KARELIA-AMMATTIKORKEAKOULU
Tietojenkäsittelyn koulutusohjelma

Roope Timonen

ÄÄNIEN OHJELMALLINEN TOTEUTUS UNITY3D-PELIMOOTTORILLA

Opinnäytetyö
Tammikuu 2016

Karelia AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Tammikuu 2016
Tietojenkäsittelyn koulutusohjelma

Karjalankatu 3
80200 JOENSUU
013 260 600

Tekijä(t)
Roope Timonen

Nimeke
Äänien ohjelmallinen toteutus Unity3D-pelimoottorilla

Toimeksiantaja
-

Tiivistelmä

Tässä opinnäytetyössä käydään läpi Unity3D-pelimoottorin ääniominaisuuksia ja niiden käyttämistä kaksiulotteisessa tasohyppelyssä. Työn tavoitteena on käydä läpi peliääniin ja musiikkiin liittyviä ongelmia ja haasteita, joita ohjelmoija kohtaa tasohyppelyä kehittäessään. Työssä tuodaan myös esille pelin kehityksessä käytettävät äänityökalut ja niiden toimintaperiaatteet syventyen pelimoottorin uusimman version tarjoamiin ominaisuuksiin.

Näihin kysymyksiin etsitään vastauksia toiminnallisessa osuudessa käytetyn Lost Home -pelin kautta, jonka esittelemisen ja lyhyen kehittämisprosessin kuvaamisen jälkeen siirrytään tarkastelemaan sen kehityksessä nousseita asioita. Ensin peliäänien ongelmia tuodaan esille kuvaamalla pelihahmo kerrallaan ratkaisuja, joita tehtiin tavoitteisiin pääsemiseksi, minkä jälkeen vuoronsa saa pelimusiikki. Opinnäytetyössä pohditaan lyhyesti myös muita vaihtoehtoja äänien käsittelyyn Unity3D-pelimoottorin ulkopuolelta ja kerrotaan, miksi niihin ei kuitenkaan tartuttu. Lopuksi pohditaan lyhyesti pelimoottorin uusien työkalujen hyödyllisyyttä pelejä kehittämisessä.

Työn lopputuloksena saatiin erilaisten peliäänien ja musiikin käsittelyyn toimivia ratkaisuja, joita Lost Home -pelin kehityksessä tullaan käyttämään tulevaisuudessa.

Kieli
suomi

Sivuja 46
Liitteet 0
Liitesivumäärä 0

Asiasanat
tasohyppely, ääniefektit, Unity3D

THESIS
January 2016
Degree Programme in Business Information Technology
Karjalankatu 3
FI 80220 JOENSUU
FINLAND
013 260 600

Author(s)
Roope Timonen

Title
Programmatic Implementation of Sounds with Unity3D Game Engine

Commissioned by
-

Abstract

The thesis focuses on Unity3D game engine's audio features and how to use them in a two-dimensional platform game. The thesis aims to bring forth possible issues with game sound and music that a programmer might encounter while developing a platform game. Audio tools used in game development and their operational principles are introduced while taking a closer look into the features provided by the newest version of the game engine.

After introducing and briefly describing history of a game called Lost Home, the functional part of thesis observes the problems that occured during the game's development. The problems and solutions with game sounds are introduced first, describing them one game character at the time, after which music of the game is discussed. The thesis also presents a few alternatives to Unity3D's own audio system and a reason why they were ultimately not used. Lastly, what has been learned is reflected and the usability of the Unity3D's new tools in game development is briefly contemplated.

The end results are working solutions for using different game sounds and music that will be used in future development of Lost Home.

Language
Finnish

Pages 46
Appendices 0
Pages of Appendices 0

Keywords

platform game, sound effects, Unity3D

Sisältö

1 Johdanto

Videopelit ovat hyvin monimediainen taiteenmuoto. Ne saavat ihmiset ajattelemaan ja tuntemaan asioita monien erilaisten aistien kautta. Yksi näistä aisteista on tietenkin kuuloaisti. Äänet peleissä ovat aivan yhtä tärkeitä kuin mikä tahansa muukin pelin osa-alue. Ilman niitä peli tuntuu keskeneräiseltä, minkä takia onkin tärkeää että äänet, joita peliin asetellaan, kuulostavat sopivilta ja luonnollisilta. Kun luolaan mentäessä äänet muuttuvat kaikuviksi, vuorelle noustaessa tuulen ääni kuuluu kovempaa ja vaaran ilmaantuessa musiikki muuttuu rauhallisesta intensiivisemmäksi, pelaaja saa vahvemman kontaktin pelimaailmaan (Huiberts 2010, 45). Peleissä voi olla hyvinkin laaja äänimaailma aina musiikista erilaisten alueiden kävelyääniin saakka.

Opinnäytetyön aiheen valitsin sekä mielenkiinnosta että tarpeesta. Olen aina ollut kiinnostunut musiikin tekemisestä ja vuosia sitten Star Wars: Republic Commando -pelin lisämateriaalista löytynyt erilaisia käytännön ääniefektejä (esimerkiksi askelia tai vaatteiden kahinaa) tekevän foley-artistin haastattelu loi aikanaan mielenkiinnon myös foley-ääniä kohtaan. Vaikka opinnäytetyön aihe ei itsessään liity äänien tekemiseen, toivon sen madaltavan kynnystä tutustua myöhemmin äänituotantoon liittyviin asioihin. Toiminnallisessa osuudessa käytetty Lost Home -tasohyppelypeli, jota olen ohjelmoinut noin vuoden, oli vailla kunnollisia ääniä ja uusi Unityn versio, joka toi mukanaan aivan uudenlaisia ominaisuuksia äänien käsittelyyn, oli juuri julkaistu kun olin valitsemassa aihetta. Olin jo aikaisemmin leikitellyt ajatuksella siitä, että voisin liittää opinnäytetyöni kyseiseen projektiin, joten mahdollisuuden ilmaantuessa päätin tarttua siihen.

Raportissa tutustutaan äänien ohjelmointiin uudesta näkökulmasta Unityn uuden komponentin, Audio Mixerin kanssa. Vaikka äänien ohjelmointi ja paikalleen laittaminen voivat olla aiheena läpikäytyjä, käsitellään niitä kaikkia Audio Mixerillä, joko yksinkertaisesti äänenvoimakkuuden säätämiseen tai monimutkaisemmin erilaisten efektien käyttämiseen.

Opinnäytetyö käsittelee Unityn äänityökaluja sekä niiden käyttöä ja kertoo, mitä asioita ohjelmoijan tulee huomioida työskennellessään musiikin ja foley-äänien

kanssa 2D-pelissä. Ensimmäiseksi raportissa käydään yksi kerrallaan läpi Unityn äänikomponentteja sekä esitellään niiden erilaisia asetuksia ja tarkoituksia Unityn sisällä. Seuraavaksi esitellään toiminnallisessa osuudessa kehitettyä Lost Home -peliä, kuinka se sai alkunsa ja minkälaisen prosessin kautta tähän pisteeseen on päädytty. Tämän jälkeen seuraa itse toiminnallinen osuus, jossa kerrotaan äänien ohjelmoinnin prosessissa pelkistä tiedostoista pelin toimiviksi osasiksi, sekä esille tulleissa ongelmista ja niiden ratkaisemista, sekä katsotaan mitä uutta Unity 5 toi pelimoottorin äänitoimintoihin. Lopuksi katsotaan muutamia esimerkkejä vaihtoehtoisista äänienkäsittelyjärjestelmistä, pohditaan mitä on opittu ja kuinka näitä taitoja tullaan käyttämään vastaisuudessa.

2 Komponentit

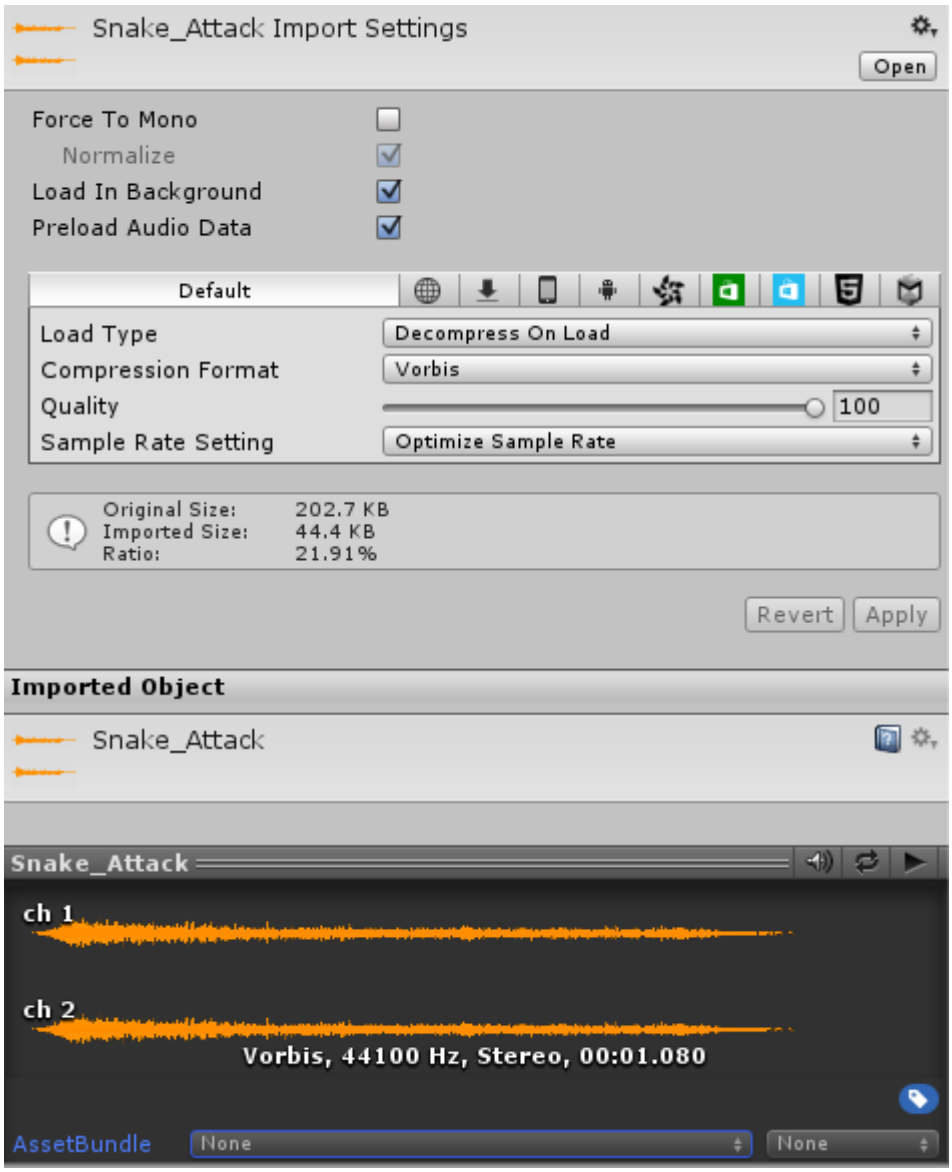
Tässä luvussa käydään läpi Unityn äänikomponentteja ja niiden toiminnollisuutta. Komponentit käydään läpi yksi kerrallaan ja niiden tarkoituksia kerrotaan enimmäkseen teknisellä tasolla, mutta myös esimerkkejä niiden käyttötarkoituksista annetaan. Lopuksi puhutaan Unity 5:n Audio Mixeristä ja siitä, miten se on muokannut jo valmiiksi hyvin monipuolista ja helppoa pelimoottoria vieläkin sujuvammaksi käyttää (Geig 2013).

Pelin rakentaminen Unitylla perustuu komponentteihin. Komponentit ovat valmiiksi tehtyjä palasia, joilla rakennetaan erilaisia peliobjekteja. Nämä komponentit suorittavat tavallisesti vain yhtä niille tarkoitettua asiaa, kuten antavat peliobjektille fysikaalisia ominaisuuksia tai mahdollisuuden törmäystarkistukseen. Myös erilaiset koodinpätkät luokitellaan komponenteiksi Unityssa. Erilaisia komponentteja yhdistelemällä on mahdollista luoda useita erilaisia peliobjekteja, jotka toimivat kaikki ilman erillistä ohjelmointia. Jotkin komponentit toimivat ilman toisen komponentin vaikutusta, jotkut taas tarvitsevat komponentteja, jotka tukevat niiden toimintaa. Unityyn on mahdollista luoda omia komponentteja, mutta opinnäytetyössä keskitytään valmiisiin komponentteihin. (Porter 2013)

2.1 Äänikomponentit

2.1.1 Audio Clip

Audio Clip -komponentti (Kuva 1) sisältää yhden äänitiedoston tiedot (Unity Technologies 2015a). Kaikki Unityyn tuotavat äänitiedostot ovat Audio Clip-komponentteja, joita voidaan muokata niiden käyttötarkoituksen mukaan (taulukko 1). Audio Clip voi olla mikä tahansa pelin ääni, aina lyhyestä askeläänestä kolme minuuttia kestävään taustamusiikkiin.



Kuva 1. Audio Clip -komponentti (Kuva: Roope Timonen).

Taulukko 1. Audio Clip -komponentin asetukset (Unity Technologies 2015a).

Load Type	Äänen lataustyyppi. Tällä voidaan valita millä tavalla sen lataus käsitellään ohjelmaa suorittaessa.
Compression Format	Kompressointiformaatti. Tällä voidaan säätää laatua äänen käyttötarkoituksen perusteella, esimerkiksi lyhyellä äänellä voidaan käyttää kompressointia, joka pitää äänitiedoston hyvälaatuisena muttei pienennä sitä.
Sample Rate Setting	Näytteenottotaajuus. Tällä määritetään pidetäänkö se alkuperäisenä, optimoidaanko se vai asetetaanko se manuaalisesti.
Force To Mono	Muutetaanko ääni yksikanavaiseksi?
Load In Background	Ladataanko ääni taustalla ohjelmaa suorittaessa sen sijaan että se ladattaisiin ennen ohjelman suorittamista?
Preload Audio Data	Ladaanko äänen tiedot ohjelman suorituksen alussa sen sijaan että se ladattaisiin kun ääni toistetaan ensimmäisen kerran?

Lisäksi joillain pakkausasetuksilla voidaan määrittää laatuasetuksia, jotka vaikuttavat tiedoston kokoon ja äänen laatuun. Ääniformaatit joita Unity tukee, ovat AIFF (Audio Interchange File Format), WAV (Waveform Audio Format), MP3 (MPEG-1 Audio Layer 3) ja Ogg.

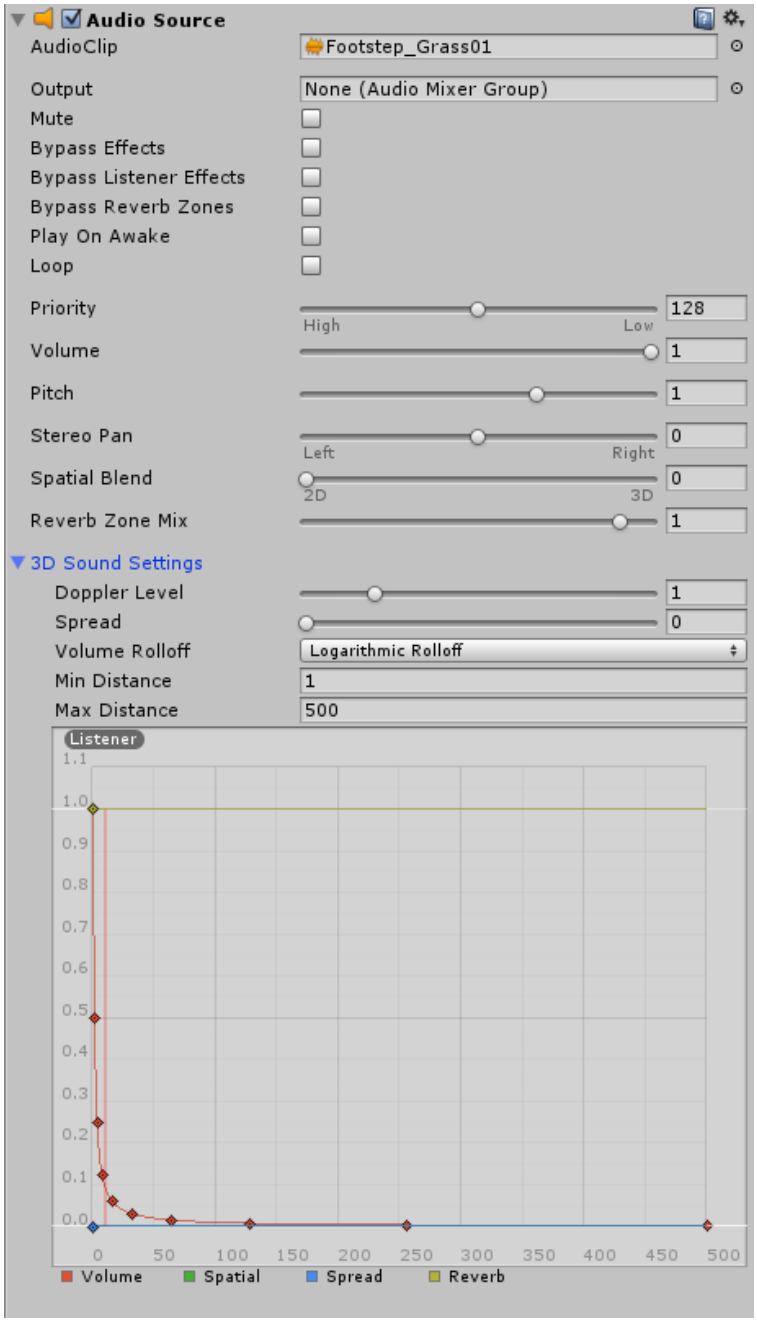
2.1.2 Audio Listener

Audio Listener toimii ikään kuin mikrofonina pelin äänille (Unity Technologies 2015b). Se kuulee kaikki scenen äänet ja toistaa ne äänilaitteen kautta. Scene on alue, johon pelin kenttiä (usein yksi scene kenttää kohden) luodaan asettamalla sinne erilaisia peliobjekteja. Sillä ei ole asetuksia, joten se on hyvin yksinkertainen komponentti. Se on automaattisesti kiinni kamera-komponentissa, joka luodaan uutta sceneä tehtäessä, joten sen tuominen erikseen on yleensä turhaa, sillä jokaisessa scenessä voi olla vain yksi Audio Listener -komponentti.

Komponentti käyttäytyy eri tavalla 2D- ja 3D-äänien kanssa. 2D-äänet kuuluvat aina samalla tavalla riippumatta niiden sijainnista scenessä, mutta 3D-äänten etäisyys ja sijainti kameraan nähden muuttaa äänen kokemista. (Unity Technologies 2015b.)

2.1.3 Audio Source

Jos Audio Listener on kuin mikrofoni, tuottaa Audio Source -komponentti (Kuva 2) äänen jonka se kuulee, ikään kuin kaiutin (Unity Technologies 2015c). Audio Source kiinnitetään objektiin, jolle halutaan antaa jokin ääni. Tämä ääni voi olla mikä tahansa Audio Clip -ääni, kuten musiikki tai foley-ääni. Audio Sourcessa ääni voidaan toistaa joko Audio Listenerin tai Audio Mixerin läpi. Äänen toistotapaa on mahdollista muokata erilaisilla asetuksilla (taulukko 2).



Kuva 2. Audio Source -komponentti (Kuva: Roope Timonen).

Taulukko 2. Audio Source -komponentin asetukset (Unity Technologies 2015c).

Mute	Onko valittu Audio Source hiljennetty?
Bypass Effects/Listener Effects/Reverb Zones	Ohitetaanko Audio Sourcen, Audio Listenerin tai Reverb Zonen efektit?

Play On Awake	Toistetaanko ääni kun ohjelma suoritetaan?
Loop	Toistetaanko ääntä silmukassa (loopissa)?
Priority	Kuinka tärkeä tämä Audio Source on (kuinka se kuuluu muiden scenessä olevien äänien seasta)?
Volume	Äänenvoimakkuus.
Pitch	Äänen nopeus ja korkeus.
Stereo Pan	2D äänten panorointi (kuuluuko ääni tasaisesti, vai enemmän yhdestä suunnasta?)
Spatial Blend	3D-moottorin vaikuttamiseen äänessä (Onko ääni 2D-ääni, 3D-ääni vai jotain siltä väliltä?)
Reverb Zone Mix	Kuinka paljon äänisignaalia lähetetään Reverb Zoneen?

Jos ääni on 3D-moottorin vaikutuksen alaisena, saadaan komponenttiin lisäasetuksia, joista mainittakoon Spread-asetus, jolla voidaan säätää äänen ”kulmaa”, eli kuinka ääni kuullaan suhteessa Audio Listeneriin (kuuluuko ääni vain siitä kaiuttimesta jonka puolella Audio Listener on, tasaisesti molemmista kaiuttimista, vai mahdollisesti jotain tältä väliltä).

2.1.4 Audio Filters

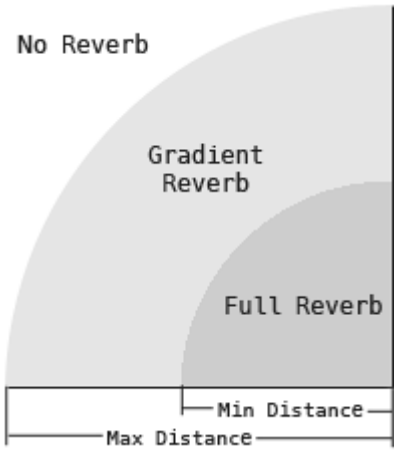
Audio Filters -komponenteilla on mahdollista muokata Audio Sourcen antamaa ja Audio Listenerin kuulemaa ääntä lisäämällä efektejä objektiin, jossa ne ovat kiinni (Unity Technologies 2015d). Jokainen efekti on oma komponenttinsa, joten niitä voi lisätä oman tarpeensa mukaan (taulukko 3). Audio Filtereitä käytettäessä kannattaa ottaa huomioon järjestys, jossa efektit ovat objektin hierarkiassa, sillä tässä järjestyksessä ne tulevat voimaan.

Taulukko 3. Audio Filter -komponentit (Unity Technologies 2015d).

Low Pass Filter	Miten korkeat taajuudet poistetaan annetuista tai kuulluista äänistä. Tällä voidaan luoda esimerkiksi luoda kuva äänestä joka kuuluu oven takaa.
High Pass Filter	Toimii samalla periaatteella kuin Low Pass Filter, mutta poistaa matalat taajuudet
Echo Filter	Toistaa äänen määrätyn ajan jälkeen uudestaan hiljentäen sitä annetun määrän verran. Näin voidaan luoda erilaisia kaikuefektejä.
Distortion Filter	Vääristää äänen toistoa saaden sen kuulostamaan huonolaatuiselta. Tätä voidaan käyttää esimerkiksi radioäänen luontiin.
Reverb Filter	Tällä voidaan luoda pitkälle itsesäädettyjä kaikuefektejä.
Chorus Filter	Tällä voidaan luoda kuoromaisia efektejä. Alkuperäistä ääntä muokataan, jolloin se kuulostaa kun se tulisi useammasta lähteestä jotka kaikki eroavat hieman toisistaan

2.1.5 Reverb Zones

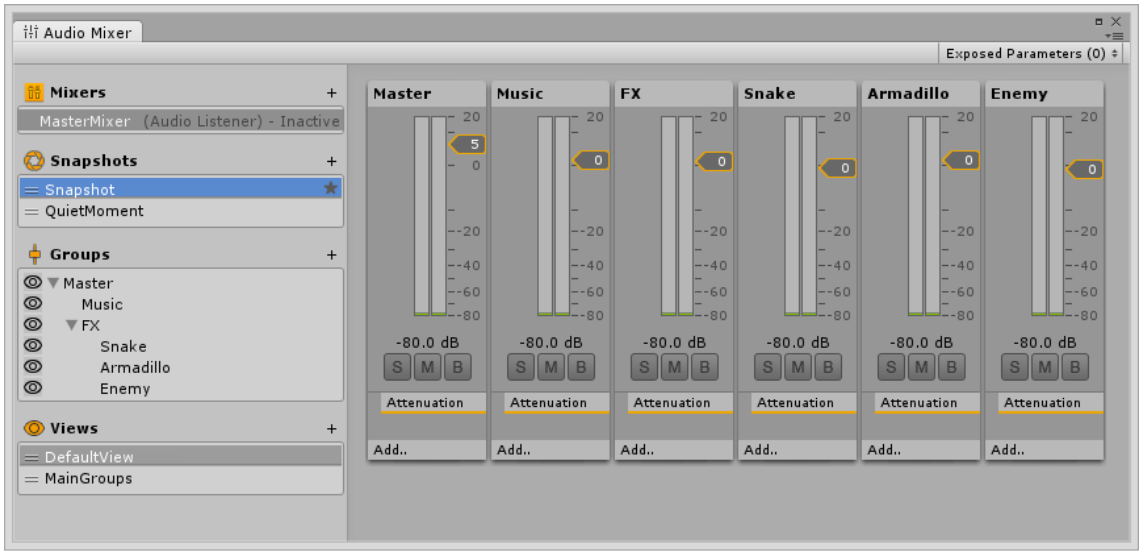
Reverb Zone -komponentilla voidaan luoda samanlaisia kaikuefektejä kuin Reverb Filterillä ja Audio Mixerin SFX Reverb Effectillä (Unity Technologies 2015e). Se kuitenkin eroaa näistä kahdesta siten, että se luo alueen jolla kaiku lisätään ääneen. Sille määritetään vähimmäisetäisyys, jonka sisällä ääni saa täydellisen kaikuefektin, ja maksimietäisyys, jonka sisällä ääni saa osittaisen kaikuefektin, joka vahvistuu vähimmäisetäisyyttä lähestyttäessä (Kuva 3). Alueen ulkopuolella kaiku ei vaikuta ollenkaan.



Kuva 3. Reverb Zonen toiminta (Kuva: Unity documentation 2015).

2.2 Audio Mixer

Audio Mixer (Kuva 4) on Unity 5:ssä implementoitu uusi tapa muokata ääniä (Unity Technologies 2015f). Se on nimensä mukaisesti virtuaalinen miksauspöytä, jolla voidaan hallita äänen tasoja ja lisätä sekä muokata efektejä. Audio Mixerissä voidaan luoda useita miksereitä ja näille erilaisia mikseriryhmiä. Nämä mikseriryhmät rakentuvat yhden pääryhmän alle, jolla voidaan hallita kaikkien alaryhmien tasoja yhtä aikaa. Myös näillä alaryhmillä voi olla omia alaryhmiä, joten mikserin hierarkia on puumainen. Audio Mixerissä on mahdollista luoda snapshotteja, joilla voidaan tallentaa senhetkiset äänentasot myöhempää käyttöä varten. Audio Mixerin ulkonäköä voi myös muokata luomalla erilaisia näkymiä piilottamalla joitakin mikseriryhmiä ja tallentamalla ne View-kohtaan.



Kuva 4. Audio Mixer-komponentti (Kuva: Roope Timonen).

Audio Mixerillä on myös omia Audio Effects -komponentteja. Nämä ovat samanlaisia kuin Audio Filter -komponentit, jotkin jopa täysin samoja, mutta toimivat vain Audio Mixerin kanssa eikä niitä tarvitse asettaa erikseen objekteille (Unity Technologies 2015g). Näitä efektejä voidaan asettaa kaikkiin mikseriryhmiin, jolloin myös niiden alaryhmät saavat nämä efektit. Audio Effects -komponenteista löytyy samat efektit kuin Audio Filter -komponenteista, mutta sillä on myös täysin omia efektejä (taulukko 4).

Taulukko 4. Audio Effects -komponentit (Unity Technologies 2015g.)

Flange Effect	Kaksi samanlaista signaalia yhdistetään, joista toista sitten hidastetaan hieman. Tällä saadaan aikaan vaikutelma äänen kiertymisestä.
Normalize Effect	Ääni normalisoidaan, eli tuodaan sen korkein äänentaso tavoitetasoon, joka on yleensä 0dB (digitaalisen äänen perustaso) Tällä hiljaisemmat äänet saadaan kuulumaan paremmin.
Parametric Equalizer Effect	Toimii kuin parametrinen taajuuskorjain. Tällä voidaan muokata äänisignaaleja. Työkalulla on mahdollista luoda esimerkiksi aiemmin mainittujen High ja Low Pass -

	efektien kaltaisia taajuuksien poistamisia.
Pitch Shifter Effect	Tällä voidaan muokata äänen sävelkorkeutta.
Compressor Effect	Hiljentää kovia ääniä ja voimistaa hiljaisia ääniä, kompressoimalla äänen signaalia. Näin mikseriryhmään saadaan aikaan yhteneväinen äänentaso.
Delay Effect	Tällä voidaan lisätä viivästystä ääneen.

Audio Mixerissä on myös mahdollista asettaa mikseriryhmälle send-toiminto, jonka avulla kyseisen ryhmän signaali voidaan lähettää recieve-toiminnolla varustettuihin mikseriryhmiin. Näissä mikseriryhmissä on tyypillisesti erilaisia efektejä ja niihin voidaan yhdistää useampia send-mikseriryhmiä, jolloin niiden kaikkien äänisignaalit miksataan recieve-mikseriryhmän efektien kanssa. Näin voidaan luoda yhtenäisiä efektejä erilaisiin tasoihin, kuten esimerkiksi kaikuvaan luolaan. Send-toiminnossa on mahdollista määrittää, kuinka voimakkaana äänisignaali lähetetään, joten sillä on mahdollista lähettää vain puolet mikseriryhmän normaalista äänentasosta. (Unity Technologies 2015h.)

2.3 Audio Mixerin hyödyt

Audio Mixer helpottaa huomattavasti äänien käsittelyä Unityssä. Ennen Unity 5:ttä äänet olivat yksittäisiä osasia, joita ei saanut pelimoottorin omilla työkaluilla yhdistettyä mitenkään. Ilman lisäosia ohjelmoijien täytyi luoda erilaisia listoja tai taulukoita, joihin ääniobjekteja varastoitiin.

Hyvä esimerkki Unityn käyttämisestä ennen Audio Mixeriä löytyy pelinkehitysblogia pitävän Mark Bureauxin (Gamasutra 2014) lisäämästä tekstistä. Siinä hän antaa neuvoja, kuinka äänitarpeiltaan pieneen peliin voidaan lisätä ääniä, joiden tasoja on helppo kontrolloida. Ilman minkäänlaisia lisäosia hän joutuu lisäämään kaikki ääniefektejä tarvitsevat peliobjektit yhden ”SFX”-nimisen peliobjektin alle. Tämän lisäksi hän joutuu kutsumaan näitä objekteja koodin kautta,

jotta hän pääsee käsiksi näiden Audio Source -komponentteihin ja pystyy tallentamaan ne käyttäen C#:n Dictionary-luokkaa. Näin hän pystyy helpommin kutsumaan ja hallitsemaan jokaisen efektin äänentasoa. Taustamusiikki on asettava jättämään Audio Listenerin äänen taso huomioimatta, jotta kaikkien SFX-äänien tasoja voidaan hallita yhtä aikaa muuttamatta musiikin tasoa. Tämän takia taustamusiikki vaati erikoisasetuksia toimiakseen kunnolla. Ratkaisu on toimiva, mutta kuten Bureaux itsekin toteaa tekstinsä alussa, tällainen toimii pelissä, jossa äänitarpeet ovat minimaaliset.

Audio Mixerin myötä tällaisia tekniikoita ei enää tarvita, mikä säästää ohjelmoijan työtä huomattavasti. Äänen liittämiseen tarvitaan vain muutama klikkaus peliobjektin Audio Sourcesta. Jokaiselle yksittäiselle äänelle on mahdollista luoda oma mikseriryhmänsä, jonka voi asettaa toisen mikseriryhmän alle ja niin edelleen. Äänen tasojen säätämiseen pelin aikana tarvitaan vielä tietenkin ohjelmointia, mutta jokaista yksittäistä ääntä ei enää tarvitse erikseen muuttaa koodin kautta.

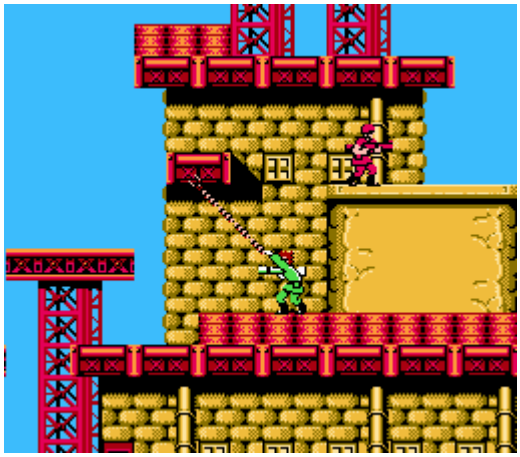
Yksi Audio Mixerin suurimpia etuja on mahdollisuus muokata tasoja ja efektejä samalla kun peliä simuloidaan. Yleensä muutokset, joita simuloinnin aikana tehdään, palautuvat tilaan, joissa ne olivat ennen simuloinnin aloittamista, mutta Audio Mixerin muutokset jäävät voimaan. Tämä mahdollistaa äänentasojen ja efektien nopean muokkauksen ilman jatkuvaa simulointi- ja muokkaustilojen välillä vaihtelua.

Audio Mixerin graafinen käyttöliittymä on yksinkertainen ja helppokäyttöinen. Äänien tasoja on helppo seurata, ja eri mikseriryhmiä on helppo hiljentää, jos ne sillä hetkellä häiritsevät muiden äänten asettelua. Säätimet ovat selvät, ja uusia mikseriryhmiä on helppo luoda ja hallita. Myös efektien asettaminen vaatii vain muutaman napin painalluksen. Tämä mahdollistaa sen, että myös ohjelmoinnista vähemmän tietävät voivat helpommin hallita äänien tasoja pelejä kehitettäessä. Onkin sanottu että mikserien yleistyttyä mikserit ovat tuoneet peliäänet ja -musiikit lähemmäksi elokuvaääniä (Bridgett 2009).

3 Lost Home -tasohyppely

Opinnäytetyön toiminnallinen osuus tehtiin peliin nimeltä Lost Home, joka saattaa myös esiintyä opinnäytetyössä työnimellä Against the Nature. Se on kaksiulotteinen tasohyppelypeli, jonka suurimpina vaikutteina ovat olleet Donkey Kong Country (Kuva 5) ja Bionic Commando (Kuva 6) -videopelit sekä Kaukametsän Pakolaiset (The Animals of Farthing Wood) (Kuva 7) -televisiosarja.

Kuva 5. Donkey Kong Countryn kahden hahmon parivaljakko on toiminut vaikuttajana Lost Homen sankarikaksikolle (kuva: Vizzed.com 2015).



Kuva 6. Käärmeen käyttö hypyn korvaavana heilautusköytenä on saanut inspiraationsa Bionic Commandon tarttuma-aseesta (kuva: Videogame Music Preservation Foundation Wiki 2012).

Kuva 7. Kaukametsän pakolaistet -televisiosarjan tarina pakenemisesta ja uuden kodin etsimisestä on innoittanut Lost Homen tyyliä ja teemoja. (kuva: European Broadcasting Union 2007).

3.1 Päähahmot

Pelissä kaksi hahmoa, käärme ja vyötiäinen, joutuvat olosuhteiden pakosta toimimaan yhteistyössä puolustautuessaan erilaisilta petoeläimiltä ja paetakseen erilaisia luonnonkatastrofeja etsien samalla uutta kotia. Vaikka heillä on matkan varrella paljon kiistoja ja erimielisyyksiä, he lopulta löytävät ystävyyden merkityksen.

Vyötiäinen on parivaljakon aivot. Se voi käärittyä kerälle, jolloin viholliset eivät tee siihen vahinkoa. Kerällä ollessaan vyötiäinen voi myös pyöriä eteenpäin, ja tarpeeksi nopean vauhdin saavuttaessaan, hypähtää hieman ilmaan. Käärme on taas parivaljakon työkalu. Erilaisiin paikkoihin päästäkseen käärme asettua vieteriksi ja lingota kerällä olevan vyötiäisen ilman halki korkeammalle tasolle. Vyötiäinen voi myös tarttua käärmeeseen ja heittää tämän. Käärme voi näin tarttua oksaan, jolloin vyötiäinen voi heilauttaa itsensä vaikkapa rotkon yli (kuva 8), tai hyökätä vihollisen kimppuun puremalla sitä. Pääasiassa hahmot liikkuvat

yhdessä, mutta toisinaan eroavat toisistaan erilaisiin soolo-osuuksiin, joissa ne joutuvat selviämään ainoastaan omien kykyjensä avulla.

Kuva 8. Vyötiäinen ja käärme toimivat yhteistyössä paetakseen metsäpaloa (kuva: Roope Timonen).

3.2 Pelin kehittäminen

Pelin kehittäminen aloitettiin jo vuonna 2014 keväällä, jolloin kolme ohjelmoijaa tekivät pienen prototyypin pelistä tietojärjestelmäprojekti-kurssille käyttäen Javaa ja sille tehtyä LibGDX-rajapintaa (Zechner 2013). Se ei kuitenkaan osoittautunut hyväksi työkaluksi kokemattomille ohjelmoijille, sillä monta asiaa, kuten törmäystarkistukset, olisi tullut tehdä alusta loppuun itse. Kun prototyyppi oli valmis, olimme vakuuttuneita, että työkalua tulisi vaihtaa.

Meillä oli hieman kokemusta Unity3D:n käytöstä eräältä ohjelmointikurssilta, joten päätimme vaihtaa kyseiseen pelimoottoriin. Tämä osoittautui hyväksi valinnaksi, sillä Unitysta löytyi paljon valmiita ominaisuuksia ja työkaluja, kuten pal-

jon kaivatut törmäystarkistukset, fysiikkamoottori ja animaattori, joilla pelin kehittäminen muuttui nopeammaksi ja yksinkertaisemmaksi. Selkeän käyttöliittymän ansiosta myös tiimimme ei-ohjelmoivat jäsenet pystyivät oppimaan nopeasti scenejen rakentamista ja muita yleensä ohjelmoijille miellettyjä tehtäviä. Unityn etuja on myös sen suuri käyttäjäkunta, jolta voi kysyä neuvoa Unity Answers -foorumilta, kattava dokumentointi sekä Unityn, että käyttäjien puolesta, ja Asset Store -kauppa, josta löytyy yli 20 000 erilaista lisäosaa (Unity Technologies 2015j).

Myöhemmin ostimme Unity Storesta Ferr2D-lisäosan (Simbryo Corporation 2014), jolla maaston tekeminen onnistuu nopeasti muokkaamalla vain yhtä maaobjektia haluttuun muotoon. Tämä nopeuttaa ja helpottaa kenttien rakentamista, sillä ennen Ferr2D käyttöönottoa käytimme yksittäisiä maatilejä maastojen muodostamiseen. Tämä oli hidasta ja välillä jopa rasittavaakin, sillä pienetkin muutokset saattoivat aiheuttaa sen, että kenttäsuunnittelijamme joutui siirtämään melkein jokaista palasta hieman eri asentoon. Ferr2D:n avulla ei tarvitse kuin ottaa kiinni maapalan nurkasta ja venytellä sitä haluamanlaisekseen.

Aloin ohjelmoimaan projektia aktiivisesti Unityn versiolla 4.2. marraskuussa 2014 osana työharjoitteluani pelihautomolla. Tällöin peli vaikutti hyvin erilaiselta, sillä meillä oli ajatuksena, että peliin voisi implementoida kaksinpelijärjestelmän, jossa toinen pelaaja pelaisi käärmettä ja toinen vyötiäistä. Tämä idea hylättiin jo aikaisessa vaiheessa. Ensimmäisenä aloin ohjelmoimaan pelin perusmekaniikkoja, kuinka vyötiäinen ja käärme toimivat yhdessä ja erikseen. Työnteko oli hidasta ja improvisoitua, sillä tein töitä yksin ilman minkäänlaista designdokumenttia tai pelisuunnittelijaa tuottajamme pistäytyessä silloin tällöin katsomaan etenemistäni ja antamaan kommentteja tehdystä työstä. Sain silti vuoden loppuun mennessä valmiiksi jonkinlaisen prototyypin, jolla saimme havainnollistettua ensimmäistä kertaa kuinka peli tulisi toimimaan. Vuoden vaihteen jälkeen työskentelin pelin parissa vielä kuukauden, kunnes työharjoitteluni loppui. Tämän jälkeen pelin kehitys keskeytyi muutamaksi kuukaudeksi, kunnes kolme muuta kehitystiimin jäsentä aloittivat harjoittelunsa pelihautomolla. Sen jälkeen olemme aktiivisesti kehittäneet peliä noin puoli vuotta. Noin kahden vuoden aikana projektiin osallistuvien määrä on vähentynyt, ja tällä hetkellä sen parissa

työskentelee viisi ihmistä: kaksi graafikkoa, ohjelmoija, kenttäsuunnittelija ja tuottaja/suunnittelija. Nykyinen käytettävissä oleva Unityn versio on 5.2.

4 Toiminnallinen osuus

Ennen toiminnallisen osuuden aloitusta pelin äänet olivat paikallaan hyvin yksinkertaisella tavalla. Jokainen ääni toistettiin AudioSource.PlayOneShot()-metodilla, joka asetettiin aina sille sopivaan kohtaan koodissa. Musiikin toistaminen tapahtui liittämällä kameraan Audio Source -komponentti, johon liitettiin toistettava musiikkiraita, joka aloitettiin aina alusta sen päättyessä. Äänet eivät olleet tuolloin prioriteettina peliä kehitettäessä, eikä niistä ollut vielä minkäänlaista suunnitelmaa. Tämä on yksi niistä syistä miksi tätä opinnäytetyötä tehdään; se edistää Lost Home -pelin kehitysprosessia sekä luo tietopohjaa äänien käsittelystä tuleville projekteille.

Kaikki pelin äänet on liitetty Audio Mixeriin, josta niiden tasoja voidaan tarkkailla ja säätää tarvittaessa. Toiminnallisen osuuden työstämisen aikana mikseri on auttanut monessa asiassa, kuten erilaisten efektien hiomisessa. Koska yksittäisen mikseriryhmän voi asettaa olemaan ainut jonka ääntä kuullaan, tai hiljentää kokonaan, on äänien paikalleen saaminen ollut helppoa.

Tässä osiossa käydään läpi äänitöitä, joita peliä varten on tehty. Ensimmäiseksi puhutaan SFX- äänien käyttämisessä eri objektien kanssa, minkä jälkeen perehdytään pelimusiikkiin liittyneisiin toimenpiteisiin. Luvuissa esitellään peliobjekteja, joille äänitöitä on tehty toiminnallisen osuuden puitteissa. Kaikkia haluttuja ääniefektejä ja musiikkia ei tässä ajassa onnistuttu saamaan, mutta demolle kaikkein ominaisimmat äänet ovat läsnä.

4.1 SFX

Kun puhutaan pelien äänistä, on selvää että niiden tärkein osa-alue on SFX, eli ääniefektit. Jos kuvitellaan että pelissä ei olisi musiikkia tai dialogiääniä, kokemus ei jäisi yhtä tyhjäksi kuin ääniefektien puuttuessa. Harva haluaa esimerkiksi ampua aseella, josta ei pääse minkäänlaista ääntä. Nämä efektit auttavat pelaajaa immersoitumaan pelimaailmaan ja antamaan näin paremman ja viimeistellymmän kokemuksen. (Isaza 2010)

4.1.1 Vyötiäinen

Vyötiäinen on toinen pelin päähahmoista, ja hahmo jolla peliä pääasiassa pelataan, käärmeen seuratessa mukana ja auttaessa eri tavoin. Tämän takia kaikki äänet, joita käärme päästää yhteisissä kentissä, ovat itseasiassa määritelty vyötiäisen käyttämässä koodissa. Vyötiäinen käyttää kahta eri mikseriryhmää, joista toinen käsittelee kaikki vyötiäisen ja käärmeen toimintaäänet: hyökkäyksen, puuhun tarttumisen, vyötiäisen ilmaan laukaisemisen ja kuolemisen. Toinen mikseriryhmä on varattu ainoastaan vyötiäisen askeläääniä varten, ja se on ensimmäisen mikseriryhmän alainen.

Hyökkäys- ja laukaisuäänet olivat yksinkertaisimmat asettaa paikalleen, sillä toiminnot joihin ne liittyivät, olivat yksinkertaisia. Äänet toistetaan kun pelaaja painaa määriteltyä painiketta hyökätäkseen tai laukaistakseen vyötiäisen ilmaan. Tarttumisääni toistetaan, kun vyötiäisen heittämä käärme osuu oksaan tai muuhun tartuttavaan objektiin. Heitettävä käärme ampuu raycast-sädettä, joka osuessaan tietynlaiseen objektiin käynnistää ehtolauseen, joka kiinnittää käärmeen tähän edellä mainittuun objektiin.

Kuolemaäänen toteutus tuotti aluksi ongelmia, sillä ruudun pimeäksi leikkaamiseen käytetty IEnumerator-funktio toisti itsensä useamman kerran sitä kutsuttaessa, jolloin kuolemaääni toistettiin useamman kerran joka kerta kun parivaljakon elämä päättyi. Funktio vähensi myös pelaajalta liikaa elämiä, jolloin kerran kuoltuaan niitä saattoi menettää useamman kerralla. Näiden ongelmien takia luotiin oma Death-funktio, joka toisti kuolemaäänen ja suoritti elämien vähentä-

misen, jonka jälkeen se aloitti ruudun pimeäksi leikkaamisen. Näin pelaajan kuollessa ääni saatiin toistettua vain yhden kerran.

Askeläänіä oli seitsemän kappaletta, jottei kävely kuulostaisi itseään toistavalta. Niiden toistamiseen luotiin oma WalkSoundScript-luokka (kuva 9), jossa äänet tallennettiin taulukkoon. Sieltä niitä toistettiin satunnaisessa järjestyksessä käyttämällä Random.Range-metodia, joka valitsi satunnaisesti luvun nollan ja kuuden välillä, ja toisti taulukosta vastaavan äänitiedoston.

```
using UnityEngine;
using System.Collections;

public class WalkSoundScript : MonoBehaviour
{
    public AudioClip[] audioClips = new AudioClip[7];
    public AudioSource aS;
    private PlayerC player;
    int i;

    void Awake()
    {
        player = GetComponentInParent<PlayerC>();
        aS = GetComponent<AudioSource>();
    }
    public void PlayClip()
    {
        if (player.grounded)
        {
            i = Random.Range(0, 6);
            aS.clip = audioClips[i];
            aS.PlayOneShot(audioClips[i], 0.5f);
        }
    }
}
```

Kuva 9. WalkSoundScript (kuva: Roope Timonen).

Askeläänien kanssa oli ongelmia useammasta syystä. Kun niitä toistettiin samalla Audio Source -komponentilla, jolla muita vyötiäisen ääniä, aiheutti kävelyäänien nopea toistotahti muiden äänien keskeytymisen liikkuessa. Esimerkiksi liikkuessa hyökkäysääntä ei kuulunut ollenkaan. Tämän takia vyötiäiselle asetettiin toinen Audio Source-komponentti, mutta se asetettiin peliobjektin alaobjektille, joka tarkisti oliko vyötiäinen maassa vai ei. Myös WalkSoundScript siirrettiin alaobjektiin. Näin vyötiäisen askeläänet voitiin toistaa ilman että ne keskeyttivät muita toimintaääniä. Askeläänet saivat oman mikseriryhmän, että

askeleiden äänenvoimakkuutta voidaan säädellä riippumatta muista hahmon äänistä.

Askelien äänet täytyi myös ajoittaa yhteen juoksuanimaation kanssa. Koska animaatio oli aina saman pituinen, ajateltiin ratkaisuksi yksinkertaista ajastinta, joka asetettaisiin aina laskemaan tietty aika kun juoksuanimaatio lähtee liikkeelle ja ääni toistettaisiin aina määritellyn ajan kuluessa animaation aloituksesta. Ennen kuin tätä päästiin kokeilemaan käytännössä, löytyi Unityn Animation-työkalusta Add event -toiminto, jolla yksittäiselle animaation ruudulle voitiin asettaa metodi, joka suoritettiin aina kun kyseinen ruutu toistettiin. Tässä ongelmaksi tuli se, että toiminnolla voitiin käyttää vain niiden luokkien metodeja, jotka ovat kiinni samassa peliobjektissa kuin Animator-komponentti, joka käsittelee peliobjektien animaatioita. Ongelma selvitettiin luomalla PlayerC-luokkaan WalkSound-metodi, jonka kautta ääni voidaan toistaa kutsumalla WalkSoundScript-luokasta PlayClip-metodia.

4.1.2 Opossumi

Opossumi on pelin ensimmäisiä vihollisia, joten se on toiminnaltaan hyvin yksinkertainen: kun se näkee pelaajan, se päästää huudon ja lähtee juoksemaan tätä kohti, ja kohdalle päästessään aiheuttaa pelaajaan vahinkoa. Hahmolla ei ollut kuin askeleäänet ja karjaisu. Askeleet saatiin aikaan käyttämällä samaa koodin pohjaa jolla vyötiäisen kävely toteutettiin, mutta siihen täytyi tehdä joitain muutoksia. Opossumille asetettiin mikseriryhmään Enemies.

Kävelyääniä käsiteltäessä ongelmaksi nousi ensimmäisenä se, kuinka 2D-äänet käyttäytyvät. Peliä suorittaessa opossumin kävelyäänet kuuluivat kaikkialle, vaikka itse peliobjekti sijaitsi toisella puolella kenttää. Ensimmäinen ratkaisu olisi ollut laskea pelaajan ja opossumin välinen etäisyys käyttäen Vector3.distance-funktiota, joka laskee kahden vektorin etäisyyden, ja asettamalla äänenvoimakkuus tämän luvun mukaan. Tarkastelemalla Audio Source -komponentin 3D-äänen asetuksia kuitenkin huomattiin että on mahdollista asettaa ääni 3D-tilaan jolloin äänen voimakkuus vähenee kun Audio Listener -

komponentti siirtyy kauemmaksi Audio Sourcesta. 3D-ääniin vaikuttaa etäisyyden lisäksi myös sijainti, joten kävelyäänet kuuluivat ainoastaan opossuminpuoleisesta kaiuttimesta, mikä ei sopinut 2D-peliin. Asettamalla komponentin Spread-asetus 180 asteeseen saatiin ääni kuulostamaan edelleen 2D-ääneltä, mutta sen voimakkuus muuttui suhteessa opossumin etäsyyteen. Max Distance -arvo asetettiin noin puoleen kameran leveydestä, jolloin opossumin äänet kuuluvat juuri ennen kuin se astuu kameran alueelle.

Opossumi oli sama ongelma kuin vyötiäisellä: sen kävelyäänet peittivät alleen karjaisun, joka toimii hyvänä merkkinä siitä milloin opossumi huomaa pelaajan. Tämä ongelma ratkaistiin pelaajahahmolla käyttäen kahta Audio Source -komponenttia, mutta opossumilla ei ollut alaobjektia, jolle Audio Source -komponentin voisi asettaa. Tätä ongelmaa varten kirjoitettiin CharacterSoundScript-luokka (kuva 10), jossa on käytetty pohjana vyötiäsen WalkSoundScript-luokkaa. Se toistaa kävelyääniä vain silloin kun ehto ”nothingPlays” on tosi. Kun jokin toinen ääni toistetaan, nothingPlays muuttuu epätodeksi, jolloin kävelyäänet lakkaavat, ja tarvittava ääni voidaan toistaa ilman että se joutuu toisten äänten katkaisemaksi.

```
using UnityEngine;
using System.Collections;

public class CharacterSoundScript : MonoBehaviour
{

    /*This array is used to store audio clips for walking sounds,
    it has initial size of one, but can be expanded for as many clips as one might need.*/
    public AudioClip[] walkingClips = new AudioClip[1];
    public AudioSource aS;
    private bool nothingPlays;
    public AudioClip attack;
    int i;

    void Awake()
    {
        nothingPlays = true;
        aS = GetComponent<AudioSource>();
    }

    //This script is used to play walking sounds. It is set to play whether there are no other sounds to be played.
    public void WalkingClip()
    {
        if (nothingPlays)
        {
            //Random number between 0 and the last seed of the array is taken, and corresponding clip from array is played.
            i = Random.Range(0, walkingClips.Length - 1);
            aS.clip = walkingClips[i];
            aS.PlayOneShot(walkingClips[i], 0.25f);
        }
        if (!aS.isPlaying && !nothingPlays)
            nothingPlays = true;
    }

    //This method is used for attack sound. Bool nothingPlayis is set to false, so walking sounds do not interrupt clip.
    public void PlayAttack()
    {
        nothingPlays = false;
        aS.PlayOneShot(attack, 1f);
    }
}
```

Kuva 10. CharacterSoundScript-luokka (kuva: Roope Timonen).

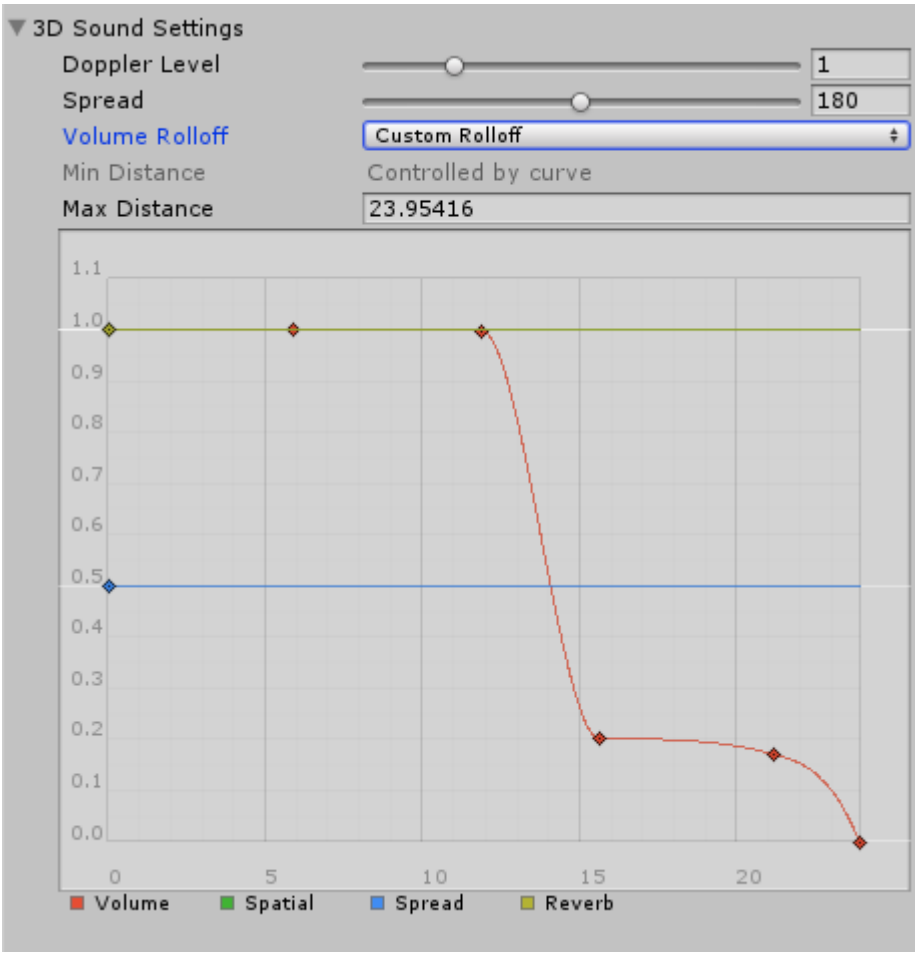
Opossumin hyökkäysääni oli ensimmäisiä, joita peliin liitettiin. Se toimi aluksi hyvin yksinkertaisesti opossumia pääasiallisesti kontrolloivassa luokassa käyttämällä Audio Sourcen PlayOneShot-metodia kun opossumin raycast-säde osuu pelaajaan, ja opossumi vaihtaa normaalista käyskentelytilastaan hyökkäystilaan. Kirjoitettaessa uudelleen vyötiäisen WalkSoundScript-luokkaa opossumille sopivammaksi, tuli ajatus siirtää myös opossumin hyökkäysääni tähän luokkaan, jotta näitä ääni voitaisiin hallita yhdestä paikasta.

4.1.3 Kovakuoriainen

Kovakuoriainen on vihollinen, joka ei hyökkää, mutta sen kosketus on silti turmiollinen pelaajalle. Näitä lentäviä hyönteisiä tavataan usein osuuksissa, joissa ollaan tavalla tai toisella irti maasta, kuten sienillä hyppiessä. Kovakuoriaisilla

on tietty lentorata, ja niitä pitää osata väistää oikealla hetkellä, mikä voi olla vaikeaa ilmassa.

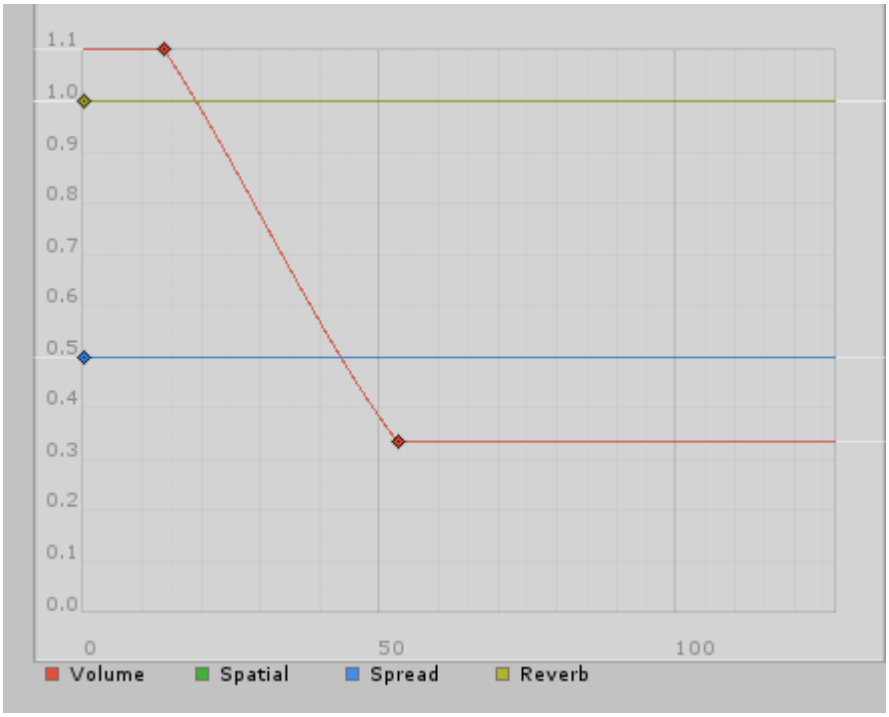
Kovakuoriaisilla on vain yksi ääni, ininä, jota toistetaan aina kun se nähdään ruudussa. Ääni on asetettu 3D-ääneksi ja spread-arvo on 180, aivan kuin opossumilla, jotta ääni kuullaan vain kun kovakuoriainen on läsnä. Opossumista poiketen, kovakuoriaisen volume rolloff -käyrä on asetettu siten, että ääni kuuluu hiljaa kunnes aivan sen läheisyyteen tullaan, jolloin äänen voimakkuus nousee huomattavasti (kuva 11). Tämä luo hyvän efektin läheltä piti tilanteesta, kun esimerkiksi sieneltä hypätessä pelaaja menee juuri ja juuri kovakuoriaisen vierestä.



Kuva 11. Kovakuoriaisen volume rolloff -käyrä (kuva: Roope Timonen).

4.1.4 Tuliseinä

Tuliseinä on ensimmäinen katastrofi, jonka päähahmot kohtaavat, ja on seurausta metsäpalosta, jonka takia käärme ja vyötiäinen alun perin alkoivat toimia yhteistyössä. Se on keskeisessä asemassa pelin ensimmäisessä pakenemiskentässä, jossa se liikkuu hitaasti kentän poikki syöden kaiken tieltään joten sen kosketus on tappava pelaajalle. Jotta tuliseinä tuntuisi tarpeeksi uhkaavalta, sen ääni kuullaan koko pakenemiskentän ajan, mutta pelaajaa lähestyessään tulen äänenvoimakkuus alkaa nousta (kuva 12). Tuliseinän ääni toimii hyvin samalla tavalla kuin kovakuoriaisella, mutta sen volume rolloff -käyrä on erilainen.



Kuva 12. Tuliseinän volume rolloff -käyrä (kuva: Roope Timonen).

4.1.5 Pyörivät kivet

Pyöriviä kiviä tavataan pelissä useamman kerran. Yleensä ne tipahtavat ylhäältä päin ja pyörivät reittiä, jota pelaajankin tulisi kulkea. Niistä varoitetaan pienemillä kivillä, joita tipahtelee jatkuvasti siitä kohtaa, josta isompi lohkare tulee ti-

pahtamaan. Pelaajan täytyy siis ajoittaa liikkumisensa oikein, tai hän voi jäädä kivien alle.

Kivien haluttiin päästävän kahdenlaista ääntä: kun ne tipahtaessaan osuvat maahan, ja kun ne pyörivät. Näiden äänien suorittamiseksi luotiin Rolling-RockScript (kuva 13). Pyörimisäänen toiminto luotiin ensimmäiseksi käyttämällä Unityn törmäystarkistusmetodeja OnCollisionEnter2D ja OnCollisionExit2D. Ensimmäinen suoritetaan kun peliobjektin törmää toiseen törmäystarkistuksella varustettuun (collider) objektiin ja toinen suoritetaan kun törmäys toiseen objektiin lakkaa. Törmäyksen kohdetta kummassakin voidaan rajata antamalla metodin sisällä erilaisia ehtolauseita, jotka tarkistavat onko törmätty objekti sellainen, että sen kanssa halutaan suorittaa joitain toimintoja. Näitä metodeita käyttäen luotiin koodi, joka toisti pyörimisäänen aina kun kivi osuu peliobjektiin joka oli merkitty ”Ground”-tunnisteella. Kun kiven kontakti tällaiseen peliobjektiin lakkasi, äänen toistaminen pysäytettiin.

```
//When rock hits ground, an impact noise is played
void OnCollisionEnter2D(Collision2D col)
{
    if (col.gameObject.tag == "Ground")
    {
        if (!audio.isPlaying)
        {
            audio.PlayOneShot(hitGround, 0.5f);
            impact = true;
        }
    }
}

//When rock is staying on ground-tagged object, grounded is true
void OnCollisionStay2D(Collision2D col)
{
    if (col.gameObject.tag == "Ground" && impact)
    {
        audio.volume = Mathf.Abs(rigidbody.velocity.x / 10);
        audio.volume = Mathf.Clamp(audio.volume, 0, 1);
        if (!audio.isPlaying)
        {
            audio.PlayOneShot(roll);
        }
    }
}

//When rock leaves ground tagged object, grounded becomes false
void OnCollisionExit2D(Collision2D col)
{
    if (col.gameObject.tag == "Ground")
    {
        audio.Stop();
        audio.volume = 1;
        impact = false;
    }
}
```

Kuva 13. RollingRockScript-luokan törmäystarkistusmetodit (kuva: Roope Timonen).

Seuraavaksi kivelle tehtiin pyörimisääni, jonka olisi tarkoitus toistua silloin kun kivi liikkuu maassa ollessaan. Tätä varten otetiin käyttöön vielä yksi törmäystarkistusmetodi: OnCollisionStay2D, joka on hieman erilainen kuin aikaisemmat kaksi käytettyä metodia. Siinnä missä OnCollisionEnter2D ja OnCollisionExit2D kutsutaan vain kerran objektiin törmätessä tai siitä irrottautuessa, OnCollisionStay2D:tä kutsutaan jokaisella ruudunpäivityksellä niin pitkään kun objektissa ollaan kiinni. Tähän metodiin lisättiin pyörimisääni, joka toistetaan aina kun pe-

liobjekti on kontaktissa Ground-tunnuksella varustetun peliobjektin kanssa, joten toimintaperiaate on sama kuin törmäysäänen kanssa. Tästä tuli pieni ongelma, sillä kiven liikkumisääni alkoi välittömästi kun se osui oikeanlaiseen peliobjektiin, eikä törmäysääntä kuultu lainkaan. Ongelman ratkaisemiseksi luotiin boolean-muuttuja impact, joka asetettiin todeksi silloin kun kivi osui maahan, mutta vasta sen jälkeen kun törmäysääni toistettiin. Kun pyörimisääni asetettiin toistumaan vasta kun impact on tosi ja ääni toistumaan vasta kun Audio Source -komponentti ei toista muita ääniä, kuullaan aina ensimmäiseksi törmäysääni, jonka jälkeen pyörimisääni alkaa.

Pyörimisäänen kanssa tuli ongelmaksi myös epärealistinen äänenvoimakkuus kiven liikkuessa. Vaikka ääni kuulosti hyvältä kiven pyöriessä nopeasti, esimerkiksi ylämäessä hidastuessa ääni jatkui samanlaisena. Tähän ratkaisuna yritettiin muokata kiven Audio Source -komponentin pitch-arvoa, joka muokkasi äänen nopeutta. Pitch-arvo liitettiin kiven liikkumisnopeuteen, joten mitä hitaammin kivi liikkuu, sitä hitaammin ääni toistuu. Tämä olisi toiminut hyvin, ellei nopeuden kanssa olisi muuttunut myös äänen korkeus. Kivi, jonka ääni muuttuu matalammaksi hidastuessa, ei kuulosta hyvin realistiselta. Seuraavaksi muokkasimme volume-arvoa, eli äänenvoimakkuutta, mikä toimi niin kuin haluttiin. Kiven hidastuessa äänenvoimakkuus pienenee, mikä antaa vaikutelman kiven hidastumisesta myös äänellisesti.

4.1.6 Puunrunko

Puunrunko on tärkeä osa peliä, sillä niiden avulla pelaaja pääsee ylittämään esimerkiksi rotkoja, joiden yli muuten ei pääsisi. Puu seisoo vaivoin pystyssä, ja sen saa kaadettua ampumalla vyötäisen sitä kohti. Kun vyötäinen osuu puuhun, se kaatuu rotkon päälle, jolloin käärmekin pääsee sen yli. Puunrunko tarvitsi kaksi ääntä, kun se katkeaa ja kun se sen jälkeen osuu maahan. Alun perin nämä kaksi ääntä olivat yksi äänitiedosto, joten puuhun osuttaessa kaatumisääni lähti toistumaan ja loppui omia aikojaan ottamatta huomioon milloin puu osui maahan. Tämän takia ääni leikattiin kahtia, jolloin saatiin sekä puun kat-

keamisääni, että maahan osumisen ääni. Äänen käsittely liitettiin jo aikaisemmin tehtyyn TreeFalling-luokkaan (kuva 14), jossa toteutettiin puun kaatumisen toiminta.

Puunrungon kaatumisääni toimii yksinkertaisesti, kun pelaaja aktivoi puunrungon kaatumisen OnTriggerExit2D metodilla, jonka toimintaperiaate on sama kuin kivessä käytetyillä OnCollision-metodeilla. Näiden kahden erona on se että OnTrigger -metodit käyttävät erilaisia trigger-collidereita, jotka eivät pysähdy törmätessä toisiin peliobjekteihin, vaan ilmoittavat vain törmäyksen tapahtuneen. Kun vyötäinen siis poistuu trigger-colliderin alueelta, toistetaan kaatumisääni.

Puunrungon maahan osumisen ääni on tehty OnTriggerEnter2D -metodilla, joka kutsutaan kun rungon yläpäässä oleva ympyrän muotoinen trigger-collider osuu maahan. Ympyrän muotoinen collider valittiin siksi, että se olisi koodissa yksinkertaisempaa kutsua, sillä puunrunko käytti jo laatikko-collideria. Tämä ääni oli siitä erikoinen, että sen haluttiin lopettavan edellisen äänen toisto (mikä oli usein ollut ongelmana muita ääniä ohjelmoitaessa), jottei puunrungon kaatumisääni estäisi maahan osumisäänen toistamista ajoissa.

```
public class TreeFalling : MonoBehaviour
{

    private AudioSource audio;
    public AudioClip falling;
    public AudioClip hit;
    private bool standing;

    // Use this for initialization
    void Start()
    {
        standing = true;
        audio = GetComponentInChildren<AudioSource>();
    }

    void OnTriggerExit2D(Collider2D col)
    {
        if (col.gameObject.tag == "Armadillo" && standing)
        {
            gameObject.GetComponentInChildren<BoxCollider2D>().enabled = true;
            gameObject.GetComponent<EdgeCollider2D>().isTrigger = false;
            GetComponent<Rigidbody2D>().isKinematic = false;
            audio.PlayOneShot(falling);
            standing = false;
        }

    }
    void OnTriggerEnter2D(Collider2D col)
    {
        if (col.gameObject.tag == "Ground" && !standing)
        {
            audio.PlayOneShot(hit);
            GetComponentInChildren<CircleCollider2D>().enabled = false;
        }
    }
```

Kuva 14. TreeFalling -luokka (kuva: Roope Timonen).

4.2 Musiikki

Vaikka aikaisemmin mainittiinkin että musiikki ei ole pelissä aivan yhtä tärkeä ääniefektit, ei sen merkitystä pidä vähätellä. Siinnä missä ääniefektit auttavat luomaan pelistä realistisemman tuntuisen, tuo musiikki mukanaan erilaiset tunnetilat, ja niiden muutokset. Kun kentän pääsee läpi, soitetaan fanfaari, jolla pelaajaa onnitellaan. Tai kun pelaaja saapuu surulliseen kohtaan tarinassa, kuul-

laan tietenkin surullista musiikkia. Näin pelaaja pystyy yhtenäistymään pelin kanssa myös tunteellisella tasolla. (Sarsfield 2014)

4.2.1 Musiikin toistaminen scenessä

Ennen toiminnallisen osuuden aloittamista pelin musiikin toistamisen hoitivat jokaisessa scenessä olevat peliobjektit, joihin oli liitetty Audio Source -komponentti. Jos eri scenejen musiikkeja halusi vaihtaa, ne täytyi käydä yksi kerrallaan muuttamassa. Jotta musiikin hallitseminen olisi yksinkertaisempaa, pelin GameManager-singletoniin luotiin taulukko, johon musiikkiraidat tallennettiin. Musiikin toistamista varten luotiin LoadMusic-luokka (kuva 15), joka hakee GameManagerin taulukosta sceneä vastaavan musiikkiraidan ja toistaa sen. Jotta virheiltä vältyttäisiin, luokkaan lisättiin try-catch -lause, joka ilmoittaa jos taulukossa ei ole paikkaa kyseiselle scenelle. LoadMusic-luokka liitettiin SoundSettings-objektiin, johon oli liitetty myös Audio Source -komponentti. Jokainen SoundSettings-objekti on liitetty Audio Mixerin Music-mikseriryhmään, joten niiden äänenvoimakkuutta voi hallita scenestä huolimatta.

```
using UnityEngine;
using System.Collections;
using UnityEngine.Audio;

public class LoadMusic : MonoBehaviour {

    public AudioMixer mixer;
    public float reverbSend = -80;

    void Start()
    {
        SetMusic();
        SetEffects();
    }


    //When level is started, music is loaded from array levelMusic located in GameManager-script.
    //If no music is found, message is printed that tell to which index music needs to be set.
    void SetMusic()
    {
        try
        {
            var clip = GameManager.instance.levelMusic[Application.loadedLevel];
            GetComponent<AudioSource>().clip = clip;
            GetComponent<AudioSource>().Play();
        }
        catch (System.IndexOutOfRangeException)
        {
            print("Please set music for this level from GameManager script to index " + Application.loadedLevel + ".");
        }
    }

    void SetEffects()
    {
        mixer.SetFloat("sendReverb", reverbSend);
    }
}
```

Kuva 15. LoadMusic-luokka (kuva: Roope Timonen).

4.2.2 Taukovalikon musiikki

Taukovalikon musiikin haluttiin olevan hieman erilainen, jotta visuaalisen eron (ruutu tummenee ja valikko tulee näkyviin) lisäksi olisi myös mahdollista kuulla pelin olevan tauolla. Itse musiikkiraitaa ei kuitenkaan lähdetty muuttamaan, vaan sitä miltä tason musiikki sillä hetkellä kuulostaa. Tätä varten päätetiin käyttää Audio Mixerin efektejä. Pitkällisen kokeilun jälkeen päädyimme Lowpass-efektiin, sillä sen bassoa korostava ääni toi mielikuvan pelistä irtoamisesta. Tämä efekti asetettiin poistamaan musiikista kaikki yli 300 hertsiä korkeammat taajuudet ja jäljelle jäi vain kaikista matalimmat äänet.

Valikkoa varten luotiin oma snapshot, ”Paused”, jossa Lowpass efekti asetettiin 300:taan (normaalisti se oli 22000), ja SFX-mikseriryhmän äänentaso asetettiin -80 desibeliin, eli kuulumattomiin. Snapshotin vaihtaminen lisättiin jo aikaisemmin Taukovalikkoa varten tehtyyn PauseScriptiin, jossa snapshotin muutos ta-

pahtuu AudioMixerSnapshot.TransitionTo()-metodilla joko silloin kun peli asetetaan tauolle tai kun peliä jatketaan.

4.2.3 Musiikin häivytys

Musiikin ollessa tärkeä osa pelin tunnetilaa rakentaessa, toisinaan sen puuttuminenkin voi välittää tunnetiloja ja tilanteita. Tämä mielessä päätettiin luoda toiminto, jolla musiikki voidaan tarvittaessa häivyttää kuulumattomiin, jotta joitain tilanteita voidaan ilmaista musiikin puuttumisella.

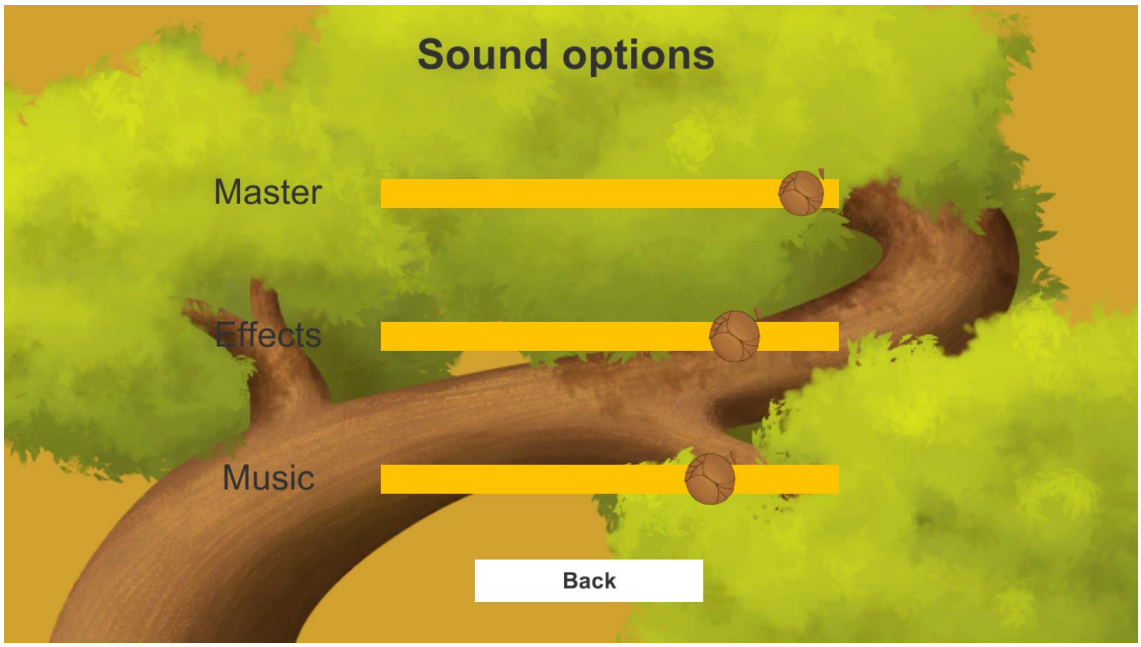
Musiikin häivyttämiseen käytettiin samaa periaatetta kuin ruudin mustaksi pyyhkäisyyn (Thirslund, 2014). Tätä varten luotiin FadeMusic-luokka (kuva 16), joka Update-metodissa lisää äänenvoimakkuuteen joko negatiivista tai positiivista arvoa riippuen dir-muuttujan arvosta. Dir-muuttuja saa joko negatiivisen tai positiivisen arvonsa FadeM-metodista joka saa parametrina direction-muuttujan, jolla määritellään myös dir-muuttuja. Directionin arvo taas asetetaan FadeTriggerMusic-luokassa, ja annetaan FadeM-metodille kun pelaaja astuu trigger-collideriin.

```
public class FadeMusic : MonoBehaviour
{

    private AudioSource audio;
    public bool active;
    public bool fadeOut;
    public float multiplier;
    public float dir;

    // Use this for initialization
    void Start()
    {
        audio = GetComponentInParent<AudioSource>();
    }

    // Update is called once per frame
    void Update()
    {
        audio.volume += Time.deltaTime * dir * multiplier;
        audio.volume = Mathf.Clamp01(audio.volume);
    }

   public void FadeM(float direction)
    {
        dir = direction;
    }
}
```

Kuva 16. FadeMusic-luokka (kuva: Roope Timonen).

4.3 Äänivalikko

Kuten kaikissa peleissä, tarvittiin tähänkin valikko jossa eri äänien tasoja voitaisiin hallita. Itse äänivalikko (kuva 17) tehtiin ensimmäisenä ja siihen tuli kolme liukusäädintä, pää-äänenvoimakkuudelle (Master), ääniefekteille (Effects) ja musiikille (Music). Kaikki nämä liukusäätimet asetettiin toimimaan välillä -80 ja 10, jotka vastaavat hiljaisinta ja voimakkainta äänentasoa. Jotta Audio Mixerin mikseriryhmien taso pystyi hallitsemaan, tuli haluttujen mikseriryhmien äänentasot asettaa julkisiksi, jotta niitä pystyi muokkaamaan koodin kautta (Unity Technologies 2015j).



Kuva 17. Ääniasetusvalikko (kuva: Roope Timonen).

Äänentasojen muokkaukseen luotiin Mixer Levels -luokka (kuva 18), jossa on kolme metodia float-parametrilla, kukin yhdelle liukusäätimelle. Aina kun liukusäädintä liikutetaan, myös mikserin tasot muuttuvat ja äänien tasot tallennetaan GameManager-singletoniin, jotta niihin päästään käsiksi myös pelin aikana ääniasetuksista. Kun valikko oli saatu toimimaan, huomattiin että äänien liukusäätimet toimivat hieman epätarkasti. Noin puolessa välin säädintä ääntä ei kuulunut enää oikeastaan ollenkaan. Ongelman ratkaisemiseksi jokaiseen metodiin tehtiin ehtolause, joka asetti mikseriryhmän äänentason suoraan -80 desibeliin (mykäksi) jos se oli -50 desibeliä. Näin liukusäätimet saatiin asetettua käyttämään pienepää väliä (-50–10) ja toimimaan koko säätimen pituudelta. Siirryttäessä scenestä toiseen liukusäätimet palautuivat alkuperäisille arvoilleen paikoilleen äänien tasojen pysyessä samana. Tätä varten luotiin setLevels-metodi, jota kutsutaan aina kun ääniasetusvalikko avataan. Se käy hakemassa nykyiset mikseriryhmien tasot ja asettaa liukusäätimet kyseisille tasoille.

```
public class MixerLevels : MonoBehaviour {

    public AudioMixer mixer;

   public void SetMaster(float masterLevel)
    {
        //if slider is at lowest point, level is set to -80dB, which is mute
        if (masterLevel <= -50)
        {
            masterLevel = -80;
        }
        mixer.SetFloat("masterVolume", masterLevel);
        //values are stored to GameManager, so they are accessible easily in-game
        GameManager.instance.master = masterLevel;
    }

    public void SetSFX(float sfxLevel)
    {
        //if slider is at lowest point, level is set to -80dB, which is mute
        if (sfxLevel <= -50)
        {
            sfxLevel = -80;
        }
        mixer.SetFloat("sfxVolume", sfxLevel);
        //values are stored to GameManager, so they are accessible easily in-game
        GameManager.instance.sfx = sfxLevel;

    }

    public void SetMusic(float musicLevel)
    {
        //if slider is at lowest point, level is set to -80dB, which is mute
        if (musicLevel <= -50)
        {
            musicLevel = -80;
        }
        mixer.SetFloat("musicVolume", musicLevel);
        //values are stored to GameManager, so they are accessible easily in-game
        GameManager.instance.music = musicLevel;
    }
}
```

Kuva 18. Mixer Levels -luokka (kuva: Roope Timonen).

4.4 Kaiun luominen

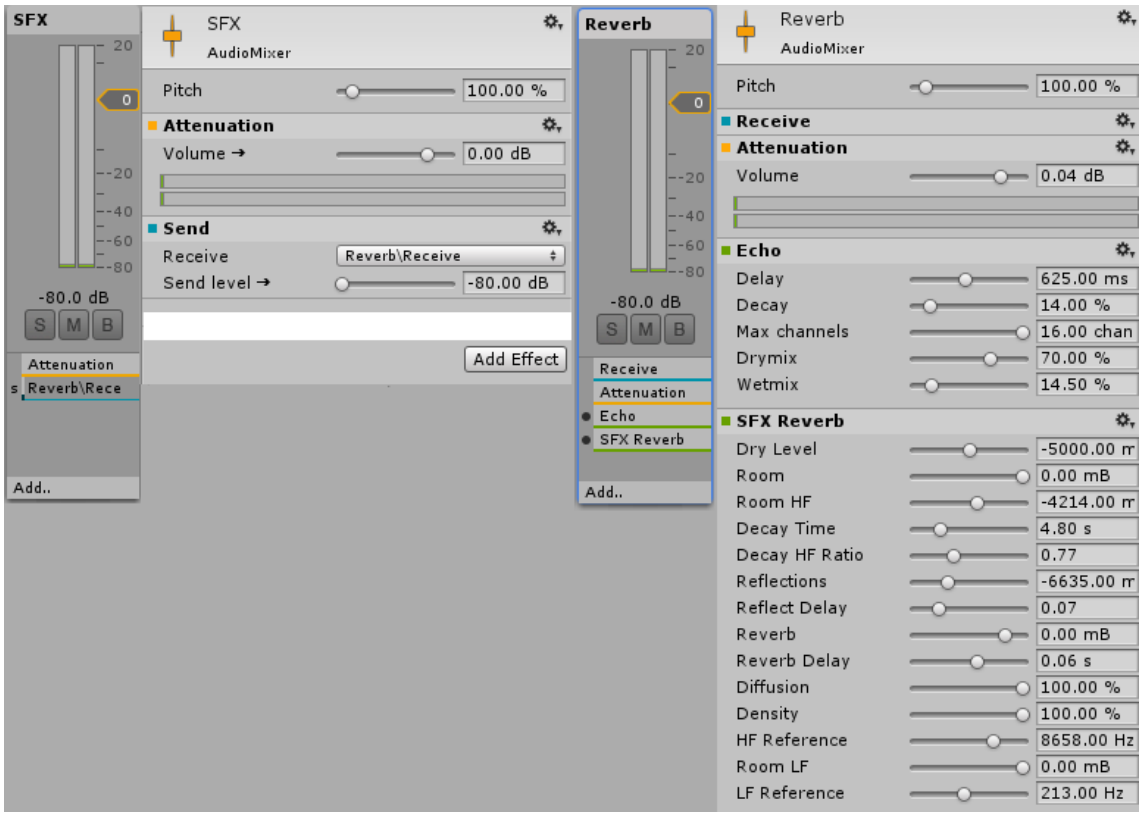
Pelin neljännessä kentässä on hetki, jossa pelaaja kävelee luolan läpi. Luola-hetkeen lisättiin kaiku, jotta pelaaja saisi tunteen alueen vaihtumisesta. Luola ei ollut koko kentän pituinen, joten kaiku päätettiin luoda käyttämällä Reverb Zone

-komponenttia, sillä sitä oli helppo käyttää ja siinnä oli valmiina kaikki mitä tarvittiin. Ensimmäiseksi tarkoituksena oli käyttää vain yhtä peliobjektia, jolle annettaisiin Reverb Zone -komponentti, mutta luolaston rakenteen takia päädyttiin käyttämään useampaa. Komponentit aseteltiin jonoon hieman limittäin, jotta kaiku ei häviäisi kesken luolassa käyskentelyn (kuva 19). Scenen Background-Music-peliobjektin Audio Source -komponentti asetettiin jättämään Reverb Zonet huomiotta, jottei musiikki alkaisi kaikua luolaan astuttaessa.

Kuva 19. Reverb Zone -komponenttien sijoittelu luolastossa (kuva: Roope Timonen).

Kun kaikua halutiin kokonaiseen kenttään ilman että sitä tarvitsi välillä häivyttää, käytettiin Reverb Zonen sijasta Audio Mixerin send-toimintoa. SFX-mikseriryhmän signaali, lähetettiin Reverb-nimiseen mikseriryhmään, johon oli asetettu Echo ja SFX Reverb -efektit (kuva 20). Kuten mikserin äänentasojen kanssa, tuli myös send-arvo asettaa julkiseksi, että sitä olisi mahdollista muokata koodin kautta. Musiikin toistamiseen käytettyyn LoadMusic-luokkaan tehtiin

metodi SetEffect, joka määritti scenen käynnistyessä SFX-mikseriryhmän send-tason.



Kuva 20. SFX- ja Reverb-mikseriryhmät ja niiden asetukset. Nuolet Volume- ja Send level -arvojen vieressä tarkoittavat niiden olevan julkisia (kuva: Roope Timonen)

5 Vaihtoehtoja äänien käsittelyyn

Jos Unity 5 ei olisi tuonut mukanaan Audio Mixeriä, meidän olisi pitänyt ajan myötä alkaa miettiä vaihtoehtoja äänien käyttämiseen. Tämä ei olisi kuitenkaan ollut suuri ongelma, sillä Unityyn on tarjolla muutamia vaihtoehtoja, joilla äänien käsittely helpottuu melko samalla tavalla kuin Audio Mixeriä käytettäessä. Unity Storesta löytyy monenlaisia äänityökaluja, mutta kaikkein suosituin niistä on MasterAudio (Dark Tonic Inc. 2015).

Työkalu tarjoaa samoja ominaisuuksia, joita Audio Mixerikin, kuten mikseriryhmiä, erilaisia ääniefektejä ja pelin aikana tapahtuvaa äänien muokkausta. Asset Store -sivullaan Master Audio on saanut 454 arvostelua ja 5 tähteä, joten siihen ollaan selvästi tyytyväisiä käyttäjien keskuudessa. Master Audio olisikin varmaan ollut valintamme jos olisimme tarvinneet äänityökalua Unityn omien lisäksi, ja se saattaa tulevaisuudessa ollakin hankinnassa, sillä sivulla ilmoitetaan sen toimivan sujuvasti Audio Mixerin kanssa. Master Audio maksaa 30$, joten sen hankkiminen verrattuna ilmaiseen mikseriin ei tuntunut hyvältä idealta.

Toinen mainitsemisen arvoinen liitännäinen on Tazman-Audion tekemä Fabric, jota ei löydy ollenkaan Asset Storesta (minkä takia se on jäänyt huomiotta päätöksiä tehdessä). Kuten MasterAudio, se tuo mukanaan asiat, joita AudioMixer tarjoaa, mutta myös paljon valmiiksi tehtyjä ominaisuuksia, joita käyttäjä voi tarvita, kuten helpon musiikin vaihtamisen ja nopeasti luotavat askeläänet. Fabric on ilmanen alle 150 000$ tuottaville kehittäjille, joten se on oiva valinta niille, joilla ei ole varaa esimerkiksi Master Audioon. (Tazman-Audio 2015)

Miksi sitten Unityn oma äänijärjestelmä valittiin? Se ei sisällä yhtä valmiita ominaisuuksia kuin Master Audio tai Fabric. Se oli kuitenkin mielestämme järkevä valinta, koska sen julkaisu sattui hyvään ajankohtaan. Se oli Unityn kehittämä järjestelmä, joten meistä tuntui että se olisi hyvä hallita, jotta voisimme hyödyntää Unityn kaikkia aspekteja. Työkalu toimi tietenkin pelimottorin muiden komponenttien kanssa, joten se ei muokannut jo valmiiksi opittuja asioita. Kuten aikasemmin mainittiin, emme ole myöskään sulkeneet pois esimerkiksi MasterAudion käyttämistä, mutta tällä hetkellä AudioMixer täyttää tarpeemme. Tulevaisuudessa, kun saamme peliimme lisää ääniä ja musiikkia, on todennäköistä, että kokeilemme hieman pidemmälle kehitettyjä vaihtoehtoja.

6 Lopuksi

Kun aloin työstämään opinnäytetyötä, ei minulla ollut oikeastaan kokemusta äänien käsittelystä pelinkehityksessä ja ajattelin tämän olevan asia joka tulisi erikseen oppia. Loppujen lopuksi se oli kuitenkin kuin mitä tahansa ohjelmointia. Äänistä kuitenkin huomaa paljon helpommin virheensä. Voit esimerkiksi kuulla jos samaa koodinpätkää toistetaan aivan liian usein, sillä ääni alkaa toistua itsensä päälle. Ne antavat ikään kuin äänekästä palautetta siitä mikä ei toimi. Raportin kirjoittamisen aikana on tullut myös mieleen paljon asioita joita olisi voinut tehdä toisin, mutta jotka olen päättänyt jättää korjaamatta, ja myöhemmin pelin kehityskaarella katsoa niitä uudestaan. Lost Homen kehitys tulee jatkumaan kevään edetessä ja opinnäytetyössä tekemiäni ratkaisuja tullaan käyttämään varmasti jossain muodossa pelin edetessä.

Opinnäytetyön aikana jouduin muutaman kerran työstämään ääniefektejä, sillä emme olleet saaneet kaikkea tarvitsemaamme foley-artistiltamme. Teimme kivelle pyörimisäänen ja kovakuoriaisen pörräämisäänen. Tämä oli mielenkiintoinen prosessi ja antoi hieman kosketusta foley-artistin työhön. Esimerkiksi kiven pyörimisen teimme nauhoittamalla paperin rutistelua, josta Audacity-äänenmuokkausohjelman avulla hiljennettiin korkeammat taajuudet ja korostettiin matalia (The Audacity Team 2016).

Unityn äänikomponentit ovat tietenkin tulleet tutuiksi niin teoreettisella että käytännön tasolla. Huomasin pian että Audio Mixerin myötä esimerkiksi Audio Filters-komponentit jäävät käyttämättömiksi, sillä mikserillä on omat efektinsä, joita käyttää. Toisaalta, ehkäpä Audio Filter-komponentit ovat sopivia oikein pienille peliprojekteille, joissa käytetään vain muutamaa sceneä, eikä Audio Mixer ole oikeastaan hyödyllinen.

Kun Audio Mixeriä verrataan Unityn vanhaan äänijärjestelmään, huomataan että tällaiselle työkalulle on ollut tarvetta. Se yhdistää monia aikaisemmin irrallisia osia yhden päätyökalun alle, jolla jokaisen palasen hallitseminen on helppoa. Se onnistuu poistamaan paljon monen yksittäisten äänien ja efektien etsimistä ja säätämistä mikseriryhmillään ja luomaan samalla käyttäjäystävällisen kokemuksen. Tämän, ja monien muiden ominaisuuksien ansioista, ei ole suuri ihme, että Unity on saanut suuren suosion pelinkehittäjien keskuudessa (Batchelor 2015).

Lähteet

Batchelor, J. 2015. 174,000 games were made with Unity this summer. http://www.develop-online.net/news/174-000-games-were-made-with-unity-this-summer/0211497. 26.10.2015.

Bridgett, R. 2009. The Future Of Game Audio - Is Interactive Mixing The Key? 14.4.2009 http://www.gamasutra.com/view/feature/132416/the_future_of_game_audio__is_.php. 3.1.2016

Bureaux, M. 2014. Music and Sound Effects in Unity. 21.10.2014. http://gamasutra.com/blogs/MarcBreaux/20141021/228259/Music_and_Sound_Effects_in_Unity.php. 25.10.2015.

Dark Tonic Inc. 2015. Master Audio. http://www.darktonic.com/p/master-audio.html. 9.1.2016.

European Broadcasting Union. 2007. The Animals of Farthing Wood. 23.11.2007. https://www.ebu.ch/en/eurovisiontv/children_youth/animals_of_farthing_wood.php. 26.1.2016.

Geig M. 2013. Why You Should Be Using the Unity Game Engine. 4.4.2013 http://www.informit.com/articles/article.aspx?p=2031153. 4.1.2016.

Huiberts, S. 2010. Captivating Sound. The role of audio for immersion in computer games. http://download.captivatingsound.com/Sander_Huiberts_CaptivatingSound.pdf. 25.10.2015.

Isaza, M. 2010 Aaron Marks Special: Function of Game Sound Effects. http://designingsound.org/2010/10/aaron-marks-special-function-of-game-sound-effects/. 3.1.2016

Porter, M. 2013 Unity: Now You're Thinking With Components. http://gamedevelopment.tutsplus.com/articles/unity-now-youre-thinking-with-components--gamedev-12492. 4.1.2016

Tazman-Audio. 2015. Fabric Audio Toolset 2.0 http://www.tazman-audio.co.uk/#!fabric/c1oba. 9.1.2016.

The Audacity Team. 2016. About Audacity. http://audacityteam.org/about/ 18.1.2016.

Thirslund, A. 2015 Fading Between Scenes. 30.5.2014 http://unity3d.com/learn/tutorials/modules/intermediate/graphics/fading-between-scenes. 5.1.2016.

Unity Technologies. 2015a. Audio Clip. http://docs.unity3d.com/Manual/class-AudioClip.html. 26.10.2015.

Unity Technologies. 2015b. Audio Listener. http://docs.unity3d.com/Manual/class-AudioListener.html. 26.10.2015.

Unity Technologies. 2015c. Audio Source. http://docs.unity3d.com/Manual/class-AudioSource.html. 26.10.2015.

Unity Technologies. 2015d. Audio Filters. http://docs.unity3d.com/Manual/class-AudioEffect.html. 26.10.2015.

Unity Technologies. 2015e. Reverb Zones. http://docs.unity3d.com/Manual/class-AudioReverbZone.html. 26.10.2015.

Unity Technologies. 2015f. Audio Mixer. http://docs.unity3d.com/Manual/class-AudioMixer.html. 26.10.2015.

Unity Technologies. 2015g. Audio Effects. http://docs.unity3d.com/Manual/class-AudioEffectMixer.html. 26.10.2015.

Unity Technologies. 2015h. AudioGroup Inspector. http://docs.unity3d.com/Manual/AudioMixerInspectors.html. 3.11.2015.

Unity Technologies. 2015i. Asset Store. https://www.assetstore.unity3d.com/en/. 25.11.2015.

Unity Technologies. 2015j. Exposed Audio Mixer Parameters. 24.6.2015. https://unity3d.com/learn/tutorials/modules/beginner/5-pre-order-beta/exposed-audiomixer-parameters?playlist=17096. 7.1.2016.

Unity Technologies. 2015k. Master Audio: AAA Sound. https://www.assetstore.unity3d.com/en/#!/content/5607. 9.1.2016

Sarsfield, J. 2014. Music in Video Games: The Importance of a Good Soundtrack. 8.12.2014. https://blogs.chapman.edu/students/2014/12/08/music-in-video-games-the-importance-of-the-a-good-soundtrack/ 4.1.2014.

Simbryo Corporation. 2014. FerrLib. http://www.ferrlib.com/. 4.1.2016.

Videogame Music Preservation Foundation Wiki. 2012. Bionic Commando 13.8.2012. http://www.vgmpf.com/Wiki/index.php?title=File:Bionic_Commando_-_NES_-_Area_1.png 26.1.2016

Vizzed.com. 2015. Donkey Kong Country. http://www.vizzed.com/play/donkey-kong-country-snes-online-super-nintendo-7787-user-screenshots#user-screenshots. 26.1.2016

Zechner, M. 2013. LibGDX. https://libgdx.badlogicgames.com/. 25.11.2015.