SeAMK
SEINÄJOEN AMMATTIKORKEAKOULU
SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

Mikael Säntti

Puhekeskusteluominaisuuden kehittäminen metaversumiin

Opinnäytetyö
Syksy 2023
Insinööri (AMK), Tietotekniikka

SEINÄJOEN AMMATTIKORKEAKOULU

Opinnäytetyön tiivistelmä

Tutkinto-ohjelma: Insinööri (AMK), Tietotekniikka

Suuntautumisvaihtoehto: Ohjelmistotekniikka

Tekijä: Säntti, Mikael

Työn nimi: Puhekeskusteluominaisuuden kehittäminen metaversumiin

Ohjaaja: Hietamäki, Marko

Vuosi: 2023 Sivumäärä: 33 Liitteiden lukumäärä: 0

Opinnäytetyön aiheena oli kehittää puhekeskusteluominaisuus Seinäjoen ammattikorkeakoulun virtuaaliseen oppimisympäristöön. Työn tarkoituksena oli mahdollistaa reaaliaikainen puheyhteys metaversumin käyttäjien välillä. Tämä ominaisuus parantaisi immersiota ja saisi keskustelun tuntumaan todenmukaisemmalta. Lisäksi työn tavoitteena oli selvittää, miten Unity-pelimoottorin 3D-ääniasetuksia voisi hyödyntää ja muokata niin, että toisten käyttäjien puhe kuuluisi eri tavalla riippuen heidän sijainnistaan.

Työn alussa tutustuttiin Unity-pelimoottoriin ja tutkittiin erilaisia vaihtoehtoja ja tekniikoita, joilla tiedonsiirron asiakassovelluksen ja palvelimen välillä voisi toteuttaa. Tämän jälkeen toteutettiin asiakassovellusta ja palvelinta moduuleittain ja yhdistettiin ne toimivaksi kokonaisuudeksi. Lopuksi tutkittiin pelimoottorin 3D-ääniasetuksia ja miten niitä pitäisi muokata realistisemman äänimaailman saavuttamiseksi.

Tuloksena oli toimiva ohjelmakoodi, joka mahdollistaa puhekeskusteluominaisuuden metaversumin käyttäjien välillä. Ohjelmakoodi on kirjoitettu C#-ohjelmointikielellä. Palvelinpuolella on käytetty Node.js-alustaa, joka on avoimen lähdekoodin alustariippumaton ajoympäristö JavaScriptin suorittamiseen.

[1] Asiasanat: Unity, Metaversumi, C#, NodeJS, 3D-ääni

SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

Thesis abstract

Degree programme: Information Technology

Specialisation: Software Engineering

Author: Säntti, Mikael

Title of thesis: Developing a voice chat feature for the metaverse

Supervisor: Hietamäki, Marko

Year: 2023 Number of pages: 33 Number of appendices: 0

The goal of the thesis was to design and develop a voice chat feature for the virtual learning environment of Seinäjoki University of Applied Sciences. The purpose of the thesis project was to enable a real-time voice communication feature for the users of the metaverse. This feature was aimed to improve immersion and make conversation with other users feel more realistic. In addition, the aim of the thesis was to find out how the 3D audio settings of the Unity game engine could be adjusted and modified so that the volume of the speech of other users would change depending on their location and distance.

In the beginning of the thesis project, the game engine, and various options for implementing data transfer between the client application and the server were studied. After this, it was time to develop and test the client application and server side in modules and they were combined into a functioning application. Finally, the 3D audio settings of the game engine were examined, and it was studied how the settings should be adjusted to achieve a more realistic sound world.

As the result there was a functional Unity script that enabled the voice chat feature between the users of the metaverse. The Unity script was written in the C# programming language. On the server side Node.js was used, which is an open-source cross-platform server environment that executes JavaScript.

[1] Keywords: Unity, Metaverse, C#, NodeJS, 3D audio

SISÄLTÖ

Kuvaluettelo

Käytetyt termit ja lyhenteet

AudioClip	Sisältää äänitiedoston, jota AudioSource-komponentti käyttää äänentoistoon.
AudioListener	Unity-pelimoottorin komponentti, joka vastaanottaa syötteet kaikista kohtauksessa olevista äänilähteistä, jotka ovat tarpeeksi lähellä kuuntelijaa.
AudioSource	Unity-pelimoottorin komponentti, joka mahdollistaa äänen toistamisen kohtauksessa. Se sisältää monia säädettäviä äänenhallintaan liittyviä asetuksia.
Metaversumi	Sosiaalinen, internetiä hyödyntävä, monen käyttäjän virtuaalitodellisuus. Voidaan käyttää esimerkiksi tapahtumien luomiseen ja kokoontumispaikkana.
Ohjelmakoodi	Kutsutaan myös skriptiksi. Ohjelmoitu komponentti, joka voidaan liittää peliolioihin.
Radiopuhelintila	Mikrofonin toiminta edellyttää ennalta määritetyn näppäimen painamista puhuttaessa.
Scene	Kohtaus. Pelissä oleva yksittäinen kenttä tai valikko, missä pelaaja liikkuu. Sisältää erilaisia olioita, jotka näkyvät käyttäjille.
Unity	Ilmainen monialustainen pelimoottori, jolla on mahdollista luoda 2D- ja 3D-sovelluksia.
WebSocket	Reaaliaikainen viestintäprotokolla, joka mahdollistaa asiakkaan ja palvelimen välisen jatkuvan kaksisuuntaisen viestintäkanavan käytön yhden TCP-yhteyden kautta.

1 JOHDANTO

1.1 Työn tausta

Opinnäytetyö sai alkunsa Seinäjoen ammattikorkeakoulun tarpeesta kehittää metaversumiin puhekeskusteluominaisuus, jotta käyttäjät voisivat puhua keskenään virtuaalimaailmassa. Metaversumin kohdeyleisönä ovat erityisesti mahdolliset tulevat opiskelijat eli nykyiset ammattikoululaiset ja lukiolaiset. Metaversumin avulla käyttäjät voivat tutustua siihen minkälaista opiskelu SeAMKissa on.

1.2 Työn tavoite

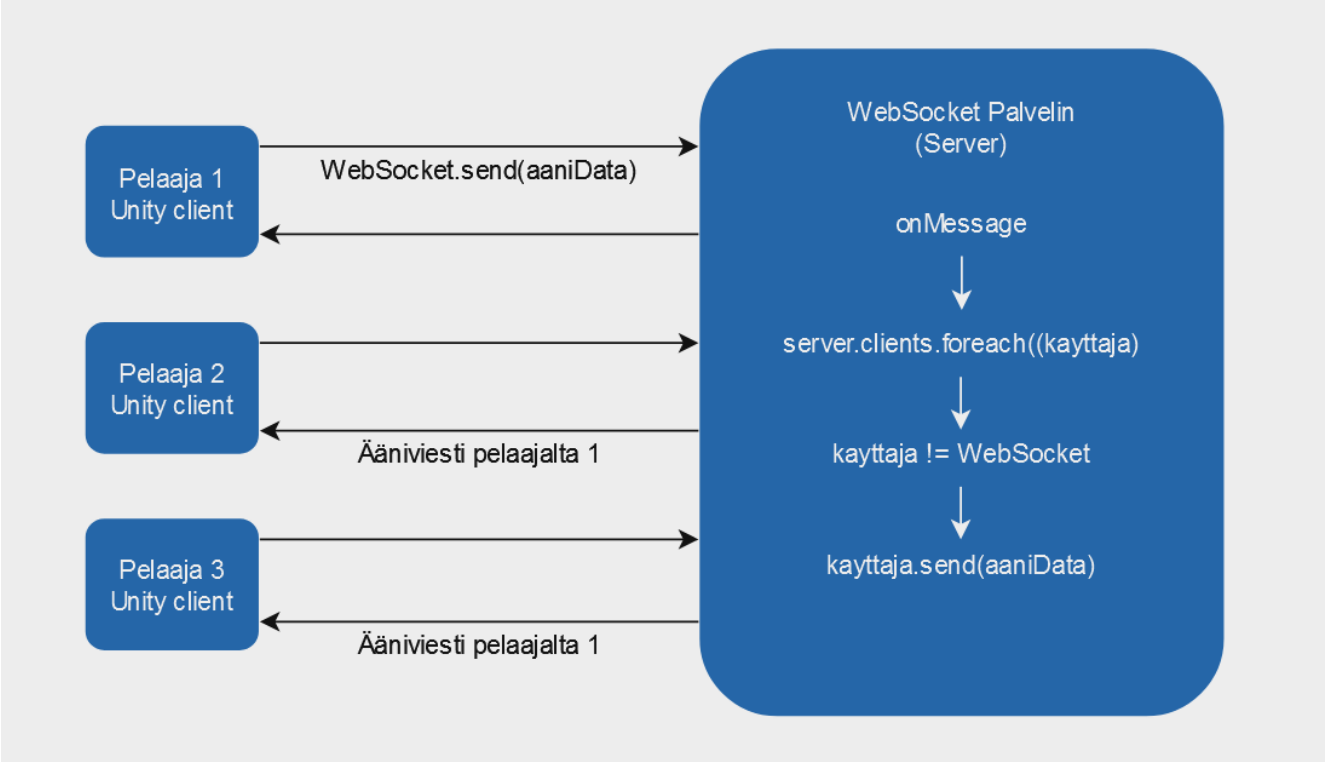
Opinnäytetyön tavoitteena oli aluksi tutustua erilaisiin reaaliaikaisiin äänensiirtovaihtoehtoihin ja selvittää, mitä niistä kannattaisi käyttää äänitiedoston siirtoon asiakassovelluksen ja palvelimen välillä. Miten se olisi mahdollista toteuttaa Unityssa ja miten Unityn 3D-ääniasetuksia voisi hyödyntää realistisen kuuloisen tilaäänen luomisessa, joka simuloisi pelaajalle toisten käyttäjien sijaintia, etäisyyttä ja nopeutta.

1.3 Työn rakenne

Aluksi kerrotaan työn sovelluksen arkkitehtuurista eli millä perusteella palvelin jakaa äänitiedostoja asiakassovellusten kesken. Seuraavaksi tutustutaan spatiaaliseen ääneen yleisesti ja kerrotaan, mitä 3D-ääniasetuksia täytyy muuttaa realistisemman äänimaailman aikaansaamiseksi. Tämän jälkeen kerrotaan asiakassovelluksen ja palvelimen välillä käytössä olevasta tiedonsiirtoprotokollasta. Teoriaosuuden jälkeen käydään läpi asiakassovellusta ja sen ohjelmakoodia funktioittain ja selostetaan, mitä jokaisessa koodin osassa tapahtuu. Seuraavaksi vuorossa on palvelinsovelluksen kuvaus yleisellä tasolla, jonka jälkeen syvennytään palvelimen koodiin. Lopuksi opetetaan, miten puhekeskusteluominaisuus otetaan käyttöön Unity-projektissa, ja käydään läpi työn testaus ja tulokset.

2 ASIAKAS-PALVELIN-ARKKITEHTUURI

Asiakas-palvelin-arkkitehtuurissa sovellus muodostuu kahdesta eri osasta, asiakkaasta ja palvelimesta (Syedmodassirali, 2022). Asiakas on Unity-sovellus, jonka pelaajaolio sisältää C#-ohjelmakoodin. Ohjelmakoodi muodostaa yhteyden palvelimelle. Palvelin on alustariippumaton Node.js-palvelin, joka käyttää tiedonsiirtoon WebSocket-protokollaa. Yhdistämisen jälkeen asiakkaan ja palvelimen välillä on jatkuvasti auki oleva kaksisuuntainen yhteys. Palvelimen tehtävänä on vastaanottaa asiakkaalta viestin ja välittää se eteenpäin toisille asiakkaille (kuva 1). Viestiä ei kuitenkaan lähetetä samalle asiakkaalle, joka lähetti viestin.



Kuva 1. Asiakkaan ja palvelimen välisen tiedonsiirron toimintaperiaate.

3 SPATIAALINEN ÄÄNI JA IMMERSIIVISYYS

3.1 Spatiaalinen ääni

Spatiaalisella äänellä eli toisin sanoen 3D-äänellä tarkoitetaan äänitekniikkaa, joka luo pelaajalle realistisemman ääniympäristön (Vicente, 2021). Tämä tekniikka parantaa pelikokemusta ja immersiota luoden uskottavamman maailman. Se antaa pelaajalle vaikutelman siitä, että ääni tulee virtuaalimaailmassa tietystä suunnasta ja etäisyydeltä. Esimerkiksi, jos jokin ääni kuuluu pelaajan vasemmalta puolella, niin se kuuluu voimakkaammin vasemmassa korvassa. Lisäksi äänilähde, joka loittonee kuuntelijasta, kuuluu vaimeammin sen etääntyessä ja vastaavasti lähestyvä äänilähde voimistuu, mitä lähempänä se on.

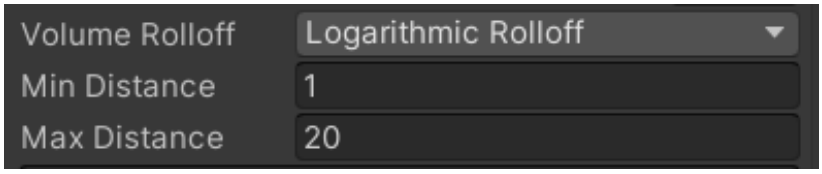
3.2 3D-ääniasetukset

Unitylla luotu sovellus tarvitsee jonkin lähteen äänelle (Unity Technologies, 2023j). Tämä onnistuu lisäämällä Audiosource-komponentti kohtauksessa olevaan olioon. Kun AudioSource on lisätty olioon, siihen voidaan liittää äänitiedosto Audio Clip -kenttään. AudioSourcelle on annettava AudioClip-tyyppiä oleva äänitiedosto eli projektiin tuotu äänitiedosto. AudioSource-komponentilla on useita 3D-äänentoistoon liittyviä asetuksia, joita säätämällä voidaan vaikuttaa siihen, miten ääni kuuluu virtuaalimaailmassa suhteessa hahmoon.

Spatial Blend-asetuksella määritetään, kuinka paljon äänilähteeseen vaikuttaa 3D-tilan ominaisuudet (Unity Technologies, 2023d). Tämä asetus vaikuttaa äänen sijaintiin, suuntaan ja etäisyyteen. Jos Spatial Blend-asetuksen arvo on 0.0, ääni kuuluu samalla tavalla joka puolelta. Jos Spatial Blend-asetuksen arvo on 1, ääni kuuluu vain siitä suunnasta missä äänilähde on. Kuvassa 2 on määritetty, että AudioSource-komponentti kuullaan 3D-lähteenä.

Kuva 2. Spatial Blend-asetus

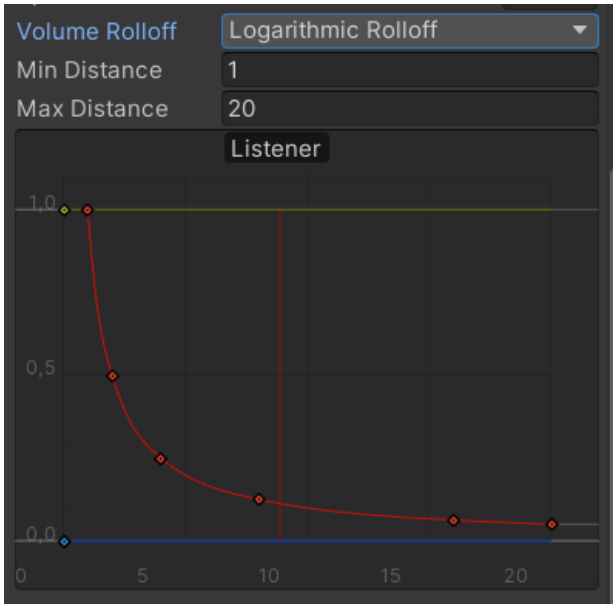
MinDistance- ja maxDistance-asetukset määrittävät äänen minimi- ja maksimietäisyydet. MinDistance-asetus määrittää etäisyyden, jonka sisällä ääni kuuluu mahdollisimman kovaa ja jonka jälkeen ääni alkaa vaimentua (Unity Technologies, 2023c). MaxDistance-asetus määrittää etäisyyden, mihin asti ääni vaimenee ja jonka jälkeen ääni ei enää kuulu (Unity Technologies, 2023b). Kun äänilähde on MinDistance-asetuksen ja MaxDistance-asetuksen välissä, etäisyyden kasvaessa ääni vaimenee Volume Rolloff-asetuksella määritetyllä tavalla. Kuvan 3 määritysten mukaisesti ääni kuuluu metrin säteellä mahdollisimman kovaa ja vaimenee logaritmisen käyrän mukaisesti 20 metriin asti. Jos valittu käyrä olisi lineaarinen niin ääni ei enää kuuluisi 20 metrin jälkeen.



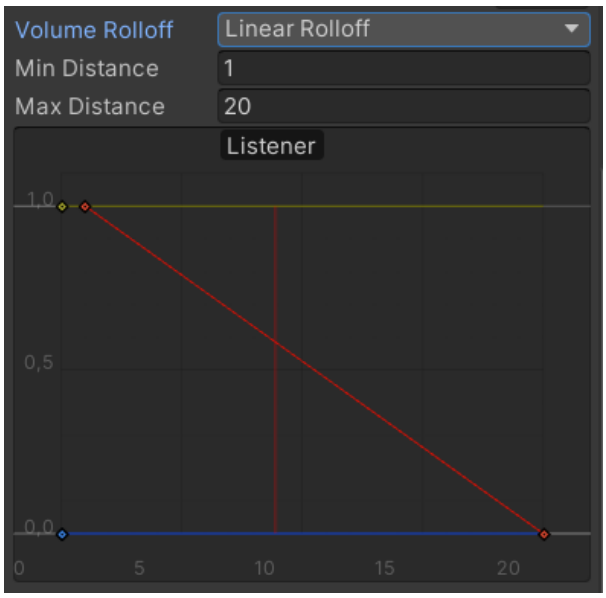
Kuva 3. Min Distance- ja Max Distance -asetukset

Doppler Level -asetus määrittää, miten voimakkaasti Doppler-ilmiötä (äänen taajuudessa havaittavaa muutosta, joka johtuu havaitsijan ja äänilähteen liikkeestä toistensa suhteen) simuloidaan äänilähteen ja kuulijan liikkuessa suhteessa toisiinsa (Buckley, 2023).

Volume Rolloff -asetuksella määritetään, miten äänenvoimakkuus vaimenee, kun äänilähde etääntyy kuuntelijasta (Unity Technologies, 2022). Jos valittu käyrä on logaritminen, niin äänenvoimakkuus laskee aluksi nopeasti, mutta hidastuu loppua kohden. Lineaarinen tarkoittaa, että äänenvoimakkuus laskee tasaisesti koko matkan, kunnes se lakkaa kuulumasta kokonaan. Mukautettu tarkoittaa, että käyttäjä voi itse määrittää, miten äänen voimakkuus kasvaa tai vaimenee riippuen etäisyydestä. Kuvissa 4 ja 5 on käytetty vertailun vuoksi kahta erilaista käyrää. Kuvassa 4 ääni kuuluu metrin säteellä mahdollisimman kovaa ja vaimenee logaritmisen käyrän mukaisesti 20 metriin asti, kunnes se ei enää vaimene. Kuvassa 5 ääni kuuluu metrin säteellä mahdollisimman kovaa ja vaimenee lineaarisesti 20 metriin asti, kunnes se lakkaa kuulumasta kokonaan.



Kuva 4. Logaritminen käyrä



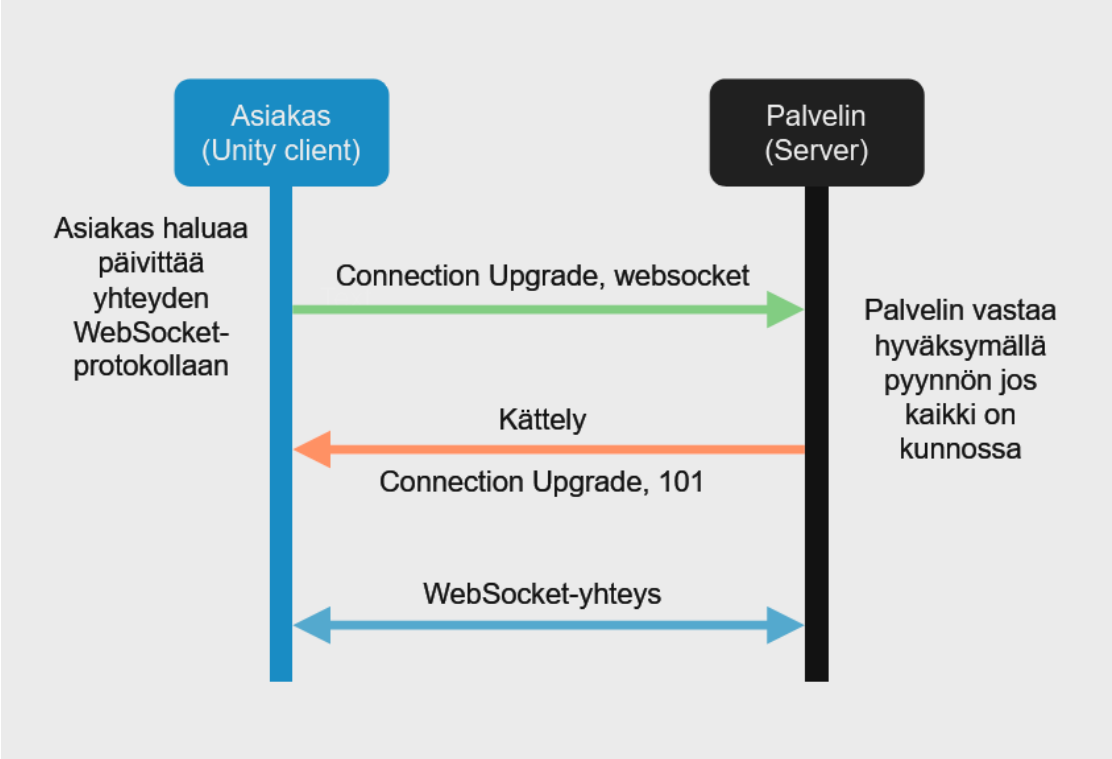
Kuva 5. Lineaarinen käyrä

Äänen toistamiseen tarvitaan AudioListener-komponenttia (Unity Technologies, 2023i). Se toimii pelaajan korvana eli se mahdollistaa äänen kuulemisen lähistöllä olevista ääniläh-teistä. Mikäli AudioListeneriä ei ole lisätty, niin pelaaja ei kuule mitään ääniä. Ensimmäisen persoonan peleissä AudioListener lisätään normaalisti pääkameraan eli pelaajan kame-raan.

4 TIEDONSIIRTO

WebSockets-protokolla mahdollistaa reaaliaikaisen, kahdensuuntaisen tiedonsiirron palvelimen ja asiakkaan välillä (Calvin, 2022). Toisin kuin perinteinen HTTP-pyyntö, joka sulkee yhteyden jokaisen pyynnön jälkeen, WebSockets pitää yhteyden avoimena ja mahdollistaa jatkuvan, kaksisuuntaisen tiedonsiirron. Tämä tarkoittaa, että palvelin ja asiakas voivat kommunikoida keskenään asynkronisesti eli ilman, että asiakkaan tarvitsee odottaa palvelimen vastausta. Asiakkaan ei tarvitse kysyä palvelimelta uusia päivityksiä, vaan palvelin voi lähettää niitä aina kun niitä on saatavilla. Tämä vähentää viivettä ja parantaa suorituskykyä, koska yhteyden avaaminen ja sulkeminen vie aikaa ja resursseja.

WebSocket-yhteys muodostetaan tavallisella HTTP-pyynnöllä (WebSockets Standard, 2023a). Pyyntö sisältää Upgrade-otsikkokentän, jonka arvona on websocket (Upgrade: websocket). Upgrade-otsikko kertoo, että asiakas haluaa päivittää yhteyden WebSocket-protokollaan. Vaatimuksena on, että palvelin kuuntelee WebSocket-pyyntöjä ja protokollan on oltava HTTP/1.1 tai uudempi. Yhteys pysyy aktiivisena niin kauan, kunnes toinen osapuolista tai jokin ulkoinen tekijä katkaisee yhteyden. Kuvassa 6 on kuvattu WebSocket-yhteyden muodostaminen.



Kuva 6. WebSocket-yhteyden muodostaminen.

.

5 ASIAKASSOVELLUS

5.1 Unity

Unity on Unity Technologiesin luoma ilmainen, suorituskykyinen ja monia alustoja tukeva pelimoottori (Zenva, 2023). Unity mahdollistaa pelien luomisen alustoille, kuten Windows, Mac, Linux, Android ja iOS. Unitylla on mahdollista luoda sekä 2D- että 3D-pelejä. Ohjelmakoodit kirjoitetaan C#-ohjelmointikielellä, joka kuitenkin eroaa tavallisesta siten, että Unityssa oletuksena kaikki C#-ohjelmakoodit periytyvät MonoBehaviour-luokasta (Unity Technologies, 2023f).

MonoBehaviour-luokka on Component-luokan perivä luokka, joka sisältää erilaisia ennalta määritettyjä metodeja, joilla määritetään, miten olio käyttäytyy eri tilanteissa (Unity Technologies, 2023g). Opinnäytetyön kannalta tarvittavat metodit ovat Start ja Update. Start-metodia kutsutaan tasan kerran ohjelmakoodin elinaikana ennen ensimmäistä ruudunpäivitystä ja ennen kuin Update-metodia kutsutaan ensimmäisen kerran.

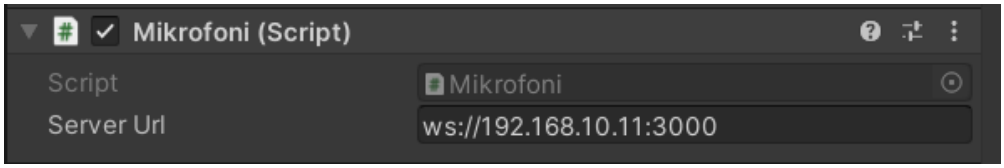
Update-metodia kutsutaan jokaisen ruudunpäivityksen yhteydessä ennen ruudun piirtämistä (Unity Technologies, 2023h). Se on yksi tärkeimmistä toiminnoista, jotka liittyvät pelin logiikan päivitykseen. Esimerkiksi, jos ruudunpäivitysnopeus on 60 FPS (Frames Per Second), Update-metodia kutsutaan 60 kertaa sekunnissa. FPS ei kuitenkaan ole aina vakaa, joten Update-metodin kutsujen määrä sekunnissa saattaa vaihdella. Update-metodi sopii hyvin käyttäjän syötteiden, kuten näppäinten painallusten, tarkkailuun.

5.2 Ohjelmakoodi

Palvelimeen yhdistämistä ja äänen nauhoitusta, lähetystä ja vastaanottamista varten luotiin ohjelmakoodi, joka tutkii, missä tilassa radiopuhelintilan aktivoimiseksi määritetty painike on. Kun uusi hahmo liittyy huoneeseen, liittyy se samalla myös palvelimelle, joka välittää äänitiedostoja käyttäjien kesken.

5.2.1 Palvelimelle yhdistäminen

Pelaajaoliolle komponentiksi lisätty Mikrofoni-ohjelmakoodi hoitaa monta eri toimintoa. Heti aluksi, kun pelaaja liittyy huoneeseen, kutsutaan YhdistaPalvelimelle-funktiota. Start-metodi suoritetaan heti käynnistyksessä, joten sieltä on hyvä kutsua funktioita, jotka täytyy ajaa heti aluksi, mutta joita ei tarvitse kutsua enää uudelleen. serverUrl-muuttuja on määritetty julkiseksi. Tällöin Unity-editorin Inspector-näkymässä ohjelmakoodissa näkyy Server Url-kenttä, joka saadaan konfiguroitua WebSocket-yhteyttä varten syöttämällä palvelimen osoite ja portti. Kuvan 7 Inspector-näkymässä on määritetty pelaajaoliolle ohjelmakoodi ja palvelimen osoite sekä portti.



Kuva 7. Pelaajan mikrofoni -ohjelmakoodi

YhdistaPalvelimelle-funktiossa luodaan uusi WebSocket-olio. Parametrina annetaan palvelimen osoite ja portti. WebSocket-oliolla on tapahtumankäsittelijä yhteyden avaamiselle (OnOpen), viestien vastaanottamiselle (OnMessage), virheille (OnError) ja yhteyden sulkemiselle (OnClose) (kuva 8).

OnOpen-tapahtuma laukeaa, kun WebSocket-palvelin on onnistuneesti käsitellyt yhteyspyynnön ja hyväksynyt yhteyden avaamisen (WebSockets Standard, 2023b). OnMessage-tapahtuma laukeaa joka kerta, kun vastaanotetaan uusi viesti palvelimelta. Tässä tapauksessa viesti on toisen asiakkaan lähettämä binääridata-tyyppinen muuttuja. OnError-tapahtuma laukeaa, jos syntyy virhe. Virhe katkaisee asiakkaan yhteyden palvelimelle. OnClose-tapahtuma laukeaa, kun yhteys on katkaistu.

```
1 reference
private void YhdistaPalvelimelle() {
    ws = new WebSocket(serverUrl); // Julkinen muuttuja serverUrl konfiguroidaan Unity-editorilla
    ws.OnOpen += (sender, err) => Debug.Log("Yhdistetty palvelimelle!");

    ws.OnMessage += VastaanotettuAaniDataa;
    ws.OnError += (sender, err) => Debug.LogError("WebSocket error: " + err.Message);
    ws.OnClose += (sender, err) => Debug.Log("Yhteys palvelimeen katkaistu.");

    ws.Connect();
}
```

Kuva 8. YhdistaPalvelimelle-funktio

5.2.2 Äänen nauhoitus ja lähetys

Mikrofoni-ohjelmakoodin Update-metodissa tarkkaillaan näppäimistön syötteitä eli onko määritetty painike painettu pohjaan vai vapautettu. Kuvassa 9 on radiopuhelintilan näppäimeksi ohjelmakoodissa määritetty T-näppäin. Jokaisen ruudunpäivityksen yhteydessä tarkistetaan, onko näppäimen tila muuttunut. Jos näppäin on pohjassa, tarkistetaan, onko nauhoitus jo aloitettu. Jos nauhoittaa-muuttujan arvo on epätosi eli ”false” ei nauhoitusta ole vielä aloitettu, joten se aloitetaan kutsumalla AloitaNauhoitus-funktiota.

```
private void Update() {
    if (Input.GetKeyDown(KeyCode.T)) {
        if (!nauhoittaa) {
            AloitaNauhoitus();
        }
    } else if (Input.GetKeyUp(KeyCode.T)) {
        if (nauhoittaa) {
            LopetaNauhoitus();
        }
    }
}
```

Kuva 9. Update-metodi

Kuvassa 10 Microphone-luokan Start-metodia kutsumalla aloitetaan äänen nauhoitus mikrofonilla (Unity Technologies, 2023e). Parametreina annetaan:

- deviceName: null – Null tarkoittaa, että käytetään järjestelmän oletusmikrofonia. Tähän voisi määrittää haluamansa laitteen nimen.
- loop: true - Määrittää jatketaanko nauhoittamista, jos kolmannessa parametrissa (lengthSec) määritetty äänileikkeen pituus saavutetaan.
- lengthSec: 5 - Tämä parametri määrittää, kuinka monta sekuntia ääniraitaa tallennetaan ennen kuin äänitys automaattisesti pysäytetään.
- frequency: 48000 - Tämä on äänen näytteenottotaajuus eli kuinka monta ääninäytettä tallennetaan sekunnissa. Yleisesti käytettyjä arvoja ovat 44100 Hz (CD-laatu) ja 48000 Hz (DVD-laatu). Tässä tapauksessa 48000 tarkoittaa 48000 ääninäytettä sekunnissa. Näytteenottotaajuuden asettamiseen voidaan käyttää Unityn AudioSettings-luokan outputSampleRate-ominaisuutta, joka palauttaa käytettävän mikrofonin näytteenottotaajuuden.

Kun äänen nauhoitus on aloitettu, asetetaan nauhoittaa-muuttujan arvoksi tosi eli ”true”.

```
private void AloitaNauhoitus() {
    if (!nauhoittaa) {
        aanileike = Microphone.Start(null, true, 5, AudioSettings.outputSampleRate);
        nauhoittaa = true;
    }
}
```

Kuva 10. AloitaNauhoitus-funktio

Kun T-näppäin vapautetaan ja nauhoittaa-muuttujan arvo on tosi, lopetetaan nauhoitus kutsumalla LopetaNauhoitus-funktiota. End-metodi lopettaa mikrofonilla nauhoituksen. Parametrina annetaan:

- deviceName: null – Null tai tyhjä merkkijono tarkoittaa, että käytetään järjestelmän oletusmikrofonia. Tähän voisi määrittää haluamansa laitteen nimen.

Kun näppäin on vapautettu, muunnetaan ääniviesti binäärimuotoon MuunnaBinaariMuotoon-funktiolla ja lähetetään palvelimelle käyttämällä WebSockets-luokan Send-metodia. Tämän jälkeen asetetaan nauhoittaa-muuttujan arvoksi epätosi, merkiksi siitä, että nauhoitus on loppunut (kuva 11).

```
private void LopetaNauhoitus() {
    if (nauhoittaa) {
        Microphone.End(null);

        float[] data = new float[aanileike.samples * aanileike.channels];
        aanileike.GetData(data, 0);

        byte[] aaniviesti = MuunnaBinaariMuotoon(data);
        ws.Send(aaniviesti);

        nauhoittaa = false;
    }
}
```

Kuva 11. LopetaNauhoitus-funktio

5.2.3 Äänidatan vastaanottaminen

OnMessage-tapahtumankäsittelijä laukeaa, kun WebSocket-palvelin lähettää viestin. Silloin kutsutaan VastaanotettuAaniData-funktiota. Argumentti ”e” sisältää käyttäjän lähettämän äänidatan binäärimuodossa ja se annetaan arvoksi raakaData-muuttujalle. Seuraavaksi raakaData muunnetaan AudioSource-komponentin vaatimusten mukaan liukulukutaulukoksi ja tallennetaan aaniData-muuttujaan, joka annetaan ToistaVastaanotettuAaniData-funktiolle parametrina (kuva 12).

Useat Unityn toiminnot täytyy suorittaa pääsäikeessä ja tämä koskee myös äänen toistoa. Pääsäie on oletussäie, joka käynnistyy sovelluksen alussa (Unity Technologies, 2021). Pääsäie (main thread tai UI thread) luo tarvittaessa uusia säikeitä, jotka suorittavat vaadittuja tehtäviä rinnakkain ja synkronoivat pääsäikeen kanssa, kun tehtävä on suoritettu. VastaanotettuAaniData-funktio suoritetaan WebSocketsin säikeessä eli taustasäikeessä, kun palvelimelta saapuva viesti laukaisee OnMessage-tapahtuman.

UnityMainThreadDispatcher-luokkaa tarvitaan varmistamaan, että palvelimelta vastaanotettu ääniviestin käsittely ja toistaminen tapahtuu taustasäikeen sijaan pääsäikeessä (PimDeWitte, 2023).

```
private void VastaanotettuAaniData(object sender, MessageEventArgs e) {
    try {
        // Toisen käyttäjän lähettämä äänitiedosto
        byte[] raakaData = e.RawData;
        float[] aaniData = MuunnaAaniLeikkeenMuotoon(raakaData);

        // Vaihdetaan pääsäikeeseen äänen toistoa varten
        UnityMainThreadDispatcher.Instance().Enqueue(ToistaVastaanotettuAaniData(aaniData));
    }
    catch (System.Exception ex) {
        Debug.Log(ex);
    }
}
```

Kuva 12. VastaanotettuAaniData-funktio

ToistaVastaanotettuData-funktiossa luodaan vastaanotettuAaniLeike-niminen AudioClip-komponentti (Unity Technologies, 2023a). Ensin haetaan AudioSource-komponentti ohjelmakoodin olioon liitetyistä komponenteista ja nimetään se aaniLahteeksi. AudioClipin-komponentin luomiseen tarvitaan funktion parametrina saatua aaniData-taulukkoa. Taulukon pituus annetaan äänileikkeelle lenghtSamples-parametrina, lisäksi aaniData-taulukko asetetaan AudioClip-komponentin arvoksi SetData-metodilla. Parametreina äänileikkeelle annetaan:

- name: ”VastaanotettuAani” – Tämä on äänileikkeelle annettava nimi,
- lengthSamples: integer - Vastaanotetun äänitiedoston pituus,
- channels: 1 tai 2 - Määrittää kanavien lukumäärän eli onko äänileike mono vai stereo,
- frequency: 48000 – Näytteenottotaajuus on 48kHz.
- _3D: true tai false - Hyvin oleellinen tämän työn kannalta, sillä se määrittää toistetaanko äänileike 2D- vai 3D-tilassa.

Kun AudioClip on luotu, asetetaan se AudioSourcen clip-ominaisuuden arvoksi. Lopuksi toistetaan äänileike kutsumalla aaniLahde-AudioSource-komponentin Play-metodia (kuva 13).

```
private IEnumerator ToistaVastaanotettuAaniData(float[] aaniData) {
    aaniLahde = gameObject.GetComponent<AudioSource>();

    int aaniDataPituus = aaniData.Length;

    vastaanotettuAaniLeike = AudioClip.Create("VastaanotettuAani", aaniDataPituus, 1, 48000, true);
    vastaanotettuAaniLeike.SetData(aaniData, 0);

    if (aaniLahde == null) {
        aaniLahde = gameObject.AddComponent<AudioSource>();
        aaniLahde.playOnAwake = false;
        aaniLahde.spatialBlend = 0f;
    }

    // Toistetaan vastaanotettu äänileike
    aaniLahde.clip = vastaanotettuAaniLeike;
    aaniLahde.Play();

    yield return new WaitForSeconds(vastaanotettuAaniLeike.length);
}
```

Kuva 13. ToistaVastaanotettuAaniData-funktio

6 PALVELINSOVELLUS

6.1 Palvelin

WebSocket-palvelin on toteutettu NodeJS-alustalla, se on ominaisuuksiltaan ja toteutukseltaan hyvin yksinkertainen. NodeJS on alusta, joka mahdollistaa JavaScriptin käytön palvelinpuolella (OpenJS Foundation, i.a.). NodeJS:n avulla ohjelmoijat voivat hyödyntää olemassa olevaa JavaScript-osaamistaan ja luoda palvelinpuolen sovelluksia helpommin. WebSocket-palvelin on vastuussa käyttäjien lähettämien äänitiedostojen vastaanottamisesta ja eteenpäin lähetyksestä toisille käyttäjille. Palvelin pitää myös kirjaa siitä, kuka on lähettänyt minkäkin äänitiedoston, jotta se ei lähetä samaa äänitiedostoa takaisin lähettäjälle. Tämä mahdollistaa sujuvan ja häiriöttömän ääniviestinnän käyttäjien kesken.

6.2 Palvelimen koodi

Tässä työssä WebSocket-palvelin on toteutettu käyttäen ws-kirjastoa sen nopeuden ja yksinkertaisuuden vuoksi (websockets, 2023). Kirjasto asennetaan ”npm install ws” -komennolla ja importoidaan se projektiin WebSocket-muuttujalle (kuva 14). Tämän jälkeen luodaan uusi WebSocket-palvelinolio, joka kuuntelee porttia 3000 (kuva 15). Mikäli hostia ei ole määritetty, palvelimelle ei voi yhdistää lähiverkon ulkopuolelta.

```
const WebSocket = require('ws');
```

Kuva 14. Importoidaan ws-kirjasto

```
const server = new WebSocket.Server({ port: 3000, host: '0.0.0.0' });
```

Kuva 15. Luodaan uusi WebSocket server

Kuvassa 16 Connection-tapahtumalle lisätään kuuntelija, joka laukeaa, kun uusi WebSocket-yhteys luodaan. Tapahtuman sisällä luodaan uusi satunnainen käyttäjäId uudelle WebSocket-yhteydelle. Lisäksi jokainen WebSocket-yhteys ja käyttäjäId tallennetaan kayttajat-nimiseen Map-tietorakenteeseen, että pysytään perillä siitä, mikä WebSocket-yhteys kuuluu kullekin käyttäjälle (kuva 17).

```
server.on('connection', (ws) => {
```

Kuva 16. Kuuntelija Connection-tapahtumalle

```
const kayttajaId = generoiKayttajalleId();
kayttajat.set(ws, kayttajaId);
```

Kuva 17. Käyttäjät-tietorakenne

Kuvassa 18 Message-tapahtumalle lisätään kuuntelija. Tapahtumankäsittelijä laukeaa, kun WebSocket-asiakas lähettää viestin eli kun jokin käyttäjistä on lähettänyt äänisviestin. Vastaanotettu viesti on binääridatamuodossa. Viestiä ei käsitellä mitenkään palvelimella, vaan se välitetään eteenpäin kaikille muille käyttäjille. Ennen lähetystä tarkistetaan, että viestin lähettäjä ei ole käyttäjä, jolle palvelin on seuraavaksi lähettämässä ääniviestin, ja että yhteys käyttäjään on varmasti auki (kuva 19).

```
ws.on('message', (viesti) => {
```

Kuva 18. Kuuntelija message-tapahtumalle

```
server.clients.forEach((kayttaja) => {
  if (kayttaja !== ws && kayttaja.readyState === WebSocket.OPEN) {
    kayttaja.send(aaniData);
  }
});
```

Kuva 19. Äänidatan välitys käyttäjille

Lopuksi tarvitaan vielä kuuntelija Close-tapahtumalle. Kuvassa 20 oleva tapahtumankäsittelijä ”close” laukeaa, jos käyttäjä katkaisee yhteyden palvelimelle. Tässä tapauksessa käyttäjä poistetaan kayttajat-Map-tietorakenteesta.

```
ws.on('close', () => {
  kayttajat.delete(ws);
  console.log(`Käyttäjä ${kayttajaId} katkaisi yhteyden.`);
});
```

Kuva 20. Käyttäjän poistaminen

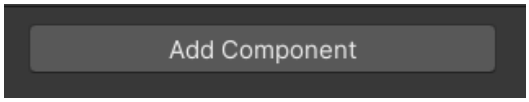
```
const WebSocket = require('ws');

const server = new WebSocket.Server({ port: 3000, host: '0.0.0.0' });
const kayttajat = new Map();

server.on('connection', (ws) => {
  const kayttajaId = generoiKayttajalleId();
  kayttajat.set(ws, kayttajaId);

  console.log(`Käyttäjä ${kayttajaId} liittyi palvelimelle.`);

  ws.on('message', (viesti) => {
    const aaniData = viesti;
    console.log(`Vastaanotettu äänidataa käyttäjältä ${kayttajaId}: ${aaniData.length} tavua.`);

    // Lähetetään äänidataa kaikille liittyneille käyttäjille paitsi lähettejälle itselleen
    server.clients.forEach((kayttaja) => {
      if (kayttaja !== ws && kayttaja.readyState === WebSocket.OPEN) {
        kayttaja.send(aaniData);
      }
    });
  });

  ws.on('close', () => {
    kayttajat.delete(ws);
    console.log(`Käyttäjä ${kayttajaId} katkaisi yhteyden.`);
  });
});
```

Kuva 21. Palvelimen koodi

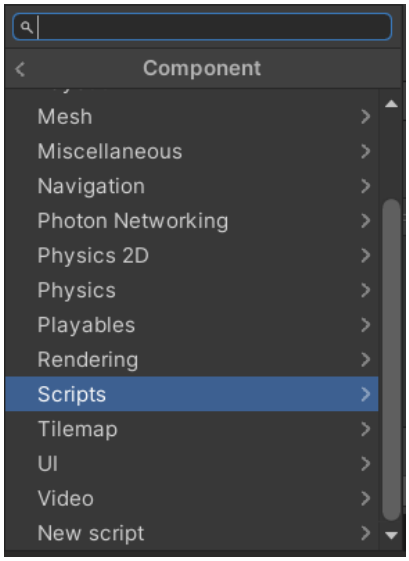
7 PUHEKESKUSTELUOMINAISUUDEN KÄYTTÖÖNOTTO UNITYSSA

7.1 Pelaajaolio

Tässä työssä on oletuksena, että kohtaus kohtaiset ohjelmakoodit löytyvät Unity-projektin Assets > Scenes > scenen nimi > Scripts-kansiosta. Mikrofoni-ohjelmakoodi lisätään pelaajaoliolle seuraavasti: Hierarkianäkymästä valitaan Player, jolloin Inspector-näkymä avautuu. Inspectorissa napsautetaan Add Component -painiketta (kuva 22) ja etsitään Mikrofoni-ohjelmakoodi seuraavasti: Scripts > Mikrofoni (kuva 23). Inspector-näkymään avautuu Mikrofoni-komponentti.



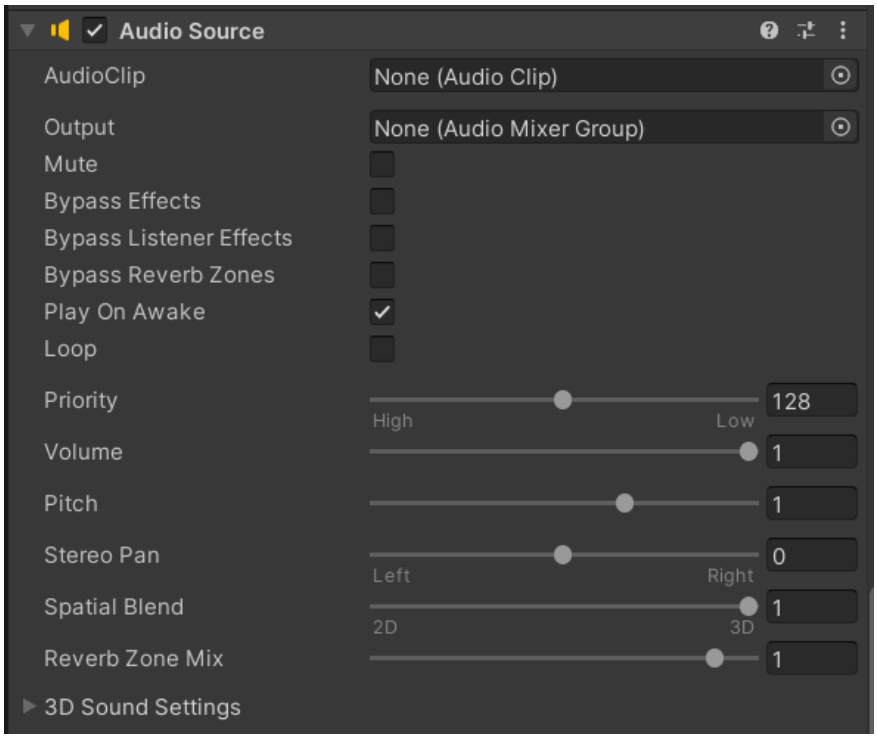
Kuva 22. Inspector-näkymän Add Component -painike



Kuva 23. Component-lista

7.2 AudioSource-komponentti

Seuraavaksi lisätään pelaajaoliolle AudioSource. Painetaan taas Add Component-painiketta (kuva 22) ja valitaan listasta Audio > Audio Source. Vaihtoehtoisesti AudioSourcen voisi lisätä pääkameraan eli tässä työssä pelaajaolion kameraan. Kuvassa 24 näkyvä AudioSource-komponentti sisältää erilaisia asetuksia, joita tarvitaan myöhemmin spatiaalisen äänen toteuttamiseen.



Kuva 24. AudioSource-komponentin asetuksia

7.3 AudioListener-komponentti

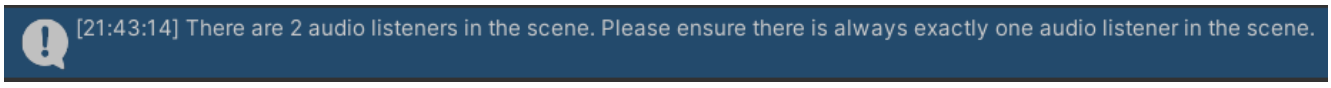
AudioListener-komponentti mahdollistaa äänen kuulemisen. AudioListener kuuntelee kaikkia kohtauksessa olevia äänilähteitä (AudioSource). Jos äänilähteet ovat 3D-tilassa, kuuntelija emuloi äänilähteen sijaintia ja liikkeitä.

Pelaajaoliolle lisätään äänen kuuntelija. Uusi komponentti lisätään taas painamalla Add Component -painiketta (kuva 22) ja valitaan listasta Audio > Audio Listener (kuva 25).

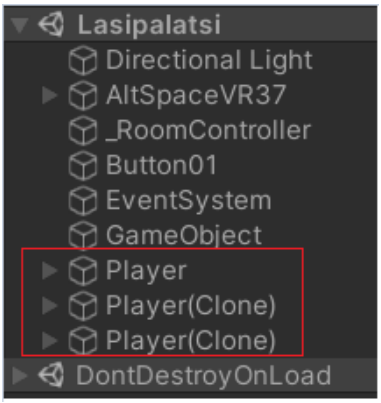
Muita vaiheita tai asetuksia ei tarvita. AudioListener-komponentti on yleensä kiinnitetty pelaajaolioon tai kameraan, jotta se seuraa pelaajan liikkeitä.

Kuva 25. Audio Listener-komponentti

On tärkeää huomioida, että kohtauksessa saa olla aktiivisena vain yksi AudioListener-komponentti kerrallaan. Jos AudioListenereita on useampia, Unity huomauttaa tästä, koska se saattaa aiheuttaa odottamattomia ongelmia äänen toistamisen kanssa (kuva 26). Etenkin moninpelien kehitysvaiheessa tämä saattaa aiheuttaa ongelmia, koska jokaiselle pelaajalle tarvitaan kohtauksen äänien kuulemiseksi oma AudioListener-komponentti. Photon Unity Networking -kirjasto luo kopion jokaisesta etäpelaajasta (remote Player), joten jos pelaajia on esimerkiksi 3, niin yhdessä kohtauksessa on myös 3 AudioListener-komponenttia. Kuvassa 27 näkyy paikallisen pelaajan (local Player) lisäksi myös 2 kopioita, jotka edustavat etäpelaajia.



Kuva 26. Kohtauksessa saa olla vain yksi aktiivinen AudioListener-komponentti.



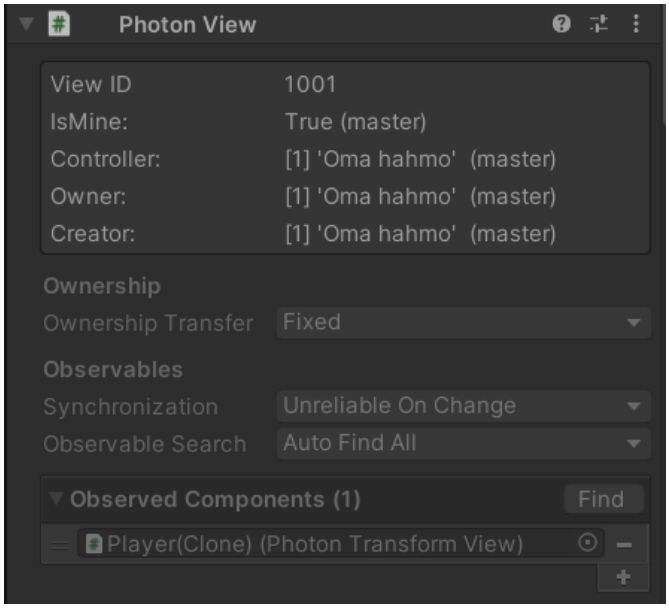
Kuva 27. Paikallinen pelaaja ja 2 kopiota.

Tässä työssä usean AudioListener-komponentin ongelma on ratkaistu käyttämällä Photon Unity Networking (PUN) -kirjaston PhotonView-luokkaa ja sen ominaisuuksia (Photon Engine, 2023b). Jokaiselle pelaajaoliolle asetetaan Photon View -ohjelmakoodi, jolla saadaan kaikki PhotonView-luokan funktiot ja ominaisuudet käyttöön.

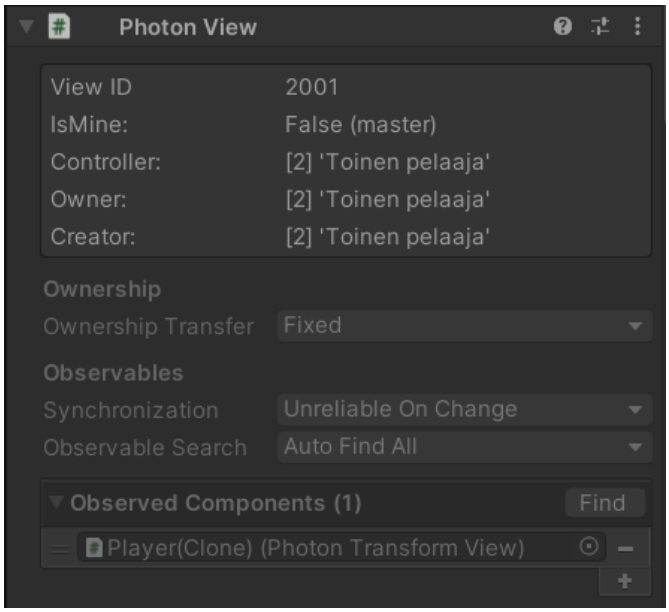
IsMine-ominaisuus on kätevä tapa varmistaa, että kyseessä on paikallinen pelaaja ja että vain se suorittaa haluttuja toimintoja (Photon Engine, 2023a). IsMine palauttaa totuusarvomuuttujan ”true”, jos kyseessä on paikallinen pelaaja (kuva 29). Mikäli totuusarvomuuttujan arvona saadaan ”false”, kyseessä on kopio toisen pelaajan hahmosta (kuva 30). Tällöin käytetään Destroy-funktiota tuhoamaan pelaajaolion AudioListener-komponentti (kuva 28).

```
if (!photonView.IsMine) {
    // Poistetaan remote Playerin kopiolta AudioListener
    Destroy(GetComponent<AudioListener>());
}
```

Kuva 28. Kopiolta poistetaan AudioListener-komponentti



Kuva 29. IsMine: True. Kyseessä on paikallinen pelaaja.



Kuva 30. IsMine: False. Kyseessä on kopio toisesta pelaajasta.

8 TESTAUS JA TULOKSET

Opinnäytetyön tavoitteet täyttyivät, sillä ohjelma mahdollistaa käyttäjien osallistumisen puhekeskusteluihin liittymällä muiden käyttäjien huoneisiin ja hyödyntämällä tietokoneeseen liitettyä mikrofonia. Toisena tavoitteena oli tutkia, miten Unityssa voitaisiin 3D-ääniasetuksia hyödyntämällä toteuttaa tilaääni, joka simuloisi toisten käyttäjien sijaintia.

Ohjelman eri osioita testattiin kehitysvaiheessa erillisinä yksikköinä luomalla jokaista testiä varten tyhjä projekti. Näin muut ohjelmakoodit tai kohtauksessa olevat oliot eivät vaikuta testaukseen. Testauksen ensimmäinen vaihe oli tarkistaa palvelimelle yhdistämisen, äänitiedoston lähettämisen, vastaanottamisen ja toistamisen toimivuus erillisellä C#-sovelluksella. Tavoitteena oli saada ääni kuulumaan kahden eri tietokoneen välillä lähiverkossa. Seuraava vaihe oli testata yhteyden toimivuutta asiakkaiden välillä, jotka liittyivät palvelimelle eri verkoista. Kun äänen siirto oli saatu toimimaan, luotiin vastaava ominaisuus Unityn ohjelmakoodilla.

Puhekeskusteluun sopivat 3D-ääniasetukset löytyivät lähinnä kokeilemalla. Tarvittavia asetuksia säädettiin, kunnes puhe kuului halutulle etäisyydelle asti. Etäisyyden kasvaessa asetetun rajan yli, puhe alkoi vaimenemaan. Lopulta, kun etäisyys kasvoi liian suureksi, puhe ei enää kuulunut lainkaan.

Ohjelman suorituskykyä ja käyttökokemusta voitaisiin parantaa optimoimalla äänitiedostojen käsittelyä, lähetystä ja vastaanottamista viiveen vähentämiseksi. Toinen mahdollinen kehityskohde on äänitiedoston vastaanottaminen, josta voisi tehdä täysin reaaliaikaista eli mahdollistaa suoratoisto.

LÄHTEET

Buckley, D. (22.10.2023). *How to Add the Doppler Effect in Unity*. GameDev Academy. https://gamedevacademy.org/unity-doppler-effect-tutorial/

Calvin, D. (6.2.2022). *Networking: Introduction to WebSockets*. Medium. https://medium.com/@Arockne/networking-introduction-to-websockets-7c0ffc3c2d6e

OpenJS Foundation. (i.a.). *About Node.js*. https://nodejs.org/en/about

Photon Engine. (2023a). *PhotonView Class Reference: IsMine*. https://doc-api.photonengine.com/en/pun/current/class_photon_1_1_pun_1_1_photon_view.html#a67184424cffe2daae9001e06a6192d21

Photon Engine. (2023b). *PhotonView Class Reference: PhotonView*. https://doc-api.photonengine.com/en/pun/current/class_photon_1_1_pun_1_1_photon_view.html

PimDeWitte. (2023). *UnityMainThreadDispatcher.* GitHub. https://github.com/PimDeWitte/UnityMainThreadDispatcher

Syedmodassirali. (2.12.2022). *Client-Server Model*. GeeksforGeeks. https://www.geeksforgeeks.org/client-server-model/

Unity Technologies. (2021). *What is multithreading?*. https://docs.unity3d.com/2021.2/Documentation/Manual/JobSystemMultithreading.html

Unity Technologies. (2022). *Scripting API: AudioRolloffMode*. https://docs.unity3d.com/2022.3/Documentation/ScriptReference/AudioRolloffMode.html

Unity Technologies. (2023a). *Scripting API: AudioClip Create*. https://docs.unity3d.com/2023.3/Documentation/ScriptReference/AudioClip.Create.html

Unity Technologies. (2023b). *Scripting API: AudioSource maxDistance*. https://docs.unity3d.com/2023.3/Documentation/ScriptReference/AudioSource-maxDistance.html

Unity Technologies. (2023c). *Scripting API: AudioSource minDistance*. https://docs.unity3d.com/2023.3/Documentation/ScriptReference/AudioSource-minDistance.html

Unity Technologies. (2023d). *Scripting API: AudioSource spatialBlend*. https://docs.unity3d.com/2023.3/Documentation/ScriptReference/AudioSource-spatialBlend.html

Unity Technologies. (2023e). *Scripting API: Microphone Start*. https://docs.unity3d.com/2023.3/Documentation/ScriptReference/Microphone.Start.html

Unity Technologies. (2023f). *Scripting API: MonoBehaviour*. https://docs.unity3d.com/2023.3/Documentation/ScriptReference/MonoBehaviour.html

Unity Technologies. (2023g). *Scripting API: MonoBehaviour Start*. https://docs.unity3d.com/2023.3/Documentation/ScriptReference/MonoBehaviour.Start.html

Unity Technologies. (2023h). *Scripting API: MonoBehaviour Update*. https://docs.unity3d.com/2023.3/Documentation/ScriptReference/MonoBehaviour.Update.html

Unity Technologies. (2023i). *Manual: Audio Listener*. https://docs.unity3d.com/2023.3/Documentation/Manual/class-AudioListener.html

Unity Technologies. (2023j). *Manual: Audio Source*. https://docs.unity3d.com/2023.3/Documentation/Manual/class-AudioSource.html

Vicente, V. (9.11.2021). *What is Spatial Audio, and How Doest It Work?.* How-To Geek. https://www.howtogeek.com/764288/what-is-spatial-audio-and-how-does-it-work/

websockets. (2023). *ws: a Node.js WebSocket library*. GitHub. https://github.com/websockets/ws

WebSockets Standard. (27.9.2023a). *WebSocket protocol alterations: Opening handshake*. https://websockets.spec.whatwg.org/#websocket-opening-handshake

WebSockets Standard. (27.9.2023b). *The WebSocket interface: Interface definition*. https://websockets.spec.whatwg.org/#interface-definition

Zenva. (24.10.2023). *What is Unity? – A Top Game Engine for Video Games*. GameDev Academy https://gamedevacademy.org/what-is-unity/