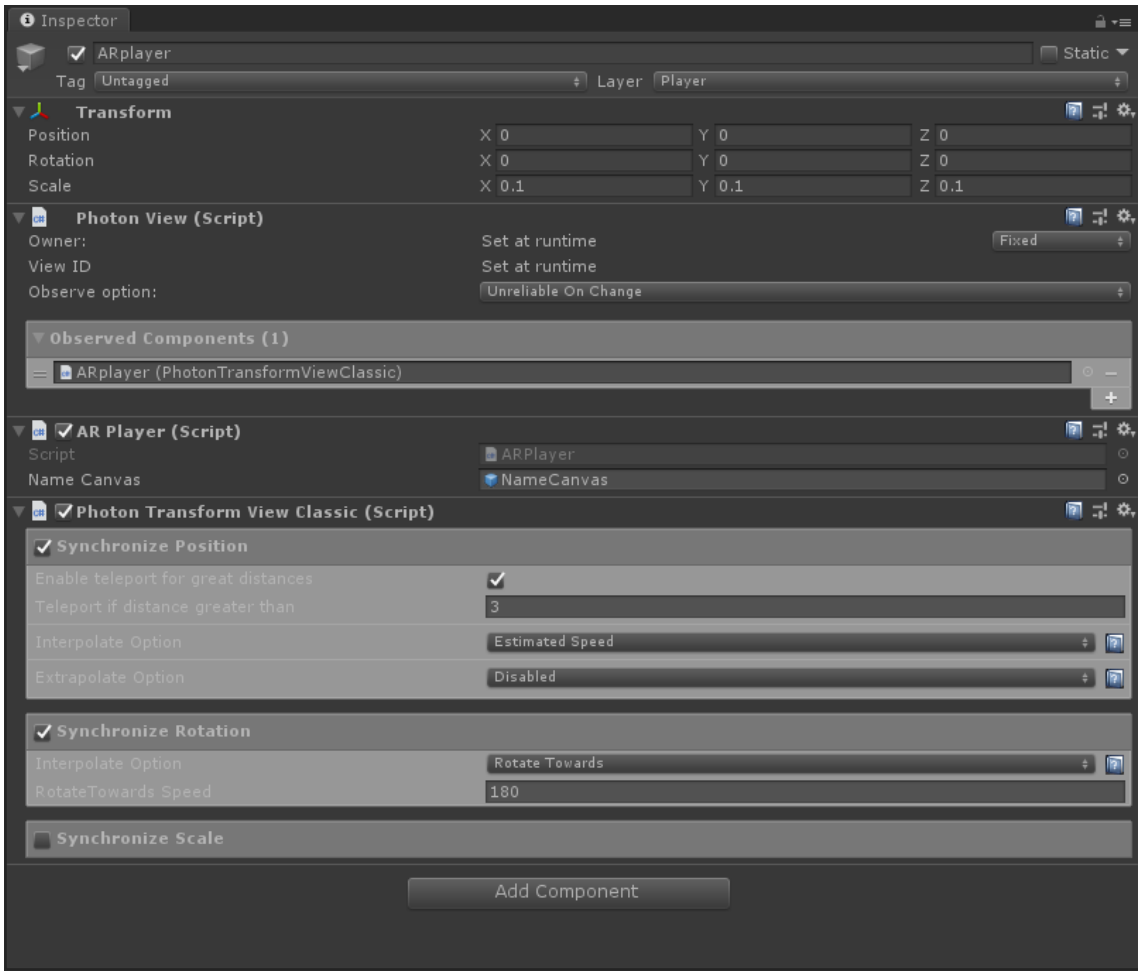
Kun kartta on ladattu muistiin, yrittää ARKit automaattisesti verrata sitä sillä hetkellä generoituun pistepilveen ja sijoittaa käyttäjän juuri siihen kohtaan maailmassa.

Useassa tapauksessa kun sovelluksen käynnistää saa ympäristöä skannata tovin ennen kuin ARKit keksii missä päin pilveä käyttäjä on. Tätä voidaan kuitenkin nopeuttaa yrittämällä skannata joitakin kohtia missä tiedetään pistepilven olevan tiheä kuten kirjahyllyjä tai suuria koriste-esineitä.

3.4 Käyttäjien synkronointi

Käyttäjälle pitää vielä määrittää objekti joka kuvaa/edustaa pelaajaa virtuaalimaailmassa. Tämän sovelluksen tapauksessa tämän objektin piti pystyä vain pitämään lukua siitä missä käyttäjän laite milläkin hetkellä on keskellä pistepilveä. Kun ARKit onnistuu keksimään missä kohtaa se on ladattua WorldMapin pistepilveä se sijoittaa näkymässä määritellyn pääkameran siihen kohtaan missä se ajattelee olevansa. Tämä tieto voitiin

välittää toisille pelaajille heti kun he olivat myös löytäneet itsensä pistepilven jostakin kohtaa.



Kuva 7 Pelaajaobjektin koostumus

Pelaajaobjektin kolme komponenttia ovat:

- **Photon view**. Tämä komponentti tarvitaan kaikilta objekteilta/sisällöiltä jotka halutaan laskettavan mukaan verkon yli synkronointiin. Se ei yksinään tee mitään vaan hoitaa muiden komponenttien synkronointitarpeet. Vaikka kyseinen komponentti on jokaisen pelaajan sovelluksessa sen toiminta vaihtelee riippuen kuka komponentin omistaa. Jos komponentin omistaa käyttäjä joka ajaa kyseistä sovellusta se lähettää dataa. Jos taas komponentin omistaja ei ole sovelluksen ajaja se vastaanottaa dataa sen omistajalta. Jos meillä olisi kilpailullinen peli niin tämä ei olisi haluttua koska jokainen käyttäjä luottaa siihen että muut käyttäjät eivät huijaa.

  Observer components listaan pitää laittaa kaikki skriptit ja komponentit jotka halutaan kyseisen photon view komponentin synkroinoivan muiden pelajaien kanssa.

- **ARPlayer**. Alustaa kaikki tarpeelliset muuttujat ja hoitaa nimikyltin piilottamisen käyttäjältä jos kyseessä on omaa pelaajaa kuvaava objekti.
- **Photon Transform View Classic**. Hoitaa pelaajan sijainnin / rotaation / skaalan synkronoinnin. Tämän voi tehdä itse mutta yksinkertaisimmissa tapauksissa tämä avustuskomponentti riittää. Se pitää myös muistaa rekisteröidä PhotonView komponenttiin jotta se päivittyy.

3.5 Virtuaalinen avustaja sisätilojen navigointiin

Projektin mietittiin jotain sisältöä mitä lisätyn todellisuuden sovellukseen voisi lisätä. Haluttiin että sovellus joka toteutetaan saa jotain lisäarvoa siitä, että siinä käytetään lisättyä todellisuutta hyväksi. Lopulta idea virtuaalisesta avustajasta syntyi. Ongelmaksi kuitenkin tässä muodostui se asia että jos virtuaalisen avustajan halutaan jotenkin olevan hyödyllinen ja reagoivan ympäristöön, niin se tarvitsee hyvän käsitysen ympärillä olevasta maailmasta. Koska lisätyn todellisuuden teknologiat ovat vielä todella alkuajoilla niin tarpeeksi korkeatasoisen mallin ulkomaailmasta saamiseksi tarvitaan kalliita laitteistoja. Tähän kuitenkin keksittiin ratkaisuksi sisätilojen navigoinnissa auttava avustaja. Se tarvitsi vain karkean kuvan ympäristöstä ja tämä saatiin kartoittamalla ja mallintamalla ympäristö etukäteen.

3.5.1 Äärellinen automaatti

Avustajan ydinlogiikka toteutettiin käyttämällä äärellistä automaattia (Statemachine). Automaatit ovat yksi suosituimmista tavoista tehdä yksinkertaisia tekoälyjä koska jokaisen tilan toiminnallisuus on eristetty omaan funktioonsa. Lisäksi tilojen välisten yhteyksien määrittely helpottuu.

Äärellisestä automaatista on vielä kehittyneempi versio nimeltään pinoautomaatti (pushdown automata) joka pitää lukua tilojen vaihdosta puskemalla ne pinoon. Pinon päällimmäinen tila on aina sen hetken aktiivinen tila. Tämä on hyödyllistä monimutkaisemmissa tilanteissa kuten videopeleissä jos hahmolla on väliaikainen tila kuten lyönti- tai ampumisanimaatio. Sen sijaan että jokaiselle hahmon yhdistelmälle kuten liikkeestä lyömiselle, paikallaan lyömiselle tai ilmasta lyömiselle pitäisi tehdä omat linkityksensä voi väliaikaisen animaation vain puskea pinoon ja sen valmistuessa poistaa pinon päältä (pop) jolloin aikaisempi animaatio aloittaa taas toimintansa. Tässä

projektissa ei kuitenkaan tarvittu näin monimutkaista logiikkaa joten pelkkä äärellinen automaatti riitti logiikan toteuttamiseen.

```
public delegate void StateFunc();
private Dictionary<T, State> m_States = new Dictionary<T, State>();

public void Add(T id, StateFunc enter, StateFunc update, StateFunc leave) {
    m_States.Add(id, new State(id, enter, update, leave));
}
```

Automaattiin lisätään tiloja joille määritellään jokin geneerinen tyyppi sitä luodessa. Tämä tyyppi toimii avaimena jolla vaihdetaan tilojen välillä. Lisäksi tietylle tilalle pitää määritellä kolme parametritöntä funktiota Enter, Update, ja Leave. Enter funktiota kutsutaan kun johonkin tilaan vaihdetaan ja Leave funktiota kun siitä poistutaan. Nämä auttavat alustamaan tietyt tilat jos niiden pyörittäminen vaatii tiettyjä tietoja. Update funktiota taas ajetaan samaan tahtiin Unityn Update funktion kanssa eli kerran jokaisen ruudun piirron yhteydessä. Se pitää sisällään kaiken toimintalogiikan jota juuri siinä automaatin tilassa käytetään. Automaattia hyväksikäyttämällä voidaan helposti rakentaa monimutkaisia toiminnallisuuksia ilman että koodista tulee monta tuhatta riviä pitkä kaikkien rajatapausten välttämiseksi. Kaikkien tilojen toimintalogiikka on eristettynä omaan funktioon jonka ansiosta rajatapauksista ei tarvitse huolehtia kuin omassa tilassa.

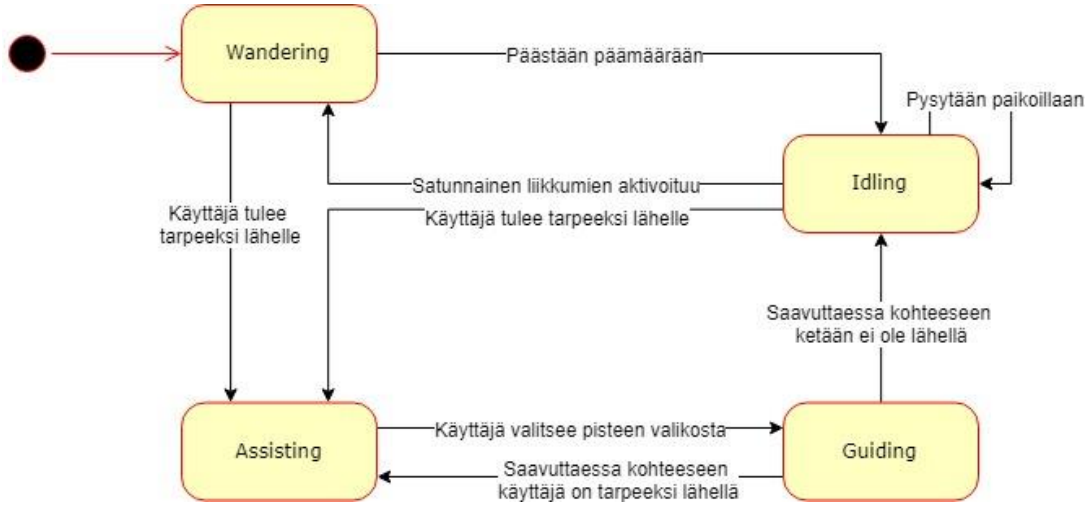
Avustajan tilan määrittelemiseen luotiin *enum* ActorState joka määritteli kaikki mahdolliset tilat johin avustaja pystyi vaihtamaan.

```
public enum ActorState {
    Wandering, Idling, Assisting, Guiding
}
```

Tilojen määritteleminen aloitettiin tarpeista. Avustajan pitää pystyä antamaan neuvoa, seisomaan joutilaana paikoillaa jos kukaan ei ole lähettyvillä, ohjaamaan tiettyihin kohteisiin ympäri tilaa, ja ihmismäisyyden luomiseksi itsenäisesti pyöriä ympäristössä. Tilojen toiminnallisuudet tiivistettynä:

- **Wandering**. Tähän tilaan vaihdettaessa avustaja etsii ympäriltään satunnaisen pisteen mihin se voi kävellä ja vaeltaa siihen. Saavuttaessa päämäärään se vaihtaa *Idling* tilaan. Tämän tilan voi keskeyttää kävelemällä tarpeeksi lähelle avuastajaa jolloin *Assisting* tila menee päälle.

- **Idling.** Avustaja seisoo paikoillaan ja tietyin väliajoin heittää kolikkoa pysyykö se Idling tilassa vai vaihtaako se Wandering tilaan. Tämä tila vaihtuu myös *Assisting* tilaan jos joku käyttäjä kävelee tarpeeksi lähelle avustajaa.
- **Assisting.** Tämä tila laukaistaan silloin kun jokin sovelluksen käyttäjistä kävelee tarpeeksi lähelle avustajaa. Kun tilaan tullaan se avaa valikon avustajan viereen josta voi valita mihin ennaltavalittuun pisteeseen halutaan avustajan opastavan/kävelevän. *Assisting* tila vaihtuu *Idling* tilaksi siinä vaiheessa jos sen ympärillä ei ole käyttäjiä.
- **Guiding.** Tämä tila laukaistaan kun avustajan halutaan opastavan johonkin. Tätä tilaa ei pysäytä mikään muu kuin päämäärään pääseminen. Kun päämäärään päästään avustaja vaihtaa joko *Assisting* tai *Idling* tiloihin riippuen onko joku sovelluksen käyttäjistä sen vieressä kun päämäärään saavutaan.



Kuva 8 UML-kaavio avustajan teköälystä

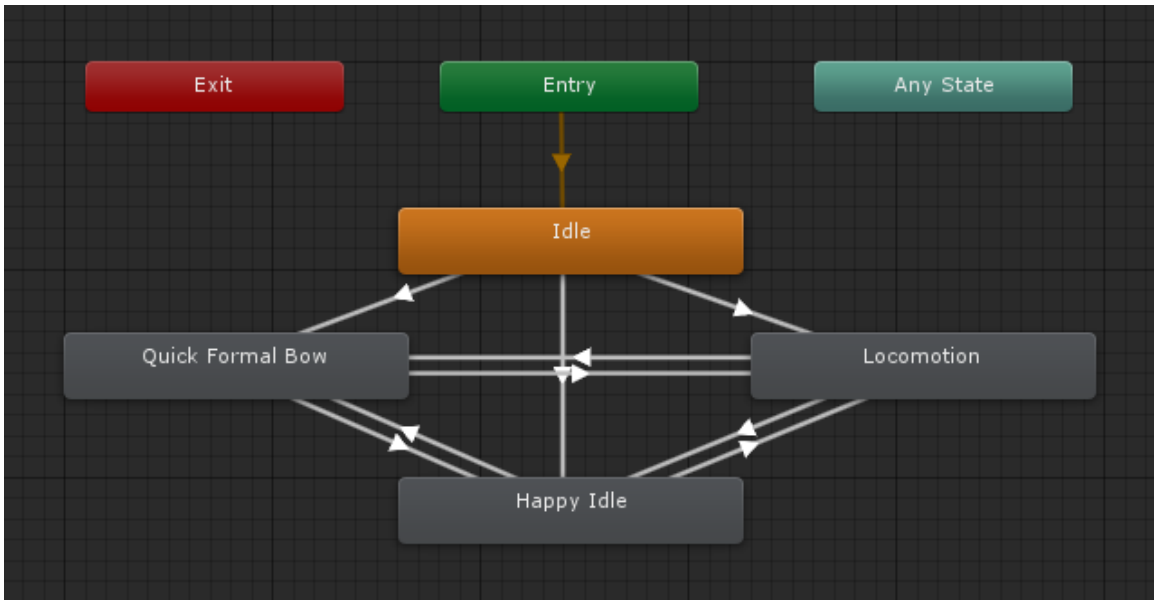
3.5.2 Läheisyyden tunnistaminen

Läheisyyden tunnistus avustajalla tapahtui luomalla sen ympärille ympyrän muotoinen laukaisija (collider) osoittamaan sen henkilökohtaista aluetta. Se asetettiin avustajan hierarkiassa pääobjektin alle. Kun joku pelaajaobjektiksi merkitty esine tulee henkilökohtaisen alueen sisälle tai poistuu siitä se lähettää viestin kaikille hierarkiassa ylempänä oleville objekteille. Viesti sisältää viitteen pelaajaan joka laukaisi tapahtuman ja kuinka monta pelaajaa tapahtuman jälkeen avustajan henkilökohtaisessa tilassa vielä on.

3.5.3 Mallit ja animaatiot

Avustajan animaatioita ja mallia ei yritetty alkaa kasaamaan itse vaan ne ladattiin Adoben tarjoamasta mixamo.com palvelusta. Mixamon mallit ovat ilmaisia ja niitä voi käyttää ilman lisenssimaksuja jopa kaupallisissa projekteissa [12]. Avustajan avatariksi eli malliksi valittiin ritari johtuen siitä että skannatulla alueella oli ritarin haarniska.

Avustajan animaatiot koostuvat kävely ja kääntymisanimaatioiden lisäksi myös tervehdys- ja toimettomuusanimaatioista.



Kuva 9 Avustajan animaatioiden statemachine ja sen transitiot

Animaatioita ohjataan avustajan nopeuden ja satunnaisen numeron avulla suoraan koodista. Animaattori ei itsessään tee mitään muuta kuin jatkuvasti pyörittää animaatiota joka sillä tällä hetkellä on parametrien mukaan valittuna. Idle ja Happy Idle animaatiot pyörivät vain silloin kun *Idling* tilassa ja lähellä ei ole ketään. Quick Formal bow taas ajetaan kun avustaja siirtyy tai poistuu *Assisting* tilasta. Locomotion animaatio on taas käytössä aina jos avustaja ei ole paikoillaan. Se koostuu eteenpäin kävely ja kääntymisanimaatioista joita sulautetaan avustajan etenemisnopeuden ja kääntymisnopeuden perusteella.

3.5.4 Opastepisteet

Opastepisteet toteutettiin luomalla tyhjiä peliobjekteja ja liittämällä niihin skripti joka pelin käynnistyessä lisäsi staattiseen listaan objektin nimen ja lokaation sisältävän struktin.

```
public class GuideSpot : MonoBehaviour {

    // Staattinen lista johon kerätään kaikkien GuideSpot skriptien nimet ja
    lokaatiot.
    public static List<GuideSpotDTO> GuideSpots = new List<GuideSpotDTO>();

    // Unityn editorissa määritelty nimi opastepisteelle.
    public string Name = "";

    private void Awake() {

        GuideSpots.Add(
            new GuideSpotDTO {
                name = Name,
                pos = transform.position
            }
        );

        Destroy(this);
    }

}

// Apustrukti datan käsittelyyn.
public struct GuideSpotDTO {
    public string name;
    public Vector3 pos;
}
```

Kun nimi ja lokaatio on lisätty listaan objekti poistetaan jotta se ei jää lojumaan näkymään ja että voidaan varmistaa sen muuttumattomuus. Pisteitä voi tämän jälkeen käyttää missä tahansa GuideSpots staattisen muuttujan kautta. Tämän projektin käyttötapauksessa niitä kuitenkin käytettiin vain avustaja käyttöliittymän luomiseen.

3.5.5 Käyttöliittymä

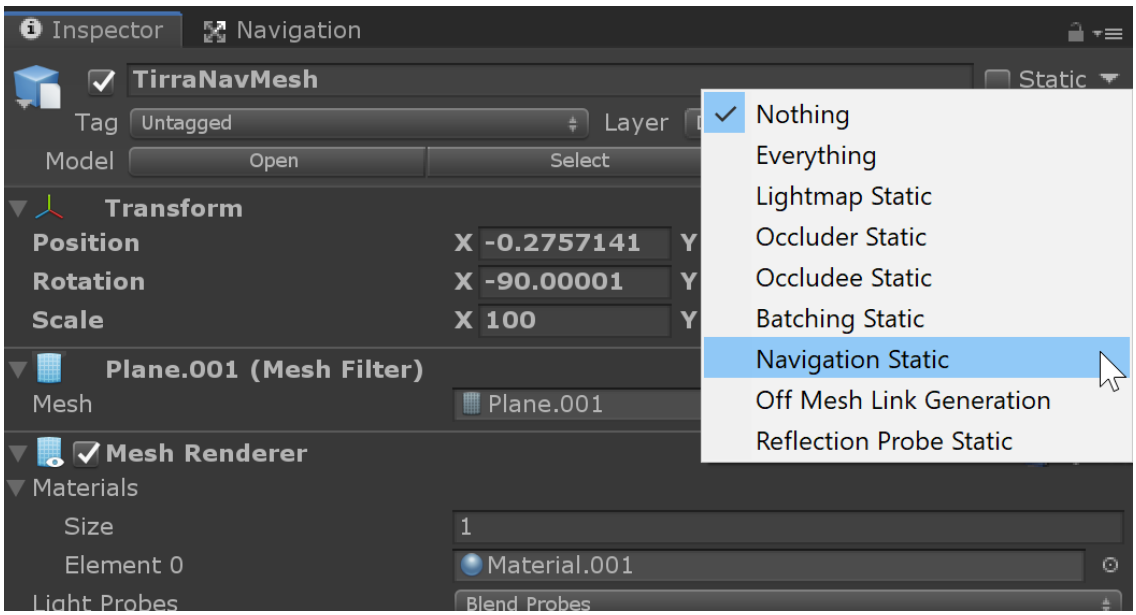
Avustajan käyttöliittymä on sen vieressä leijuva lista nappuloista jotka luodaan aikaisemmin luotujen opastepisteiden perusteella. Käyttöliittymä on esillä vain silloin kun avustajan lähellä on pelaaja. Silloin kun käyttöliittymä on näkyvissä nappuloita voi painaa kuka tahansa käyttöliittymän näkevä käyttäjä huolimatta siitä kuka on käyttäjää lähimpänä.

Kuva 10 Opastepisteillä alustettu käyttöliittymä avustajan vieressä. Kuva Unityn editoristä.

3.5.6 Polunetsintä

Polunetsintä koostuu kahdesta eri ongelmasta. Nämä ovat miten rakentaa jostakin tietystä ympäristöstä kartta jota voidaan loogisesti edetä ja miten kartan perusteella määritellään nopein reitti kohteeseen.

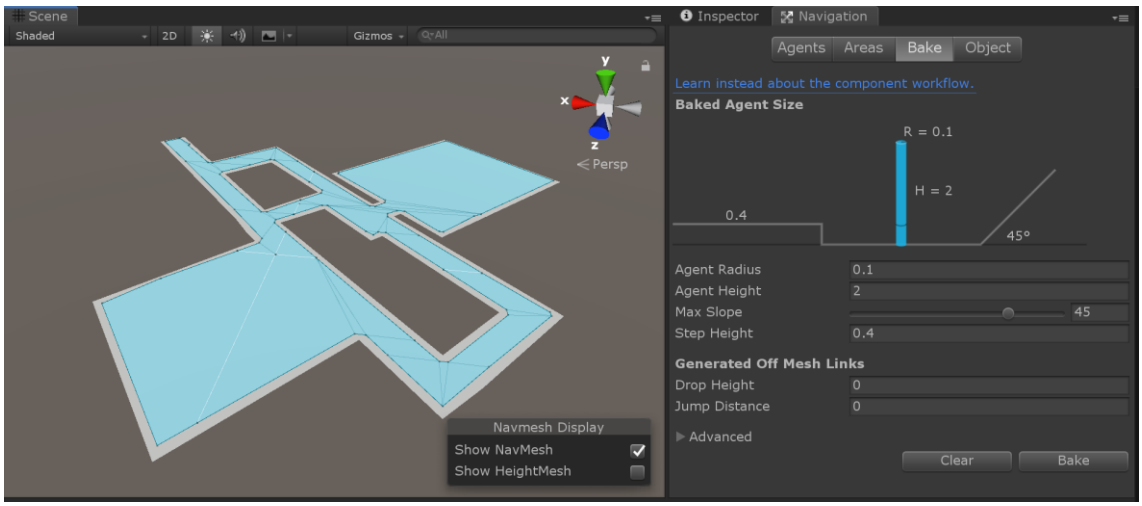
Ympäristön ymmärtämiseen unity käyttää navigaatioverkkoa (Navigation Mesh). Sen luominen tapahtuu olemassaolevien mallien päälle. Mallit jotka halutaan mukaan navigaatioverkon luontiin pitää merkitä *Navigation Static* lipulla.



Kuva 11 Navigation Static lipun saa laitettua unityssä inspector näkymän oikeasta yläkulmasta.

Kun kaikki halutut mallit on merkitty, pitää navigaatioverkko vielä rakentaa. Se tapahtui unityn Navigation välilehdessä, jossa sille määritellään parametrit ja se leivotaan eli luodaan etukäteen jotta sitä ei tarvitse sovelluksen käydessä laskea aina uudestaan. Verkon luomiseen tarkoitetut parametrit ja niiden tarkoitukset selitettynä:

- **Agent Radius.** Määrittää kuinka lähelle mallien reunoja navigaatioverkko menee.
- **Agent Height.** Vähimmäiskorkeus joka vaaditana pelin sisäisissä yksiköissä pinnan yläpuolelta jotta se otetaan mukaan verkon luontiin.
- **Max Slope.** Korkein kaltevuus jota pitkin verkko luodaan.
- **Step Height.** Kuinka korkeita askelia voidaan astua ylös. Askel tässä asiayhteydessä meinaa korkeinta mahdollista vertikaalia pintaa joka hahmon on mahdollista kiivetä ylös. Tämä on mm. portaikoiden takia hyvä säätää ensimmäisellä kerralla oikein tai navigaatioverkkoa ei generoida niiden päälle.



Kuva 12 Aikaisemmassa vaiheessa luodun mallin päälle rakennettu navigaatioverkko ja sen luontiin käytetyt parametrit.

Navigaatioverkko määrittelee liikuttavia alueita sovelluksen sisällä. Se koostuu kuperista monikulmioista (Convex polygon) jotka ovat käteviä edustamaan liikuttavia pintoja. Tämä johtuu siitä että monikulmion sisällä ei kahden pisteen välillä voi olla minkäänlaista estettä. Monikulmioiden rajojen lisäksi navigaatioverkko pitää sisällään myös tiedon siitä, mitkä monikulmiot ovat toistensa vieressä [12].

Toiseen ongelmaan eli loogisesti ennaltamääritellyn alueen navigointiin Unity tarjoaa myös ratkaisun joka on **NavMeshAgent**. NavMeshAgent on komponentti joka lukee aikaisemmin generoitua navigaatioverkkoa ja liikkuu käyttäen sitä hyväksi. Tämä liikkuminen tapahtuu ensin määrittämällä agentille päätepiste sen destination ominaisuuden (property) kautta jostakin skriptistä. Kun päätepiste on määritetty NavMeshAgent etsii ensin monikulmion jolla itse seisoo, sekä määriteltyä päätepistettä lähimmän monikulmion. Näiden kahden monikulmion väliltä etsitään monikulmioita pitkin etsimällä lyhyin reitti käyttämällä A* (A star) polunetsintäalgoritmiä [13].

3.5.7 Verkkologiikka

Avustajan verkkologiikka koostuu sen paikan, rotaation ja tilan synkronoinnista. Kaikki tämä hoidetaan master clientin toimesta. Myös avustajan toimintalogiikka ajetaan vain master clientin puolella jotta vältyttäisiin eroavaisuuksilta käyttäjien kesken. Avustajan tila koostuu tässä tapauksessa äärellisen automaatin tilasta, pelaajien lukumäärästä

avustajan lähellä, sekä lähimmän pelaajan uniikista identifikaationumerosta. Näitä ei käytännössä tarvitse synkronoida ollenkaan, koska master client hoitaa kaiken prosessoinnin. Ne synkronoidaan vain sen takia jos master client lopettaa sovelluksen käyttämisen tai menettää verkkoyhteyden. Palvelin valitsee näiden tapahtuessa uuden master clientin jäljellä olevien pelaajien joukosta. Jos uudelta master clientillä on eri tietdot kuin vanhalla tai siltä puuttuu tarvittavaa tietoa saattaa syntyä rajatapauksia tai ei haluttuja vuorovaikutuksia.

3.6 Havainnot

Tässä insinöörityössä toteutettu projekti vie tämän hetkisiä lisätyn todellisuuden teknologioita niiden äärirajoille. Suurimpia ongelmia tuottivat pistepilvien kanssa temppuilu projektin alussa koska niitä käytetään tavalla jota ei vielä suoraan tueta. Tästä huolimatta ideasta saatiin toteutettua käytettävä proof of concept sovellus. Projektin aikana tuli vastaan uusia konsepteja niin verkkologiikan suhteen kuin mallintamisen ja pistepilvien generoinnin suhteen. Haastavin osuus verkkologiikassa oli määritellä mitä dataa tarvitsee lähettää ja mikä voidaan laskea käyttäjien puolella aiheuttamatta ristiriitoja käyttäjien näkemän sisällön välillä ja pitämällä kaistan käyttö minimissä.

Lisäksi peiteseinien kalibrointi ei parhaista yrityksistä huolimatta ole joissakin kohdissa lähelläkään missä sen pitäisi. Tämä johtuu siitä että kartoitettu tila oli suhteellisen suuri joten pistepilven ääripäät ovat vääntyneet hieman suhteessa tilan muotoon. Tämä teki niiden sijoittamisesta pilven päälle hieman haastavampaa. Ne silti täyttävät tarpeensa tilanteessa missä toisessa huoneessa oleva käyttäjä tai avustaja ei näy seinien läpi.

Lisätyn todellisuuden sovelluksia rajoittaa kuitenkin myös vielä todella paljon laitteisto millä sovelluksia pyöritetään. Vaikka lopputulos oli tyydyttävä vaatii se edelleen todella tarkkoja olosuhteita kuten oikean valaistuksen ja suhteellisen esteettömän näkymän ympärillä olevaan tilaan.

4 Yhteenveto

Lisätty todellisuus teknologiana on todella lupaava. Siitä löytyy käyttökohteita aina teollisuussovelluksista mainostamiseen. Lisätyn todellisuuden persistenssi ja sen kehittyminen tulevat olemaan yksi tärkeimmistä askeleista sen laajamittaiseen käyttöönottoon koska se mahdollistaa sisällön tarkan jakamisen muiden käyttäjien kesken ja lisää sovellusten interaktiota ja mahdollisuuksia.

Jo tällä hetkellä persistenttien lisätyn todellisuuden sovellusten luominen ei ole rakettitiedettä. Se ei vaadi enää syvällistä tietoa laitteen sensoreista vaan melkein jokainen lisätyn todellisuuden laite tukee jotakin suurimmista pelimoottoreista jotka piilottavat sisäänsä monimutkaisen toteutuksen joka piti alussa toteuttaa aina itse.

Koska lähes kaikki suuret teknologiayhtiöt ovat kiinnostuneita lisätyn todellisuuden käyttötarkoituksista tulee se kasvamaan valtavasti seuraavan vuosikymmenen aikana. Tämän lisäksi teknologian parantuessa sille löydetään melko varmasti uusia käyttökohteita, mitkä saavat tällä hetkellä muutkin kuin teknologiayhtiöt kiinnostumaan lisätyn todellisuuden mahdollisuuksista.

Lähteet

1 Bill Meehan. 2017. NJ Utility on Forefront with New Mixed Reality Application. Verkkoaineisto. https://www.esri.com/about/newsroom/publications/wherenext/nj-utility-on-forefront-with-new-mixed-reality-application/ Luettu 11.5.2019.

2 https://www.augmented-reality-games.com/

3 Immersal. Verkkoaineisto. https://immersal.com

4 Immersal. Immersal AR Cloud SDK dokumentaatio. Verkkoaineisto. https://immersal.gitbook.io/sdk/

5 Apple. ARKit dokumentaatio. Verkkoaineisto. https://developer.apple.com/documentation/ARKit/understanding_world_tracking_in_ARKit

6 Mohammed Ibrahim. 2018. An introduction to ARKit 2—World Mapping. Verkkoaineisto. https://medium.com/@mohams3ios01/an-introduction-to-ARKit-2-world-mapping-5b38827f8ec0. Luettu 14.05.2019.

7 https://learnopengl.com/Advanced-OpenGL/Depth-testing

8 Brandi House. 2018. Evolving multiplayer games beyond Unet. Verkkoaineisto. https://blogs.unity3d.com/2018/08/02/evolving-multiplayer-games-beyond-unet/. Luettu 18.05.2019

9 DarkRift Networking. Verkkoaineisto. https://darkriftnetworking.com/DarkRift2/

10 Photon Unity Networking (PUN). Verkkoaineisto. https://www.photonengine.com/en/pun

11 Unity. Unity Documentation Verkkoaineisto. https://docs.unity3d.com/ScriptReference/Unity.Collections.NativeArray_1.html

12 Adobe. Mixamo | Common questions. Verkkoaineisto. https://helpx.adobe.com/creative-cloud/faq/mixamo-faq.html

13 Unity. Unity Documentation. Verkkoaineisto. https://docs.unity3d.com/Manual/nav-InnerWorkings.html

14 https://developer.apple.com/videos/play/wwdc2019/604

15 ARcritic. 2019. Verkkoaineisto. https://arcritic.com/3049/instagram-ar-face-filters-popularity-explained/ 22.07.2019

16 Ana Javornik. 2016. The Mainstreaming of Augmented Reality: A Brief History https://hbr.org/2016/10/the-mainstreaming-of-augmented-reality-a-brief-history Luettu 22.07.2019

17 https://vrscout.com/news/airbus-hololens-app-train-future-airline-crews/