VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Anna Leskinen

ÄÄNEN VISUALISOINTI UNITY-PELIMOOTTORILLA

Liiketalous
2020

VAASAN AMMATTIKORKEAKOULU
Liiketalous, tietojenkäsittely

TIIVISTELMÄ

Tekijä	Anna Leskinen
Opinnäytetyön nimi	Äänen visualisointi Unity-pelimoottorilla
Vuosi	2020
Kieli	suomi
Sivumäärä	32 + 1 liite
Ohjaaja	Raija Tuomaala

Tämän opinnäytetyön avulla selvitetään, kuinka ääntä voidaan visualisoida partikkelisysteemien avulla Unity-pelimoottorilla. Tavoitteena on luoda toimiva ääntä visualisoiva ohjelma. Tavoitteena on myös ladata video valmiista ohjelmasta YouTube-palveluun. Opinnäytetyön aihe valittiin kiinnostuksen pohjalta ja aiheen koettiin myös olevan tärkeä, sillä tässä opinnäytetyössä tehtyä projektia voidaan jatkokehittää ja luoda esimerkiksi arkea helpottavia sovelluksia heikkokuuloisille tai kuulovammaisille henkilöille.

Toisessa luvussa puhutaan suurimmaksi osaksi ohjelmistosta, jota projektissa käytetään ja selvitetään, millä logiikalla se toimii. Kolmannessa luvussa selvitetään, mitä ääni on sekä perehdytään digitaaliseen audioon. Neljännessä luvussa käydään läpi projektia ja kuinka se on toteutettu. Viidennessä luvussa on tehty päätelmät ja pohdittu ohjelmiston jatkokehitysmahdollisuuksia. Lähteinä opinnäytetyössä on käytetty erilaisia netistä löytyviä materiaaleja, videoita sekä myös Unityn omaa blogia sekä ohjesivustoja.

Lopputuloksena projektissa syntyi toimiva äänen visualisointiohjelma, josta ladattiin myös video YouTube-palveluun. Projektin valmistuttua ymmärrettiin myös, kuinka kuluttajat ja yritykset voisivat mahdollisesti hyödyntää äänen taajuuksien analysointia.

Avainsanat Ääni, visualisointi, Unity, C#, partikkelisysteemi

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Liiketalous, tietojenkäsittely

ABSTRACT

Author	Anna Leskinen
Title	Sound visualization with Unity game engine
Year	2020
Language	Finnish
Pages	32 + 1 Appendice
Name of Supervisor	Raija Tuomaala

This thesis will examine how sound can be visualized with the Unity game engine's particle systems. The aim is to create a functional sound visualization program. An additional aim is to post a video of the finished program on YouTube. The topic of the thesis was chosen due to a personal interest in the topic, Moreover, the topic was also seen as important because the project can be developed further to create for example apps that would facilitate the life of people who suffer from poor hearing or are hearing-impaired.

The second chapter mainly discusses the software that is used in the project and with what kind of logic it works. In the third chapter, we figure out what sound and digital audio are. The fourth chapter presents the project and how it is executed. The fifth and final chapter draws conclusions and discusses the possibilities of developing the program further. The sources used in this thesis were different kinds of materials from the internet, videos and also Unity's own blog and software manual.

As the result of the project, a functional sound visualization program was created, and a video of the program was uploaded in YouTube. After finishing the project, we understood how analysing sound frequencies could benefit both consumers and businesses.

Keywords Sound, visualization, Unity, C#, particle system

SISÄLLYS

KUVA- JA TAULUKKOLUETTELO

KÄSITELUETTELO

C#	Ohjelmointikieli.
funktio	Ohjelman osa, joka suorittaa jonkin tietyn tehtävän.
HDRP	High-Definition Render Pipeline – Unityn luoma skriptattava renderöintiprosessi.
Initialize-vaihe	Prosessointivaihe, joka määrittää partikkelin, kun se luodaan.
Kuvataajuus	Kuvien määrä sekunnissa, jotka ilmestyvät näytölle.
Node	Tietosolu tai tietoelementti.
Parametri	Ohjelmalle tai funktiolle annettava arvo.
Partikkelisysteemi	Suuri joukko pieniä kappaleita, joiden avulla voidaan simuloida esimerkiksi kaasuja tai nesteitä.
Quad Output -vaihe	Prosessointivaihe, jossa määritellään, kuinka partikkeli renderöidään.
Skripti	Ohjelmointikielellä kirjoitettujen komentojen muodostama kokonaisuus.
Spawn-vaihe	Prosessointivaihe, jossa määritellään, kuinka monta partikkelia halutaan luoda.
Unity Package Manager	Unityn valmis pakettikirjasto, joka sisältää erilaisia toimintoja ja ohjelmistoja.
Update-vaihe	Prosessointivaihe, jossa määritellään, kuinka partikkeli käyttäytyy ajan myötä.

LIITELUETTELO

LIITE 1. Video valmiista projektista (linkki).

1 JOHDANTO

Kuinka ääntä voidaan visualisoida Unity-pelimoottorilla? Miten äänen visualisoinnista voidaan hyötyä? Nämä ovat keskeisiä kysymyksiä, joihin tässä opinnäytetyössä vastataan. Koska ääntä voidaan visualisoida lukemattomilla eri tavoilla, tässä opinnäytetyössä keskitytään äänen visualisointiin partikkelisysteemeillä.

Tämän opinnäytetyön tavoitteena on luoda Unity-pelimoottorilla ohjelma, joka visuaalisten efektien avulla visualisoi ääntä. Tavoitteena on myös ladata valmiista toimivasta ohjelmasta video YouTubeen. Opinnäytetyön teoriaosuudessa käydään läpi asiat, jotka pitää tietää ennen projektin toteutuksen aloittamista. Toiminnallisessa osuudessa esitetään itse projektin toteutus.

Tämän opinnäytetyön aihe valittiin, koska aiheeseen oli jo aikaisempaa kiinnostusta. Aihe valikoitui myös sen tärkeyden takia, sillä sitä voidaan hyödyntää myös muunlaisissa toteutuksissa, kuten esimerkiksi heikkokuuloisille tai muuten kuulovammaisille tarkoitetuissa ohjelmistoissa.

2 TOTEUTUS

Tämän opinnäytetyön projekti on toteutettu Unityn versiolla 2018.3. Unityssä projektityypiksi valittiin High-Definition Render Pipeline eli HDRP, joka sopii erittäin hyvin projektin toteuttamiseen, koska sillä tähdätään korkealaatuiseen visuaaliseen tarkkuuteen (Unity Blog 2018). Tärkeässä osassa oli myös Visual Effect Graph, jota voidaan hyödyntää HDRP-projekteissa.

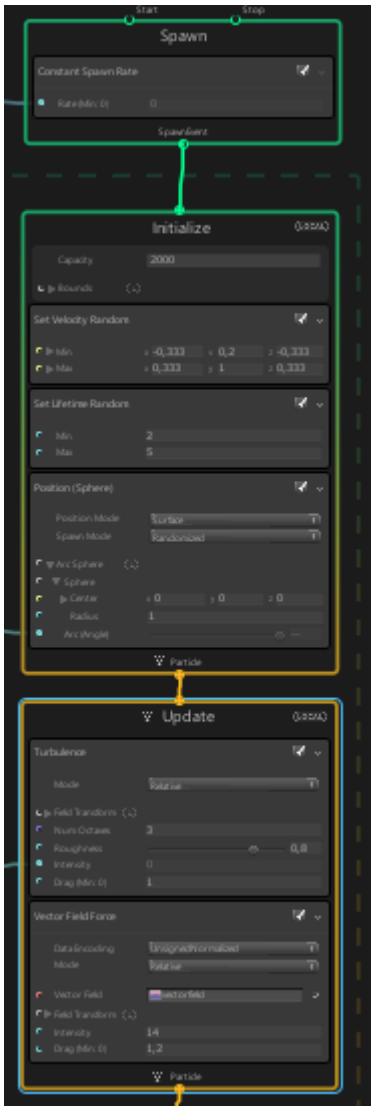
2.1 Visual Effect Graph

Projektissa luodaan visuaalisia efektejä partikkelisysteemeillä. Normaalisti Unityn partikkelisysteemit tuotetaan prosessorin voimalla. Visual Effect Graph tuottaa partikkelit grafiikkaprosessorin avulla. Visual Effect Graphin avulla voidaan siis tuottaa jopa miljoonia partikkeleita samanaikaisesti, kun taas normaalilla partikkelisysteemillä voidaan tuottaa vain tuhansia. Koska projektin visuaaliset efektit perustuvat partikkeleihin ja myös niiden suureen määrään, Visual Effect Graph on loistava valinta niiden tuottamiseen. (Unity, 2020a.)

Visual Effect Graph -paketti asennetaan erikseen HDRP-projektiin Unityn Package Managerin kautta. Pääasiassa Visual Effect Graph -efektien muokkaus tapahtuu Visual Effect Graph -ikkunassa. Tässä ikkunassa voidaan myös asettaa visuaalisille efekteille parametrejä, joiden arvoihin voidaan vaikuttaa C#-skriptien kautta. (Unity, 2020b.)

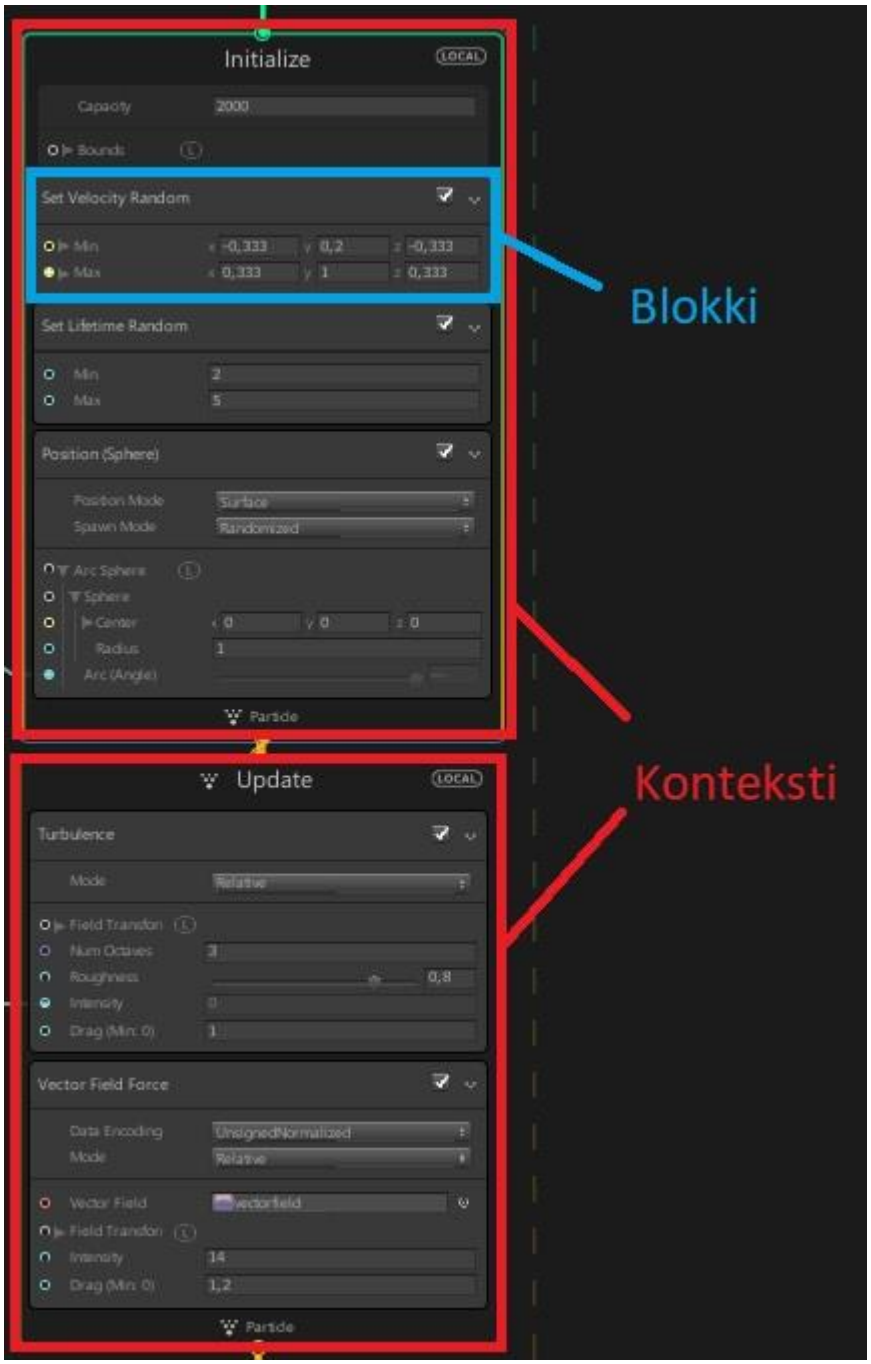
2.2 Prosessointi- ja ominaisuuslogiikka

Visual Effect Graph käyttää kahta selkeää työnkulkua: prosessointi ja ominaisuus. Prosessointilogiikan avulla määritetään efektin elinkaari, linkittämällä muokattavissa olevia efektin vaiheita pystysuunnassa (Kuva 1).



Kuva 1. Kolme kontekstia linkitettynä toisiinsa pystysuunnassa.

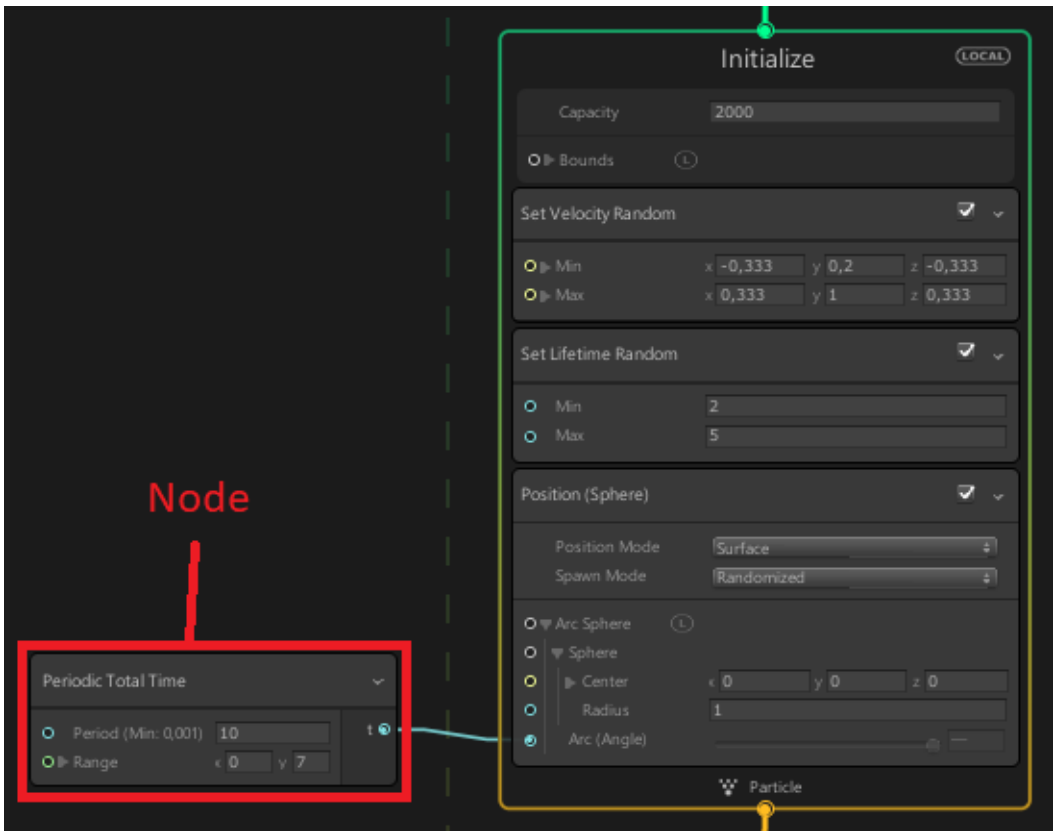
Prosessointi-logiikka määrittää siis visuaalisen efektin prosessoinnin eri vaiheet. Jokainen vaihe koostuu kontekstista (engl. Context). Jokainen konteksti liitetään toiseen sopivaan kontekstiin, joka määrittää kuinka seuraava prosessoinnin vaihe käyttää nykyistä kontekstia. Kontekstit voivat sisältää elementtejä, joita kutsutaan blokeiksi (engl. Block). Jokainen blokki on vastuussa yhdestä toteutettavasta toimenpiteestä. Blokkeja voidaan myös järjestää uudelleen, mikäli niiden prosessointijärjestystä halutaan muuttaa. Blokit suoritetaan kontekstin sisällä järjestyksessä ylhäältä alas. (Unity 2020c.)



Kuva 2. Kaksi kontekstia, joiden sisällä blokkeja.

Ominaisuuslogiikan avulla määritetään partikkeleiden ulkonäkö ja käyttäytyminen, liittämällä esimerkiksi matemaattisia laskutoimituksia eli nodeja tai itse luotuja parametreja horisontaalisesti eri kontekstien blokkeihin (Kuva 3). Visual Effect Graph -paketissa on valmiina erittäin laaja blokki- ja node-kirjasto, josta löytyy valmiina paljon erilaisia blokkeja sekä matemaattisia operaatioita. Lisäksi

voidaan luoda itse parametrejä, joille voidaan antaa arvoja C#-skriptin kautta. (Unity 2020c.)

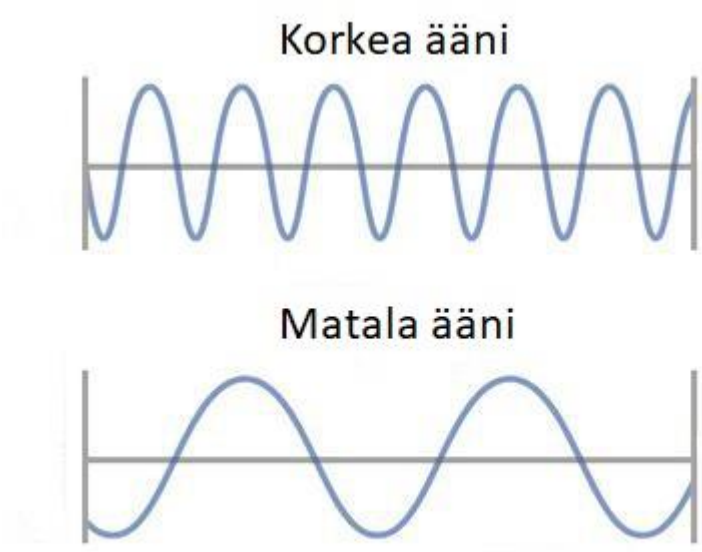


Kuva 3. Node, joka on liitetty horisontaalisesti kontekstin sisällä olevaan blokkiin.

3 MITÄ ÄÄNI ON?

Jotta opinnäytetyön projektia ymmärrettäisiin paremmin, tulee ensin tietää, mitä ääni on. Ääni on värähtelevän objektin luoma paineaalto, jota voidaan käyttää monella eri tapaa. Esimerkiksi jotkut eläimet käyttävät erittäin korkeita ultraääniä navigointiin ja saalistukseen. Ultraääntä käytetään myös esimerkiksi kaikuluotaimissa ja raskaana olevan henkilön sikiön kuvaamiseen. (Tieteen kuvalehti 2018.)

Äänen taajuudella tarkoitetaan paineväräh dysten lukumäärää sekunnissa. Paineväräh dysten lukumäärä sekunnissa määrittää äänen korkeuden. Mitä enemmän väräh dyksiä sekunnissa, sitä korkeampi ääni on. Taajuuden mittayksikkö on hertsi (Hz), joka tarkoittaa väräh dysten määrä/sekunti. Taajuusalue, jonka ihminen voi kuulla on 20–20 000 Hz. Taajuudet, jotka ovat alle 20 Hz eli infraäänet ovat niin matalia, että ihmiskorvalla niitä ei voi kuulla. Taajuudet, jotka ovat yli 20 000 Hz eli ultraäänet ovat niin korkeita, että niitäkään ihminen ei voi kuulla. (Mute 2020.)



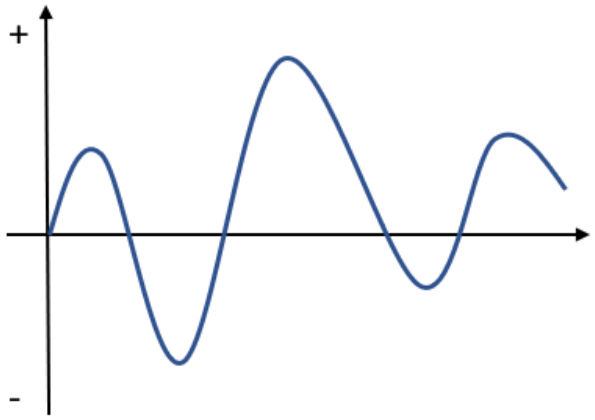
Kuva 4. Korkea ja matala ääni.

Äänenvoimakkuutta eli amplitudia mitataan desibeleissä (dB). Ääntä, jonka voi juuri ja juuri kuulla, kutsutaan kuulokynnykseksi. Kuulokynnyksen äänenvoimakkuus on 0 dB. Äänekäs puhuminen on äänenvoimakkuudeltaan 50-70 dB. Äänet,

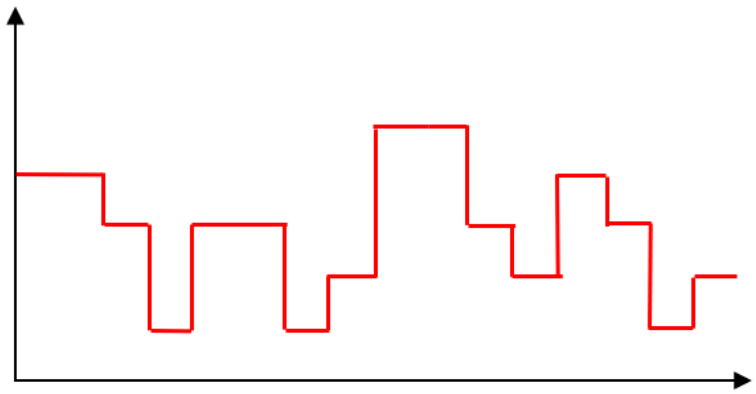
jotka ylittävät kipurajan eli ovat voimakkuudeltaan yli 120 dB, ovat kuulolle erittäin haitallisia ja niille altistumista tulisi välttää. Kuuloon voivat myös vaikuttaa haitallisesti jo 80 dB äänenvoimakkuudet, jos melulle altistuu riittävän pitkään. Esimerkiksi jos jatkuva äänentaso on 85 dB, tällöin jatkuvassa melussa ei kannata olla yli 8 tuntia. Jos taas jatkuva äänentaso olisikin 100 dB, tällöin melussa ei kannata olla 15 minuuttia pidempään. (Strack & Teräsvirta 2010.)

3.1 Digitaalinen audio

Digitaalinen audio on äänen esitysmuoto, joka on nauhoitettu tai muunnettu digitaaliseen muotoon. Analoginen audiosignaali vastaa alkuperäistä ääntä (äänen lähdettä), mutta alkuperäiset ilmanpaineen värähtelyt on muutettu sähköiseksi aaltoliikkeeksi (Kuva 5). Tämä sähköinen aaltoliike on kuitenkin erittäin altis kohinalle ja muille häiriöille. Digitaalinen audiosignaali kuvastaa alkuperäistä ääntä tai analogista audiosignaalia, mutta se on numeraalisessa muodossa (Kuva 6). Digitaalinen audiosignaali ei ole altis häiriöille, niin kuin analoginen audiosignaali on. (Harju 2012.)



Kuva 5. Analoginen audiosignaali.

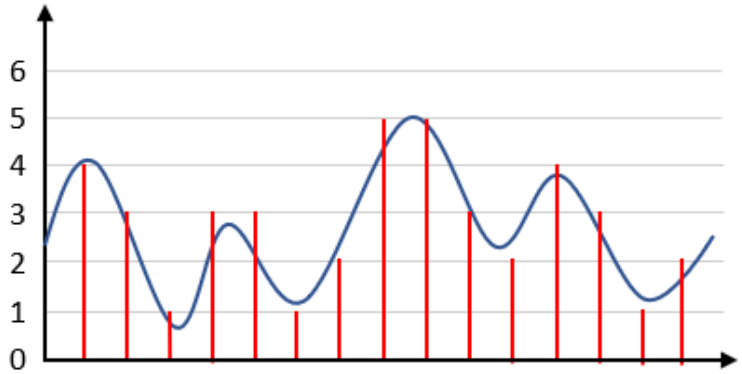


Kuva 6. Digitaalinen audiosignaali.

Bittisyvyydellä kerrotaan, kuinka monta numeroa jokaisen analogisen signaalinäytteen tallennukseen on käytetty digitaalisessa tallenteessa. Bittisyvyyden lisääntyessä yleensä äänenlaatu paranee ja samalla myös tiedoston koko kasvaa. Näytteenottotaajuudella tarkoitetaan sitä, että kuinka monta näytettä analogisesta signaalista otetaan sekunnissa. Kuvassa 7 punaiset viivat kuvastavat ääninäytteiden ottamista analogisesta signaalista. (Sony 2020.)

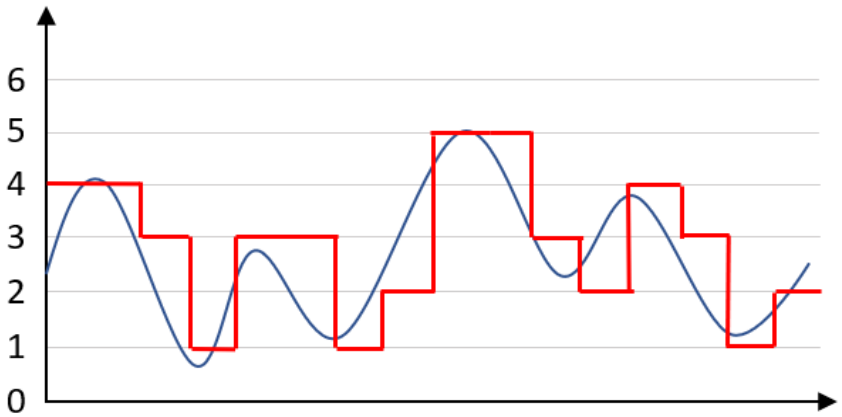
Taulukko 1. Esimerkki bittisyvyyksistä ja näytteenottotaajuuksista.

Tallennusmuoto	**Bittisyvyys**	**Näytteenottotaajuus**
CD-levy	Vakiobittisyvyys on 16	44,1kHz eli 44 100 näytettä sekunnissa.
High-Resolution Audio	Vähintään 24	Vähintään 96 kHz eli 96 000 näytettä sekunnissa.



Kuva 7. Ääninäytteiden otto analogisesta signaalista.

Kun ääninäytteet on otettu analogisesta signaalista, näiden näytteiden avulla voidaan luoda digitaalinen audiosignaali. Kuvassa 8 on esitetty digitaalinen audiosignaali punaisella värillä ja taustalla vastaava analoginen signaali. Tämä digitaalinen audiosignaali on muodostettu kuvassa 7 otettujen ääninäytteiden perusteella.



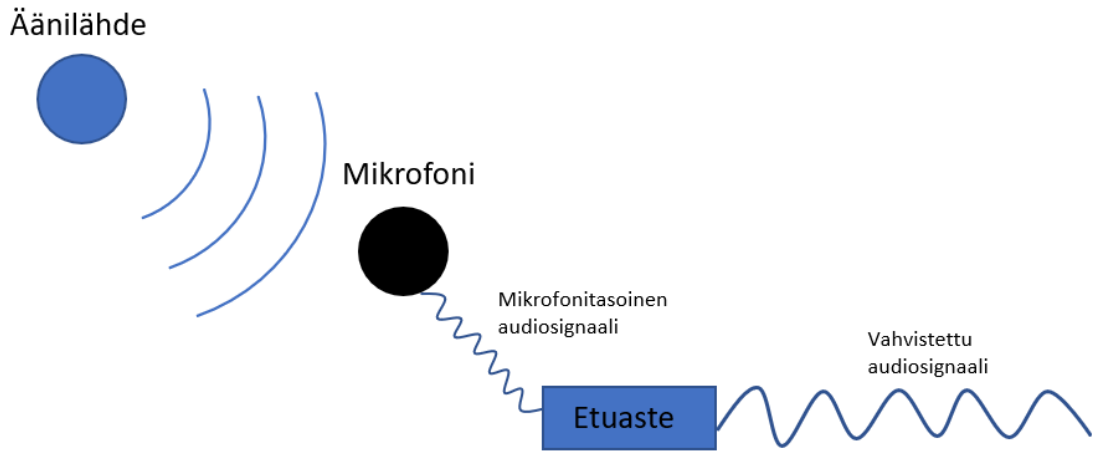
Kuva 8. Ääninäytteistä muodostettu digitaalinen audiosignaali (punaisella).

3.2 Äänen muuntaminen

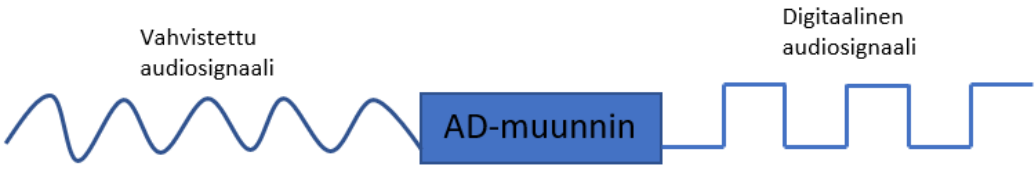
Jotta ääntä voidaan muuntaa digitaaliseen muotoon tai muuntaa digitaalista informaatiota ääneksi, tarvitaan muuntajia. Äänen muuntaja muuntaa ääniaallot johonkin toiseen muotoon, kuten esimerkiksi sähkösignaaliksi. Nykypäivän yleisimpiä äänen muuntajia ovat mikrofonit, kuulokkeet ja kaiuttimet. Nämä voivat muuntaa äänen sähköenergiaksi ja sähköenergian ääneksi. Äänen värähtelyt muunnetaan sähkösignaaleiksi välilaitteiden, kuten liikkuvien magneettikäämien avulla. Kun

ääni on elektronisessa muodossa, signaaleja voidaan käyttää suoraan sähköiseen prosessointiin äänilaitteessa ja muuntaa ne erikoispiirien avulla numeraalisiksi tiedoiksi. Nämä numeraaliset tiedot voidaan siirtää digitaalisesti tietokoneille ja muille digitaalisille laitteille. Tämä toimii myös toisinpäin, jos elektroniset signaalit muutetaan mekaanisesti akustisiksi värähtelyiksi, esimerkiksi vahvistetun sähköisen jännitteen avulla saadaan kaiuttimen kartio liikkumaan sisään ja ulos, jonka tuloksena tuotetaan ääntä. (Kirn 2005, 30.)

AD-muuntimen (engl. analog-to-digital converter) avulla analoginen audiosignaali muutetaan digitaaliseksi audiosignaaliksi. Äänilähteestä mikrofoniin tuleva ääni muutetaan ensin mikrofonin avulla sähköiseksi analogiseksi audiosignaaliksi. Koska tässä vaiheessa audiosignaali on erittäin heikko, se vahvistetaan mikrofonin etuasteessa eli etuvahvistimessa (Kuva 9). Seuraavaksi vahvistettu analoginen audiosignaali ohjataan AD-muuntimeen, jossa muunnin suodattaa pois sellaiset taajuudet, jotka ylittävät sen konvertointikyvyn (Kuva 10). Lopuksi muunnin muuntaa analogisen audiosignaalin digitaaliseksi, ottamalla analogisesta audiosignaalista näytteitä. (Harju 2012.)



Kuva 9. Äänen muuntaminen analogiseksi audiosignaaliksi.



Kuva 10. Vahvistettu analoginen audiosignaali muunnetaan AD-muuntimen avulla digitaaliseksi audiosignaaliksi.

DA-muunnin (engl. digital-to-analog converter) muuntaa vastaavasti digitaalisen audiosignaalin takaisin analogiseksi audiosignaaliksi. Näin ääntä voidaan toistaa esimerkiksi mp3-soittimesta kuulokkeisiin tai tietokoneesta kaiuttimiin. (Harju 2012.)

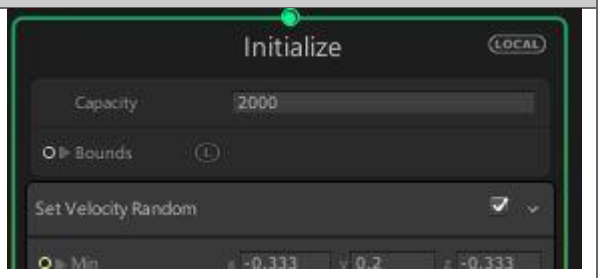
4 KÄYTÄNNÖN OSUUS

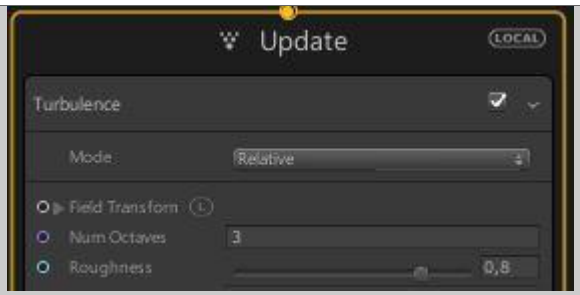
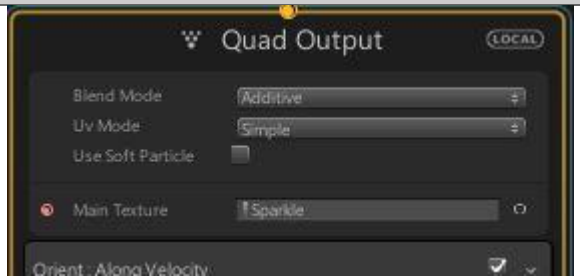
Opinnäytteen projektissa oli kolme pääkohtaa: visuaalisten efektien luonti, äänen analysointi sekä jaottelu ja visuaalisten efektien kontrollointi äänellä. Visuaaliset efektit eli kahdeksan partikkelisysteemiä luotiin ensin ja määriteltiin jokaiselle niistä oma ulkonäkö. Visuaalisten efektien luonnin jälkeen siirryttiin analysoimaan ja jaottelemaan ääntä skriptin kautta. Lopuksi lisättiin myös erillinen skripti visuaalisille efekteille, joka kontrolloi niitä äänen taajuuden mukaan.

4.1 Visuaalisten efektien luonti

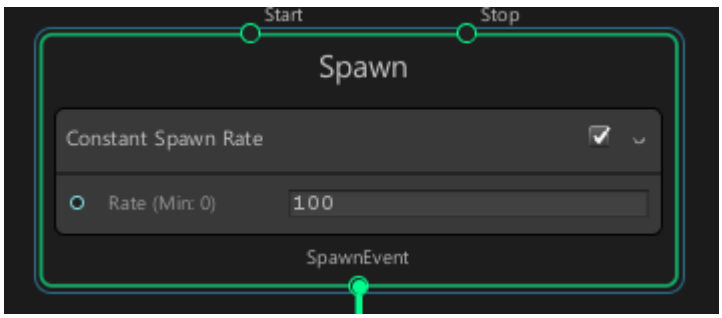
Projekti aloitettiin luomalla visuaaliset efektit. Projektissa käytettiin kahdeksaa erinäköistä partikkelisysteemiä. Partikkelisysteemit ovat muuten samanlaiset, mutta itse partikkelit ovat jokaisessa partikkelisysteemissä eriväriset. Partikkelisysteemeillä on neljä päävaihetta, joiden avulla määritellään esimerkiksi partikkeleiden koko, väri, määrä ja liike. Nämä vaiheet ovat spawn, initialize, update ja quad output.

Taulukko 2. Prosessointivaiheet.

Vaihe	**Kuvaus**	**Kuva**
Spawn	Prosessointivaihe, jossa määritellään, kuinka monta partikkelia halutaan luoda.	
Initialize	Prosessointivaihe, joka määrittää partikkelin, kun se luodaan.	

Update	Prosessointivaihe, jossa määritellään, kuinka partikkeli käyttäytyy ajan myötä.	
Quad Output	Prosessointivaihe, jossa määritellään, kuinka partikkeli renderöidään.	

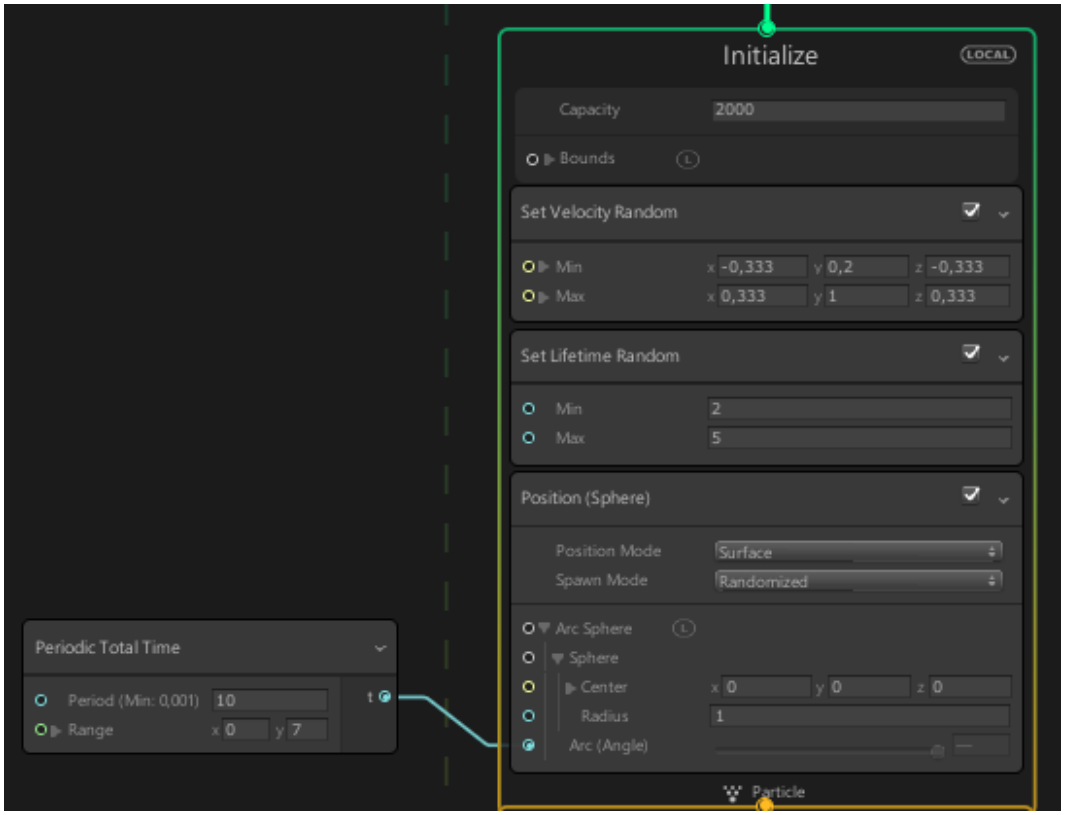
Spawn-vaiheen avulla määritellään nimensä mukaisesti, kuinka monta partikkelia tuotetaan minuutissa ja miten. Tässä projektissa käytettiin tasaista partikkeleiden tuottoa. Tällä tarkoitetaan sitä, että partikkelisysteemi tuottaa minuutin aikana niin monta partikkelia, kuin sille määrätään tuotettavaksi. Testauksen ja partikkelisysteemin ulkonäön muokkaamisen ajaksi se määrättiin tuottamaan 100 partikkelia minuutissa, jotta pystyttiin helposti näkemään, miltä efekti näyttää.



Kuva 11. Spawn-vaihe.

Initialize-vaiheessa määritellään, kuinka monta partikkelia voi olla olemassa samanaikaisesti eli kapasiteetti, partikkeleiden nopeus, elinaika ja sijainti. Kapasiteetiksi partikkelisysteemeille asetettiin 2 000 samanaikaista partikkelia. Partikke-

leiden nopeus on määritetty satunnaiseksi eli nopeudelle on yksinkertaisesti annettu minimi- ja maksimiarvo, joiden väliltä luotu partikkeli saa satunnaisesti nopeutensa. Myös elinaika on määritetty satunnaiseksi eli on annettu taas minimi- ja maksimiarvo, joiden mukaan jokaisen partikkelin elinaika määrittyy satunnaisesti. Viimeisenä tässä vaiheessa on sijainti. Sijainnilla määritellään, mihin partikkelit luodaan. Tässä projektissa käytetään aluetta, jonka säde on yhden mittayksikön kokoinen. Jotta partikkelit eivät joka kerta syntyisi samoista kohdista, lisättiin alueelle ns. kaari, jonka avulla alue saatiin pyörimään. Luotiin siis noodi, jonka avulla voidaan asettaa jaksollinen kokonaisaika ja yhdistettiin se initialize-lohkon kohtaan, jossa kaari määritellään. Yhden pyörähdyksen nopeudeksi asetettiin 10 sekuntia.



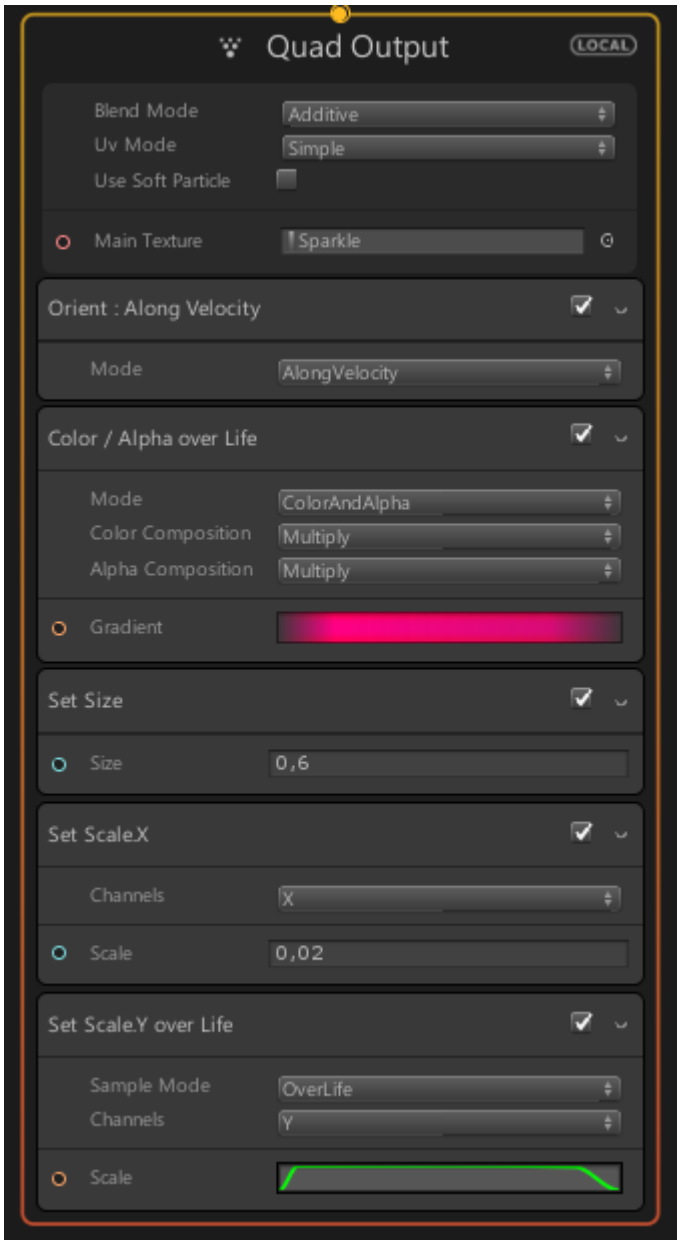
Kuva 12. Initialize-vaihe, johon on yhdistetty node.

Update-vaiheessa on määritelty turbulenssi ja vektorialueen voimakkuus. Nämä molemmat vaikuttavat partikkeleiden liikerataan. Sen sijaan, että partikkelit putoaisivat painovoiman vaikutuksesta suoraan alaspäin, tässä projektissa haluttiin, että partikkelit voivat sulavasti liikkua ilmassa eri suuntiin luonnollisesti ja painot-

tomasti. Tämän vuoksi luotiin turbulenssia ja vektorikenttä, joka työntää partikkeleita eri suuntiin.

Kuva 13. Update-vaihe.

Quad Output -vaiheessa luodaan partikkeleille ulkonäkö. Tässä lohkossa määritellään tekstuuri, orientaatio, väri ja koko. Tekstuuriksi partikkeleille valittiin valmis tekstuuri Unityn kirjastosta nimeltä ”Sparkle”. Tämä valittiin siksi, koska haluttiin että partikkelit olisivat virtaviivaisia ja että niillä olisi perässä pieni häntä. Jos orientaatiota ei olisi asetettu erikseen, partikkelit olisivat olleet koko ajan vain pystysuunnassa ja liikkuneet eri suuntiin. Jotta partikkeleiden liikkeeseen saatiin sulavuutta, partikkelit määriteltiin suuntautumaan liikkumissuunnan mukaisesti. Väri oli ainoa asia, joka määritettiin jokaiselle partikkelisysteemille erilaiseksi. Värinä kaikissa partikkelisysteemeissä käytettiin liukuväriä. Partikkeleiden koko on määritelty pituus- ja leveyssuunnassa. Pituus on määritetty partikkelin elinajalle siten, että sen pituus alussa kasvaa tiettyyn pisteeseen asti ja lopuksi pienenee, kunnes se katoaa. Tällä saavutetaan se, että partikkelit eivät vain yhtäkkiä katoa, vaan ne katoavat sulavasti.



Kuva 14. Quad Output -vaihe.

4.2 Äänen analysointi ja jaottelu

Jotta ääntä voidaan analysoida, tarvitsee sitä varten kirjoittaa skripti. Valmis skripti liitetään tyhjään objektiin, joka nimetään skriptiä kuvaavalla nimellä. Aloitetaan siis luomalla uusi skripti, joka tässä projektissa on luotu nimellä ”AnalyzeAudio”. Koska tässä projektissa ääni on kaikkein tärkeimmässä osassa ja se hallitsee kaikkia efektejä, ei haluta törmätä tilanteeseen, jossa ääntä ei olisikaan.

Siksi heti skriptin alussa lisätään äänilähdekomponentti, jonka kautta haluttu musiikkikappale voidaan lisätä Unityn editorin kautta.

Unityssä on valmiina kaksi funktiota: Start ja Update. Start-funktio suoritetaan, kun ohjelma käynnistetään ja Update-funktio suoritetaan kerran jokaisen framen kohdalla. Asia, jonka haluamme tehdä ensimmäisenä, kun ohjelma käynnistyy, on hakea äänitiedosto. Se onnistuu GetComponent-komennolla, joka lisätään Start-funktioon. Suurin osa koodista tulee Update-funktioon. Koska halutaan, että Update-funktio pysyy siistinä ja selkeänä, kaikki toiminnot lisätään sinne funktioina. Tämän avulla myös helpotetaan, jos jotain toimintoa ei haluta käyttää, sen funktion kutsu voidaan vain helposti kommentoida pois Update-funktiosta.

```
// Use this for initialization
0 references
void Start () {

    aanilahde = GetComponent<AudioSource>();
}

// Update is called once per frame
0 references
void Update () {
    HaeAaniArvot();
    TeeAaniLohkot();
}
```

Kuva 15. Start- ja Update-funktiot.

Projektissa Update-funktion sisällä on kaksi funktiota, joista ensimmäinen hakee äänen arvot ohjelmalle annetusta äänitiedostosta. HaeAaniArvot-funktio hakee, jokaisen framen kohdalla äänen arvot ja tallentaa ne array-taulukkoon. Äänen arvot on mahdollista saada GetSpectrumData-komennolla. GetSpectrumData-komennolla on kolme eri parametria: samples, channel ja window. Samples-parametriin asetetaan array-taulukko, jonne äänen arvot syötetään. Tämän array-taulukon pituus tulee olla potenssiin 2, eli esimerkiksi 128, 256 tai 512. Pituuden tulee olla vähintään 64 ja enintään 8 192. Tässä projektissa array-taulukon pituudeksi on laitettu 512. Channel-parametrilla viitataan, mistä array-taulukon solusta aloitetaan tallennus. Koska array-taulukon ensimmäinen solu on nolla, asetetaan siis channel-parametriksi 0 (nolla). Window-parametrin avulla vähennetään sig-

naalivuotoja eri taajuuskaistojen välillä. Mitä monimutkaisempi window, sitä parempi laatu. Pitää kuitenkin pitää mielessä, että monimutkaisempi window on myös hitaampi. (Unity 2020d.)

Toinen Update-funktion sisällä olevista funktioista jakaa aiemmin haetut 512 ääniarvoa kahdeksaan äänilohkoon. Teemme tämän, koska 512 eri äänilohkoa on liikaa projektiimme. Ääniarvot pitää jaotella kahdeksaksi eri äänilohkoksi harkiten. Matalien äänien arvot ovat huomattavasti suurempia, kuin korkeiden äänten arvot. Jos katsoisimme reaaliajassa kaikkia 512:n ääninäytteen arvoja, huomaisimme nopeasti, että yli puolet korkeista äänistä tuskin liikahtavatkaan verrattuna mataliin ääniin. Siksi taajuudet on jaettava niin, että sisällytetään enemmän korkeita taajuuksia yhteen äänilohkoon kuin matalia taajuuksia. (Peer Play 2016.)

Yksi tavallisimmista taajuuksista, jota mp3-tiedosto tukee, on 22 050 hertsiä. Käytämme siis tätä arvoa, kun suhteutamme jaon kahdeksalle eri äänilohkolle. Ei haittaa, vaikka arvo olisi todellisuudessa korkeampi tai matalampi, sillä suhde pysyy jaossa kuitenkin samana. Määritellään ensin, kuinka monta hertsiä yksi ääninäyte sisältää tällä hetkellä: 22050 / 512 = n. 43. Yksi ääninäyte sisältää tällä hetkellä noin 43 hertsiä. Jaotellaan ääninäytteet seuraavalla tavalla:

Taulukko 3. Hertsien jako kahdeksalle äänilohkolle.

Äänilohko	**Kerroin**	**Hertsimäärä**
Lohko 0	43^{2}	86
Lohko 1	43^{4}	172
Lohko 2	43^{8}	344
Lohko 3	43^{16}	688
Lohko 4	43^{32}	1 376
Lohko 5	43^{64}	2 752

Lohko 6	43^{128}	5 504
Lohko 7	43^{256}	11 008

Edellä olevan taulukon mukaan jaotellaan siis ensimmäiselle äänilohkolle kaksi ääninäytettä, toiselle neljä, kolmannelle kahdeksan jne., jotta hertsien määrä yhdessä äänilohkossa saadaan asteittain suuremmaksi. Jos jäämme tähän, ääninäytteistä jää käyttämättä kaksi, koska 2+4+8+16+32+64+128+256 = 510. Skriptissä lisätään lopuksi vielä nämä kaksi puuttuvaa ääninäytettä viimeiselle äänilohkolle. (Peer Play 2016.)

4.3 Visuaalisten efektien kontrollointi äänellä

Kun on onnistuneesti haettu äänidata ja jaoteltu se kahdeksaan äänilohkoon, on aika siirtyä visuaalisten efektien kontrollointiin äänellä. Visuaalisia efektejä kontrolloidaan muokkaamalla partikkeleiden syntymisen määrää, kokoa ja nopeutta. Jotta voimme vaikuttaa näihin, täytyy aloittaa muokkaamalla hieman aikaisemmin luomiamme visuaalisia efektejä. Visuaalisille efekteille voidaan luoda omia parametreja. Näihin parametreihin voidaan vaikuttaa skriptistä, esimerkiksi asettamalla niille eri arvoja. Parametrit voidaan myös liittää aikaisemmin puhuttuihin visuaalisten efektien vaiheiden sisällä oleviin blokkeihin ja näin voidaan vaikuttaa johonkin tiettyyn blokin arvoon skriptin kautta.

Partikkeleiden syntymisen määrää kontrolloidaan parametrilla ”spawnRate”. Tämä yhdistetään Spawn-vaiheen Rate-kohtaan. Seuraavaksi luodaan partikkeleiden kokoa kontrolloivat ”size” ja ”scale” parametrit. Ne liitetään samannimisiin kohtiin Quad Output -vaiheessa. Viimeisenä luodaan intensity-parametri, jolla kontrolloidaan partikkeleiden nopeutta. Se liitetään Update-vaiheen Intensity-kohtaan.

Kun parametrit on lisätty, voidaan aloittaa siirtämällä hierarkiaan kaikki kahdeksan visuaalista efektiä. Luodaan myös uusi skripti, joka on tässä projektissa nimeltä ”Visuals”. Tämä skripti liitetään jokaiseen visuaaliseen efektiin. Tämä sama skripti toteuttaa siis kaikkien kahdeksan visuaalisen efektin toiminnan. Jotta

voimme skriptin kautta saada yhteyden visuaalisiin efekteihin, haetaan Start-funktiossa GetComponent-komennolla visuaalinen efekti, joka on Unityn editorissa asetettu. Tässäkin skriptissä Update-funktion sisällä on kaksi funktiota, joista ensimmäinen on AsetaKoko-funktio ja toinen KiihtyvyysBassolla-funktio.

AsetaKoko-funktiossa määritellään syntyvien partikkeleiden määrä sekä niiden koko. Syntyvien partikkeleiden määrää kontrolloidaan tässä projektissa, hakemalla AnalyzeAudio-skriptistä äänilohkon arvo ja asettamalla sille kerroin. Mitä isompi arvo äänilohkolla on, sitä enemmän partikkeleita kyseisen äänilohkon partikkelisysteemi tuottaa. Partikkeleiden kokoa muokataan siten, että kun ääntä ei ole, partikkeleiden koko on 0 ja kun ääntä on, partikkeleiden koko kasvaa. Koska haluamme, että partikkelit ilmestyvät ja katoavat sulavasti, käytetään kokoa määritellessä Mathf.Lerp-komentoa.

```
float amn = AnalyzeAudio.aanilohko[lohko] * kerroin;
int visualamn = (int)amn;

visualEffect.SetInt(SPAWNRATE, visualamn);

float currentSize = visualEffect.GetFloat(SIZE);
float currentScale = visualEffect.GetFloat(SCALE);


if (AnalyzeAudio.aanilohko[lohko] >= raja)
{

    visualEffect.SetFloat(SIZE, Mathf.Lerp(currentSize, 0.6f, Time.deltaTime * 15));
    visualEffect.SetFloat(SCALE, Mathf.Lerp(currentScale, 0.02f, Time.deltaTime * 15));

}
else
{
    visualEffect.SetFloat(SIZE, Mathf.Lerp(currentSize, 0.0f, Time.deltaTime * 10));
    visualEffect.SetFloat(SCALE, Mathf.Lerp(currentScale, 0.0f, Time.deltaTime * 10));
}
```

Kuva 16. Partikkeleiden määrän ja koon kontrollointi.

KiihtyvyysBassolla-funktion tarkoituksena on nopeuttaa partikkeleiden vauhtia aina, kun matalimmalla äänilohkolla on arvoja. Partikkeleiden vauhti myös hidastuu, kun matalimmalla äänilohkolla ei ole arvoja. Tämä on toteutettu myös Mathf.Lerp-komennolla, jotta partikkeleiden vauhdin muutokset olisivat sulavia.

Mathf.Lerp-komennolla saadaan aikaan lineaarinen interpolaatio. Lineaarisen interpolaation avulla voidaan löytää arvo, joka on tietty prosentti kahden eri numeron väliltä. Mathf.Lerp-komennolle annetaan siis kolme parametria: lähtöarvo,

kohdearvo ja interpolaation arvo. Interpolaation arvo annetaan väliltä 0-1, jossa 0 on lähtöarvo ja 1 on kohdearvo. Jos interpolaation arvoksi annettaisiin 0,5, silloin se tarkoittaisi arvoa lähtöarvon ja kohdearvon keskeltä. Esimerkiksi jos halutaan tehdä interpolaatio numeroiden 2 ja 4 välillä ja interpolaation arvoksi annetaan 0,5, tulokseksi saataisiin 3. (Unity Learn 2020.)

Tässä projektissa kuitenkin käytettiin interpolaation arvon tilalla aikaa. Kun arvon tilalla käytetään aikaa, muuttuu interpolaatio siten, että lähtöarvo muuttuu kohdearvoksi tasaisesti sille annetussa ajassa. Time.deltaTime-komennon avulla saadaan aika, joka on kulunut viime framesta. Koska ohjelman kuvataajuus eli fps (frames per second) on 60, tällöin Time.deltaTime-komennon arvoksi saadaan noin 0,016 sekuntia (1 / 60 = n. 0,016). Tämä arvo on vielä niin pieni, että sulavaa efektiä ei synny. Siksi projektissa esimerkiksi partikkeleiden koon suurentamisessa, tämä arvo on kerrottu vielä 15, jotta aikaa saadaan pidemmäksi ja koon vaihdos sulavammaksi. (Unity Learn 2020.)

5 PÄÄTELMÄT

Opinnäytetyön projektiosioon käytettiin aikaa noin kuukausi ja asiat, jotka projektissa oli suunniteltu toteutettavaksi, saatiin valmiiksi suunnilleen siinä ajassa. Opinnäytetyö aloitettiin tekemällä itse projekti, jonka toteuttamista varten luettiin pääasiassa Unityn käyttöohjeita. Projektin toteuttamisen ohessa kirjoitettiin projektista teoriaa, jonka jälkeen kirjoitettiin muu teoria, joka johdattaa lukijan projektiin. Valmiista projektista ladattiin myös video YouTubeen, jonka linkki löytyy tämän opinnäytetyön liitteistä.

Tämän opinnäytetyön aikana opin erityisen paljon Unitystä ja sen partikkelisysteemeistä. Yksi projektin haastavimmista asioista oli saada yhteys partikkelisysteemeihin koodin kautta. Sen tutkimiseen meni monia tunteja, mutta se tuotti myös paljon tulosta oppimismielessä. Vaikka projektissa tehtiinkin muutamia jo ennestään tuttuja asioita, niistäkin oppi samalla lisää. Projektin aikana opin myös tekemään kaikki update-funktion sisälle tulevat asiat omina funktioinaan, joiden kutsu sitten laitettiin update-funktion sisälle. Näin koodista oli erittäin helppoa kommentoida vain yksi rivi pois (funktion kutsu), jos haluttiin testata jotain tiettyä osaa koodista yksistään. Aiheeseen liittyvä sanasto on pääosin englanniksi ja jos ne käännettäisiin suomeksi, niissä ei välttämättä olisi mitään järkeä. Tämän takia opinnäytetyössä useista termeistä puhutaan niiden englanninkielisillä nimillä. Näitä termejä oli myös erittäin vaikea kääntää suomeksi ja se onkin toinen syy, minkä takia englanninkieliset termit ovat käytössä.

5.1 Jatkokehitysmahdollisuudet

Äänen visualisoinnilla voidaan parantaa kuuntelijan kokemusta, varsinkin heikosti kuulevien tai muuten kuulovammaisten henkilöiden kokemusta. Koska projektin tuotos kerää siihen syötetyn äänitiedoston taajuusarvoja, voidaan sitä hyödyntää myös muunlaisien sovellusten kehittämisessä. Esimerkiksi äänitiedoston sijaan voidaan äänenlähteenä käyttää mikrofonista tulevaa syötettä, jonka kautta voitaisiin luoda mobiilisovellus, joka kertoo heikosti kuuleville henkilöille, kuinka kova meteli ympärillä on. Mobiilisovellus voisi myös viestiä, kuinka kovaa käyttäjä puhuu, sillä kuulovammaisilla henkilöillä voi olla vaikeuksia pitää oma puhe-

äänentaso sopivana. Jos kuuntelijan kokemusta haluttaisiin viedä vielä askel pidemmälle, voitaisiin luoda samantapainen ympäristö myös virtuaalitodellisuuteen.

Äänen analysointia voidaan myös hyödyntää monipuolisesti. Äänen analysoinnin avulla voitaisiin esimerkiksi ennakoida erilaisten tehtaiden tuotantovälineiden rikkoutumista tai vioittumista. Jos esimerkiksi moottorin käynti muuttuisi yhtäkkiä hieman erilaiseksi, voidaan se huomata sen käyntiääntä analysoivan ohjelmiston kautta ennen moottorin hajoamista. Näin tehdas voi ennakoida ja jopa korjata kyseisen moottorin ennen sen hajoamista.

Äänen analysointia hyödyntävät sovellukset voivat hyödyttää myös peruskuluttajia. Nykyään useat kodinkoneet saattavat antaa virhekoodit erilaisina piippauksina tai ääninä itse koodin sijaan. Tämä voi olla kuluttajille erittäin turhauttavaa, sillä usein näihin piippauksiin tai ääniin ei löydy netistä suoraa vastausta. Kodinkonevalmistajilla voisi olla esimerkiksi mobiilisovellus, joka analysoisi virhekoodin piippaukset tai äänen ja antaisi niiden perusteella kuluttajalle selkeät ohjeet, mikä laitteessa on vikana ja tarvitseeko välttämättä valmistajan tukeen edes olla yhteydessä. Näin voitaisiin välttää ”turhat” yhteydenotot asiakastukeen ja keskittyä asiakkaisiin, jotka oikeasti tarvitsevat asiakastuen palvelua. Näin voitaisiin säästää niin asiakkaan aikaa kuin myös kodinkoneen valmistajien asiakastukihenkilöiden aikaa. Myös turhat huoltokäynnit voisi olla mahdollista välttää tällaisen sovelluksen avulla.

LÄHTEET

Harju, M. 2012. Digitaaliaudion perusteet. Viitattu 10.5.2020.
https://aaltomuoto.wordpress.com/aani/aanisuunnittelu-ja-studiotyo
/digitaaliaudion-perusteet/

Kirn, P. 2005. Real World Digital Audio. Peachpit Press. Viitattu 10.5.2020.
https://learning.oreilly.com /library/view/real-world-digital/0321304608/?ar

Mute. 2020. Musiikin teoriaa webissä. Viitattu 5.2.2020.
https://www15.uta.fi/arkisto/mustut/mute/aai01.htm

Peer Play. 2016. Audio Visualization - Unity/C# Tutorial [Part 4 - Eight Frequency Bands]. Viitattu 6.2.2020.
https://youtu.be/mHk3ZiKNH48?t=112

Sony. 2020. Digitaalisen äänentoiston perusopas. Viitattu 10.5.2020.
https://www.sony.fi/electronics/support/understanding-digital-audio

Strack, J. & Teräsvirta, L. 2010. Melua voi mitata. Viitattu 3.5.2020.
https://www.kouluntervеyskirjasto.fi/aihe/opettajalle-ja-opiskeluhuollolle
/melu/mel00003

Tieteen kuvalehti. 2018. Opi lisää äänestä. Viitattu 5.2.2020.
https://tieku.fi/fysiikka/aani-on-ilman-varahtelyja

Unity. 2020a. Visual Effect Graph. Viitattu 14.4.2020.
https://unity.com/visual-effect-graph

Unity. 2020b. The Visual Effect Graph window. Viitattu 14.4.2020.
https://docs.unity3d.com/Packages/com.unity.visualeffectgraph@8.1/manual
/VisualEffectGraphWindow.html

Unity. 2020c. Visual Effect Graph Logic. Viitattu 17.4.2020.
https://docs.unity3d.com/Packages/com.unity.visualeffectgraph@8.1/manual
/GraphLogicAndPhilosophy.html

Unity. 2020d. AudioSource.GetSpectrumData. Viitattu 5.2.2020.
https://docs.unity3d.com/ScriptReference/AudioSource.GetSpectrumData.html

Unity Blog. 2018. The High Definition Render Pipeline: Getting Started Guide for Artists. Viitattu 12.4.2020.
https://blogs.unity3d.com/2018/09/24
/the-high-definition-render-pipeline-getting-started-guide-for-artists/

Unity Learn. 2020. Linear Interpolation. Viitattu 10.5.2020. https://learn.unity.com/tutorial/linear-interpolation#5c8a48bdedbc2a001f47cef6

LIITTEET

Liite 1. Video valmiista projektista (linkki).
https://www.youtube.com/watch?v=ztpt64vLZpw

Videon musiikki: TheFatRat & Anjulie – Close To The Sun

Videon musiikkia saa vapaasti käyttää YouTubessa artistin luvalla.