Tampereen ammattikorkeakoulu

Äänien implementoinnin suunnittelu ja toteutus kaupunginrakennuspeliin

Etelämäki Juuso

OPINNÄYTETYÖ
Heinäkuu 2020

Tietojenkäsittelyn tutkinto-ohjelma
Pelituotanto

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittely
Pelituotanto

ETELÄMÄKI, JUUSO
Äänien implementoinnin suunnittelu ja toteutus kaupunginrakennuspeliin

Opinnäytetyö 30 sivua
Heinäkuu 2020

Tämä opinnäytetyö käsittelee ääni-implementoinnin suunnittelun ja toteutuksen eri työvaiheita kaupunginrakennus tyyppisen pelin keskeisimpien järjestelmien kannalta. Tavoitteena oli tutkia FMOD-väliohjelmiston ominaisuuksia ja selvittää, miten ohjelmiston avulla toteutetaan kaupunginrakennus genressä käytetyt keskeisimmät äänijärjestelmät. Työssä selvitettiin myös Unity-pelimoottorin ja FMOD-väliohjelmiston yhteiskäytön mahdollisuuksia äänitoteutuksissa.

Tarkoituksena oli toteuttaa järjestelmät Unity pelimoottorissa projektiin, pääpainona prosessin kuvaaminen. Lisäksi työssä tutustuttiin kaupunginrakennuspeleille ominaisiin haasteisiin ja vaatimuksiin äänien kannalta. Työn äänijärjestelmät tuli suunnitella ja toteuttaa skaalautuvin mahdollisuuksin, ja työn ratkaisuissa otettiin huomioon isompien projektien vaatimia haasteita.

Opinnäytetyön teoreettinen viitekehys pohjautui aihetta käsittelevään kirjallisuuteen, verkkolähteisiin sekä työssä käytettyjen työkalujen dokumentaatioihin. Opinnäytetyössä valittiin äänien implementointiin konseptipohjainen näkökanta, ja luotuja ratkaisuja voidaan hyödyntää vastaavien äänijärjestelmien kehityksessä, sekä työssä käytettyjen työkalujen hallinnassa.

Äänijärjestelmien toteutukset todettiin toimiviksi, ja implementoinnin suunnittelu- ja kehitysvaiheissa onnistuttiin suunnitelman mukaisesti.

Asiasanat: fmod, unity, ääniohjelmointi, kaupunginrakennuspeli

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Business Information Systems
Game Development

ETELÄMÄKI, JUUSO
Design and application of audio implementation for a city-building game
Bachelor's thesis 30 pages
July 2020

This thesis covers aspects of audio implementation in game development focusing on city-building games, and the main audio systems behind them. The objective was to research the features of FMOD middleware software, and to find out how they can be used to develop solutions for the main audio systems used in city-building games.

The purpose of this thesis was to implement said systems to a Unity game engine project, heavily focusing on describing the process. Additionally, this thesis focuses on the challenges and requirements for creating audio systems to city-building games.

The theoretic framework for this thesis is based on literature and web sources surrounding the topic of audio implementation in games, as well as the official documentation of the tools used in the implementation process. The thesis took a concept-based approach to implementing audio systems, and the developed solutions can be benefitted from when developing similar implementations. This work can also be used as an introduction to using the tools, which the audio systems were developed with.

Planning and developing the audio implementations were deemed successful, as the produced audio systems were established operational.

Key words: fmod, unity, audio programming, city-building game

SISÄLLYS

ERITYISSANASTO

Kaupunginrakennuspeli	Genre peleille, joissa pääasiallinen tarkoitus on rakentaa kaupunkeja tai kehittää yhteiskuntaa yleensä tyhjästä alkutilanteesta.
Äänituottaja	Pelin sisäinen äänilähde, jota kuunnellen äänen kuuntelija rakentaa äänen tilassa.
Äänikuuntelija	Ääniemitterien/lähteiden vastaanottaja, joka toistaa tuottajien tuottamat äänet joko pelimaailman sijainnin laskien 3D-tilassa tai sellaisenaan.
Äänitapahtuma	Pelimoottorin sisäiselle äänelle varattu ”tapahtuma”. Toimii välikätenä pelimoottorin ja äänikirjaston kanssa.
Ääni-instanssi	Äänen toistolle varattu esiintymä pelimaailmassa.
Snapshot	Kokoelma ääni-instansseja, joita voidaan ohjata ryhmänä.

1 JOHDANTO

Videopelejä pidetään usein pääsääntöisesti visuaalisena mediana (Ekman 2005). Äänien merkitys peleissä kasvaa teknologian kehittyessä, ja nykyajan ohjelmistovaihtoehdot mahdollistavat korkean tason äänitoteutuksien luomisen projektien koosta huolimatta. Äänien suunnittelua ja tuotantoa varten on kuitenkin ääniohjelmoijan rooli vaadittu onnistuneen toteutuksen saavuttamiseksi. Pelimoottorit ja väliohjelmistot ovat kehittyneet pisteeseen, jossa ohjelmoijien on mahdollista tuottaa toimivia äänitoteutuksia ilman aiempaa aiheeseen liittyvää kokemusta tai suunnitteluun perehtymistä. Kuitenkin projektien koon kasvaessa, niiden monimutkaisuus kasvaa myös äänijärjestelmissä. Äänijärjestelmien ja erilaisten ääni-instanssien määrän kasvaessa ääniohjelmoijan roolin tärkeys kasvaa. Järjestelmien odotetaan usein skaalautuvan projektin edetessä ylläpitävän myös mahdollisimman hyvää suorituskykyä.

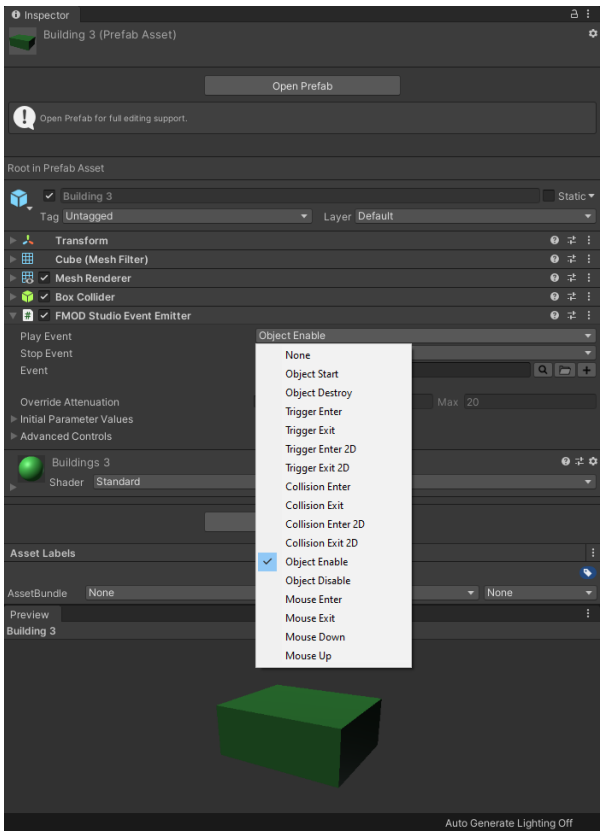
Kaupunginrakennuspelit videopeligenrenä kuvataan yleisesti pelejä, joissa simuloidaan kaupunki tai muuta asutusaluetta ja pelaajan on tarkoitus edistää alueen elintasoa johtajana. Kaupunginrakennuspelit kategorioidaan sandbox-peleiksi, eli peli etenee epälineaarisesti, ja peleissä annetaan pelaajalle vapaat kädet eri tehtävien toteutustavoiksi (Bereitschaft 2015). Peleissä kuvataan pelaajan perspektiiviä alueen yläpuolelta katsoen alas. Kaupunginrakennuspelien äänimaailmassa äänet toistuvat usein, ja rakentaminen sekä erilaisten resurssien keräys on merkittävä osa pelin keskeistä silmukkaa. Rakennukset, pelien hahmojen interaktiot ja erilaiset tehtävät toistuvat pelin edetessä, ja äänimaailman toimivan varianssin tärkeys kasvaa.

2 Äänijärjestelmien suunnittelu

2.1 Rakennuksien äänijärjestelmät

Äänien tallentamisen lähestymistapana käytettiin äänipankkipohjaista suunnittelua, eli äänitapahtumille kehitettiin uudelleenkäytettävä säilytystapa. FMOD-väliohjelmistolla luodaan ”master bank” -niminen äänipankki, joka sisältää kaikki projektin äänitapahtumat. Lähestymistapa tapahtumien tallennukseen on konseptissa sama. Äänipankki-tyyppinen kehitystapa mahdollistaa äänipankkien pohjan uudelleenkäytön erityyppisillä objekteilla, jotka jakavat samankaltaista toiminnallisuutta. Tavoite on käyttää samaa äänien säilytystapaa esimerkiksi kahden eri ääniä toistavan rakennuksen kanssa. Äänipankkipohjien luonnissa käytettiin Unityn ScriptableObject luokkaa, jolla luotiin erilliset tiedostot äänipankeille rakennuskohtaisesti. Äänipankki pohjainen lähestymistapa helpottaa järjestyksen ylläpitoa projektissa eri äänijärjestelmien kesken.

Rakennukset ovat suuri osa citybuilder pelien keskeistä pelisilmukkaa. Äänijärjestelmiä kehittäessä voidaan äänien osalta jakaa rakennuksia erityyppisiksi. Rakennukset, joissa äänien toistoon ei vaadita pidemmälle kehitettyä pelilogiikkaa. Esimerkiksi rakennukset, joissa samaa ääntä toistetaan jatkuvasti tai kertaluontoisesti niiden luontivaiheessa. Vastaavanlaiset yksinkertaiset äänitoteutukset voidaan kehittää jo pelkästään FMOD väliohjelmiston Unity puolen komponenttien natiiveilla toistovaihtoehdoilla (kuva 1). FMOD-ohjelmiston natiiveja komponentteja Unityssä ovat ”StudioEventEmitter”, joka toimii äänituottajana äänitapahtumille, sekä ”StudioEventListener” äänikuuntelija, jonka avulla 3D-äänien sijainnit voidaan paikantaa oikein. Rakennuksiin valittiin toistotapahtuman aloitukseen funktio OnEnable ja lopetukseen funktio OnDisable. Tämä tarkoittaa, että toisto alkaa, kun peliobjekti asetetaan aktiiviseksi pelinäkymässä, ja se loppuu, kun objekti asetetaan ei-aktiiviseksi tai se poistetaan. Muita sopivia funktioita ovat OnStart, jossa toisto aloitetaan vain kerran objektin luontivaiheessa, ja OnDestroy, jossa voidaan asettaa toistotapahtuma objektin tuhoutuessa.



KUVA 1. FMOD komponenttien natiivit toistovaihtoehdot

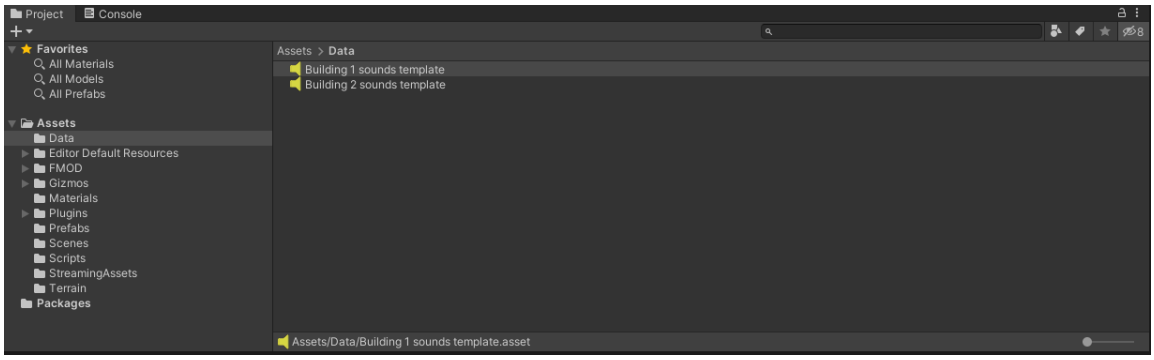
Kompleksisempien rakennusten äänijärjestelmissä vaaditaan pelilogiikassa tukea erilaisten tilojen seuraamiseen. Rakennuksien äänijärjestelmää tehdessä on oletettu, että rakennuksilla on käytössä ”finite state machine” koodausmalli, eli äärellinen automaatti malli. Äärellinen automaatti on tietojenkäsittelyssä käytetty käsite, joka viittaa abstraktin koneen mallintamiseen, jossa esiintyy rajattu määrä erilaisia tiloja (Memon 2013). Ohjelmoinnissa käsite implementoidaan tiloina esimerkiksi peliobjektille, jossa objektilla on mahdollista olla aktiivisena vain yksi tila kerrallaan. Mallin avulla nimetään peliobjektilla ajankohtainen tila, jonka avulla vastaavan tilan äänitapahtuma voidaan toistaa. Tämän perusteella rakennettiin äänipankeille uudelleenkäytettävä pohja, voitiin nimetä äänitapahtumille rakennusta vastaava tila.

Rakennuksien äänien varastointiin luotiin Unityn ScriptableObject luokasta periytyvä BuildingSoundBankTemplate luokka, joka tulee vastaamaan rakennuskohtaisesti äänitapahtumien säilytyksestä (kuva 2 ja 3). Rakennuksien tilojen nimeämiseksi luotiin enum-tyyppinen BuildingState luokka, joka vastaa äärellisen automaatti mallin yhtä tilaa. Tähän sisällytetään käytetyt rakennustilat Idle-, Working-, ja Off-tilat esimerkkeinä. BuildingSoundBank luokkaan lisätään taulukko,

johon tallennetaan FMOD-äänitapahtumien referenssit string-tyyppisinä. FMOD:in Unity integraatiossa on mukana myös EventRef attribuutti, jota voidaan string-tyyppisillä muuttujilla käyttää äänitapahtumien helppoon valintaan Unity-pelimoottorissa.

```
using UnityEngine;
using FMODUnity;

[CreateAssetMenu(menuName = "Assets/BuildingSoundBankTemplate")]
public class BuildingSoundBankTemplate : ScriptableObject
{
    [System.Serializable]
    public class BuildingSoundEvent
    {
        [EventRef]
        public string stateSound;
        public BuildingState state;
    }

    [SerializeField]
    BuildingSoundEvent[] buildingSoundEvents = null;

    public string GetStateSound(BuildingState state)
    {
        for (int i = 0; i < buildingSoundEvents.Length; i++)
        {
            if (buildingSoundEvents[i].state == state)
            {
                return buildingSoundEvents[i].stateSound;
            }
        }
        return string.Empty;
    }
}
```

KUVA 2. ”BuildingSoundBankTemplate” äänipankkipohja luokka



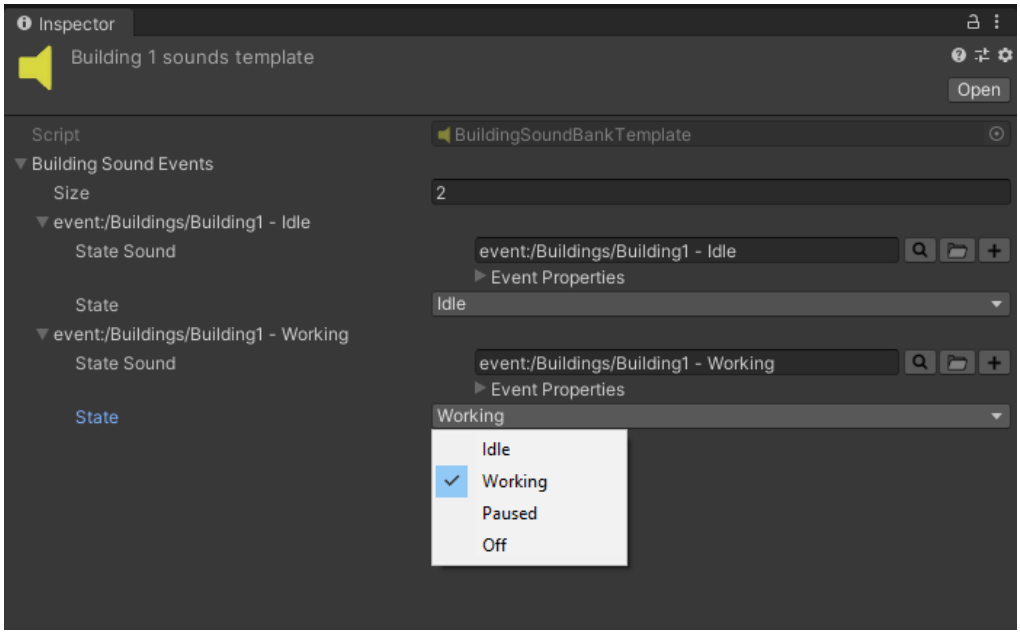
KUVA 3. Rakennuksien äänipankit Unityn tiedosto näkymässä

Rakennuksille luotiin BuildingSoundPlayer luokka, jonne luotiin logiikka äänentoistoa varten (kuva 4). BuildingSoundPlayer luokka sisällytetään kaikkiin ääniä toistaviin rakennuksiin, joissa on käytössä erilaisia tiloja. Luokkaan sisällytetään äänituottajille ja äänipankille serialisoitu kenttä, eli referenssit lisätään Unityn

editorinäkymässä peliobjektille (kuva 5). Funktio PlayStateSound ottaa parametrina enum-tyyppisen BuildingState tilan, joka vastaa rakennuksen sen hetkistä tilaa pelissä. Tilaa vastaava äänitapahtuma haetaan tämän jälkeen rakennuksen ”soundBank” äänipankista, ja löytyessä toistetaan äänituottajalla ja palautuksen ollessa tyhjä äänituottaja pysäytetään.

```
using UnityEngine;
using FMODUnity;

public class BuildingSoundPlayer : MonoBehaviour
{
    [SerializeField]
    StudioEventEmitter buildingEmitter = null;

    [SerializeField]
    BuildingSoundBankTemplate soundBank = null;

    public void PlayStateSound(BuildingState buildingState)
    {
        string stateSoundEvent = "";

        stateSoundEvent = soundBank.GetStateSound(buildingState);

        if (!string.IsNullOrEmpty(stateSoundEvent))
        {
            buildingEmitter.Stop();
            buildingEmitter.Event = stateSoundEvent;
            buildingEmitter.Play();
        }
        else
            buildingEmitter.Stop();
    }

```

KUVA 4. ”BuildingSoundPlayer” luokka

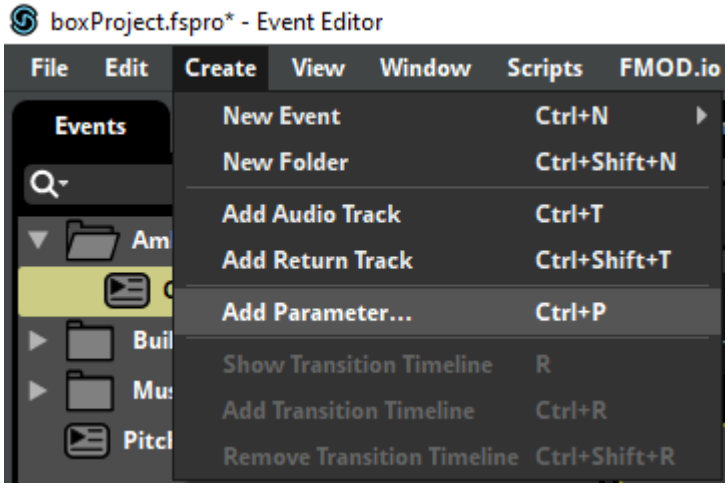


KUVA 5. Äänipankki Unityn inspector ikkunassa ja ääntä vastaavan tilan valinta

2.2 Ilmapiiriääni

Kaupunginrakennuspeleissä simuloidaan ambient-tyyppisillä eli ilmapiiriäänillä sääolosuhteita, ympäristöä, sekä ilmakehää suhteessa pelikameran korkeuteen. Näiden toteutus tehdään FMOD:in parametrimuutoksien käyttöönotolla ilmapiiri äänitapahtumiin.

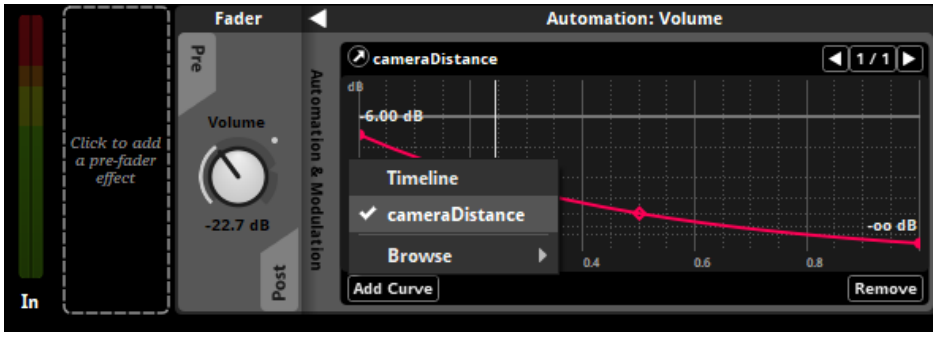
Ilmapiiri äänitapahtumien toistoon kehitettiin AmbientPlayer luokka, jolla säädetään FMOD:issa määritettyjä parametrejä. Parametrit FMOD:issa ovat määritettäviä arvoja, jotka mahdollistavat FMOD-projektin ohjaamisen (FMOD Parameters n.d.). Parametrien tarkoitus on yhdistää pelissä reaaliaikaisesti tapahtuvat arvomuutokset FMOD-studio-ohjelmistossa määritettyjen äänitapahtumien muutoksiin. Parametrin lisääminen FMOD-studio-projektiin onnistuu työkalupalkin ”Create” osiosta ”Add Parameter” vaihtoehdolla (kuva 6).



KUVA 6. Parametrin luonti FMOD-studio-ohjelmistossa

Pelin korkeussuhteen äänien simulointia varten luotiin ”ambience”-niminen äänitapahtuma FMOD-studio-ohjelmistossa, johon lisätään float-tyyppinen ”CameraDistance”-niminen parametri (kuva 7). ”CameraDistance” parametrillä seurataan pelikameran etäisyyttä. Useimmiten peleissä on kameran etäisyyslaskennassa käytetty tarkempaa laskentaa asetettujen minimi- ja maksimietäisyyksien perusteella, mutta konseptissa käytettiin kameran Y-akselin arvoa saman lopputuloksen simulointiin. Parametrin liittäminen äänitapahtumaan onnistuu painamalla oi-

keaa hiiren näppäintä halutun säädön kohdalla. Kameran etäisyyttä käytettiin ilmapiiri ääniraitojen äänentason ohjaamiseen, mutta parametriä voidaan käyttää myös muiden säätömahdollisuutta tarjoavien efektien ohjaamiseen.



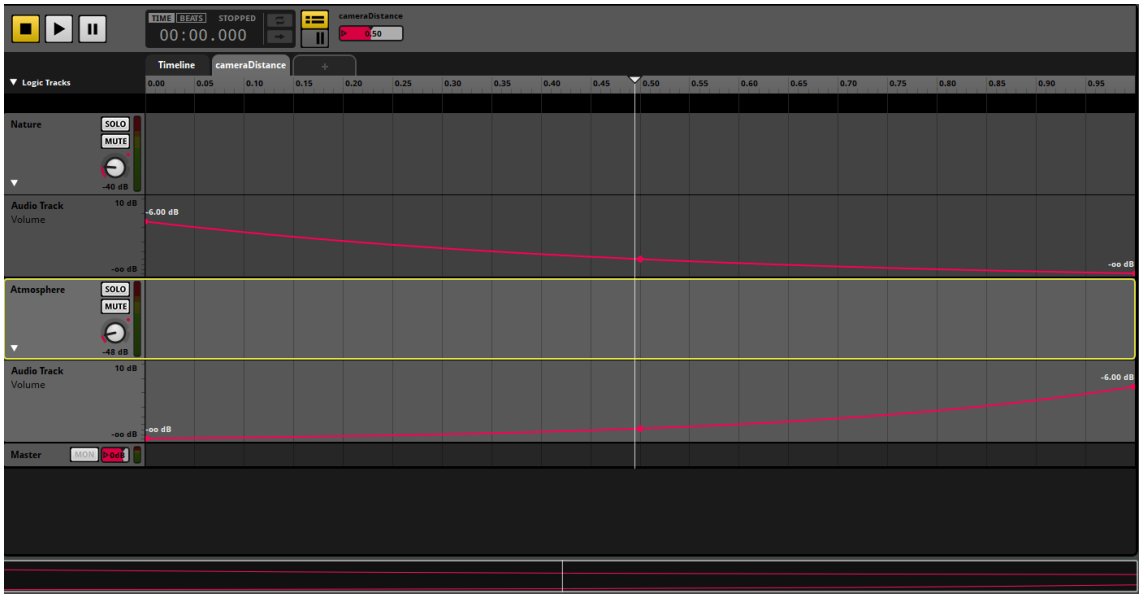
KUVA 7. ”cameraDistance” parametrin lisääminen ilmapiiri äänitapahtuman volyymi automaatioon

Pelikameran korkeusarvo suhteutettiin arvojen 0 ja 1 välille Unityn ”mathf”-kirjaston InverseLerp-funktiolla. InverseLerp-funktio palauttaa prosentuaalisen arvon sille annetun minimin ja maksimin väliltä. Tämä helpottaa FMOD-projektissa tehtyjen säädöksien säilymisessä, mikäli kameran minimi- ja maksimikorkeuksia muutetaan projektissa. AmbientPlayer luokassa minimi- ja maksimiarvon suhdetta verrataan kameran y-akselin arvoon, minimi nimettynä minHeight muuttujaksi, ja maksimi maxHeight muuttujaksi (kuva 8). Äänituottaja saadaan EventInstance muuttujasta SetParameterByName funktiota kutsumalla säädettyä parametriarvoa sen nimellä.

```
using FMODUnity;
using UnityEngine;

public class AmbientPlayer : MonoBehaviour
{
    const string CAMERA_DISTANCE_PARAMETER = "cameraDistance";

    [SerializeField]
    Camera cityCamera = null;

    [SerializeField]
    StudioEventEmitter ambienceEmitter = null;

    float minHeight = 0f;
    float maxHeight = 20f;

    float currentHeightPercentage = 0f;

    void Update()
    {
        currentHeightPercentage = Mathf.InverseLerp(minHeight, maxHeight, cityCamera.transform.position.y);

        ambienceEmitter.EventInstance.setParameterByName(CAMERA_DISTANCE_PARAMETER, currentHeightPercentage);
    }
}
```

KUVA 8. AmbientPlayer luokka

Ilmapiiri äänitapahtumalle luotiin kaksi ääniraitaa, ”Nature” ja ”Atmosphere”, joiden on tarkoitus kuvata pelikameran korkeussuhteesta johtuvaa äänien muutosta. Ilmapiiri äänitapahtuman säätö toteutettiin siten, että kameran ollessa maksimikorkeudessaan, äänitapahtuman toistossa kuullaan vain ”Atmosphere” ääniraitaa, ja kameran ollessa minimi korkeudessaan on ”Nature” ääniraita ainoastaan kuultavissa. ”CameraDistance”-parametrin sijoittuessa minimi- ja maksimiarvojen välille volyymitasot koostuvat automaatiokäyrien suhteellisesta kohdasta (kuva 9). Automaatiokäyrä on kuvaaja, joka esittää muutettavan kohdan mahdollisia arvoja suhteessa muuttuvaan parametriin (FMOD Authoring Events n.d.).

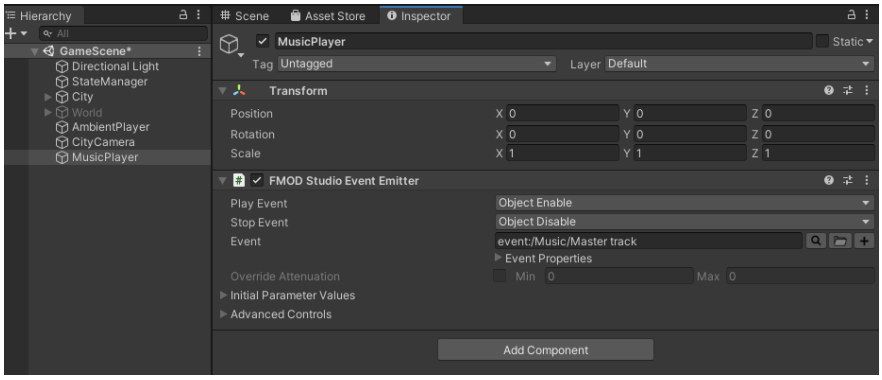


KUVA 9. Ilmapiiri äänitapahtuman ääniraitojen automaatiokäyrät punaisella

2.3 Pelimusiikin toistojärjestelmä

Musiikki kaupunginrakennuspeleissä toimii isona osana pelikokemusta. Pelien keskittyessä yhteen pelinäkymään ja muiden äänien toistuvuuden takia, on musiikin vaihtelevuuden avulla tarkoitus monipuolistaa pelaajan kokemusta. SimCity sarjan peleihin on kehitetty jukebox tyyppinen ratkaisu, joka vaihtelee eri kappaleiden välillä (Jolly 2012). Cities: Skylines pelissä on mahdollista ladata lisäsisältönä erilaisia ”music pack”-lisäosia, jotka sisältävät kappaleita pelin sisäiseen radiokanava-tyyppiseen musiikintoistojärjestelmään (Cities: Skylines Wiki n.d). Musiikintoistojärjestelmän implementoinnissa otettiin soittolista-tyyppinen lähestymistapa musiikin toistoon. Musiikin toistoon luotiin MusicPlayer peliobjekti Unity

projektiin, johon lisättiin FMOD:in äänituottaja komponentti. Komponentissa käytettiin FMOD:in natiiveista toistovaihtoehdoista OnEnable ja OnDisable vaihtoehtoja toiston aloittamiseen ja lopettamiseen (kuva 10).

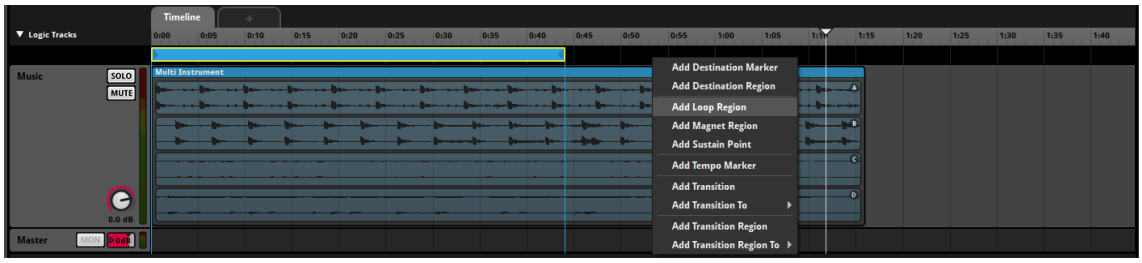


KUVA 10. MusicPlayer peliobjekti unityssä

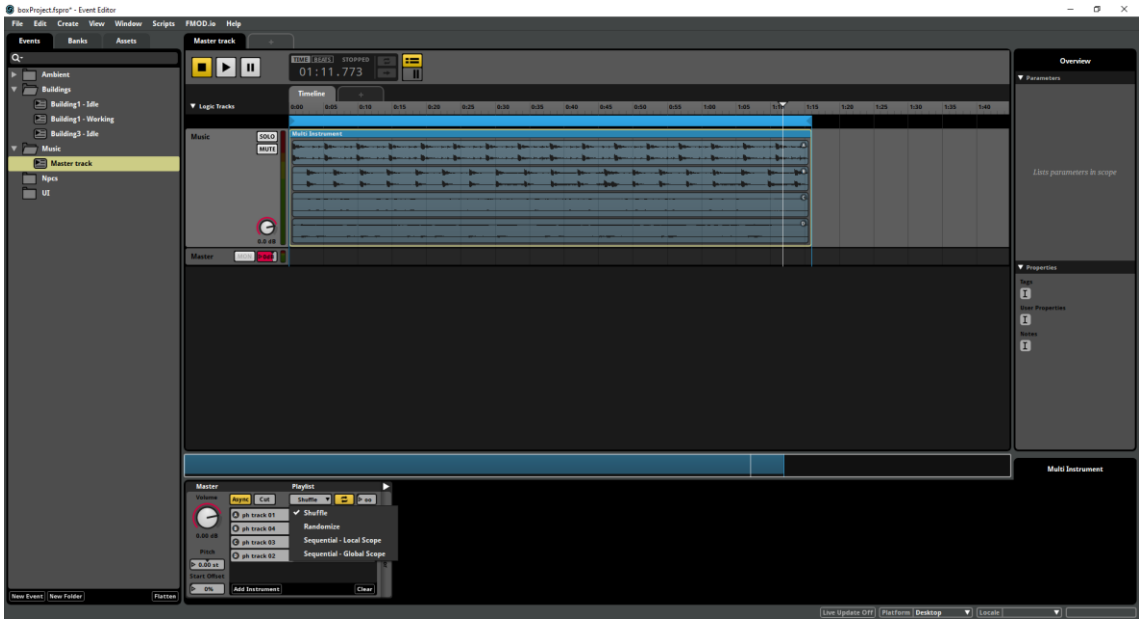
FMOD-studio sisältää ”playlist” nimisen soittolista-tyyppisen liitännäisen, johon on mahdollista lisätä monta ääninäytettä, joiden toistotapaan on saatavilla erilaisia vaihtoehtoja. komponentti lisäytyy äänitapahtumaan viemällä useampia ääninäytteitä yhteen ääniraidan kenttään kerralla. Lisättäviä kappaleita voi myös raahata suoraan soittolista komponentin kappalekenttään. Useimmiten toistettavat kappaleet ovat eri pituisia, ja ”Async” kentän valitsemisella saadaan toistotapahtuma jatkumaan kappaleiden loppuun asti niiden pituudesta huolimatta. FMOD-soittolistan toistotyypeistä musiikin toistoon on ”shuffle” sopivin (kuva 11). Shuffle-tyyppisessä toistotavassa soittolistan kappaleet toistetaan satunnaisessa järjestyksessä. Kun soittolista sisältää kolme kappaletta tai enemmän, ei samaa kappaletta toisteta koskaan kahta kertaa peräkkäin. (FMOD Glossary n.d.)

KUVA 11. FMOD soittolista-liitännäinen

Äänitapahtuman aikajanalle lisätään loop region, jotta kappaleiden toisto on jatkuvaa. Loop region on looginen osoitin (kuva 12), joka määrittää kaksi pistettä äänitapahtuman aikajanalta, jonka välissä toiston tulee tapahtua (FMOD Glossary n.d.). Näillä asetuksilla saadaan toimiva soittolista-tyyppinen musiikintoistojärjestelmä projektiin. Musiikki äänitapahtuman kokonaiskuva (kuva 13).



KUVA 12. Loop regionin lisääminen FMOD-studiossa



KUVA 13. Musiikin multitrack ja shuffle FMOD-studiossa

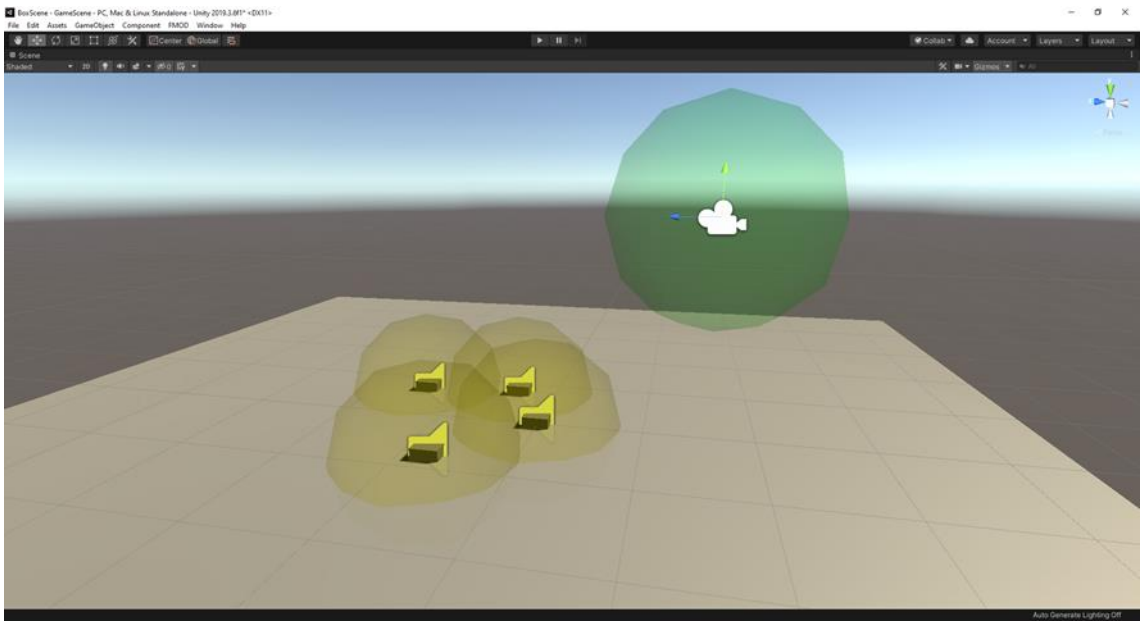
3 Äänimaailman rakentaminen

3.1 3D-äänet ja ympäristön virtualisointi

Pelien äänimaailma rakennetaan kahdentyyppisistä äänistä. ”2D-äänet ovat yleensä paikattomia, mikä tarkoittaa, että ääni toistetaan samantasoisesti oikeasta ja vasemmasta kaiuttimesta” (Sanjay 2013). 3D-äänien toimivuuteen vaaditaan niille asetettu paikka pelimaailmassa sekä kuuntelija, jonka paikkaan verrattuna 3D-äänien panorointi ja volyymi säätyvät. Kuuntelijaobjekti Studio Listener täytyy kiinnittää pelin kameraobjektiin, jotta 3D-äänet sijoittuvat oikein (FMOD Game Components n.d.).

Kaupunginrakennuspelien päämääränä on rakentaa ja edistää infrastruktuuria pelaajalle annetun alueen sisällä. Pelaaja valitsee rakennuksien paikat sekä kaupungin edistämistoiminnot kuten palkat ja työprioriteetit ja edistää kaupungin infrastruktuuria (Bereitschaft 2015). Pelin edetessä ja pelaajan resurssien kasvaessa, myös alueen sisältö kasvaa laaja-alaisesti. Äänien osalta voidaan siis olettaa, että 3D-maailman äänilähteiden määrät kasvavat myös. Jotta pelaajalle ei soitettaisi kaikkea pelinäkymän sisältöä yhtäaikaisesti, on syytä perehtyä erilaisiin menetelmiin 3D-äänien hallinnassa.

FMOD-studio-ohjelmisto sisältää erilaisia liitännäisiä äänien prosessointiin. Lisäksi liitännäisistä voidaan luoda omia ”preset”-pohjia, eli yhteen äänitapahtumaan tehdyt liitännäisasetukset voidaan tallentaa ja uudelleenkäyttää toisissa äänitapahtumissa. 3D-äänien toistamiseen vaaditaan Spatializer-liitännäisen käyttöä äänien paikkojen simulointiin 3D-ympäristössä. Äänen spatialisointia voidaan kuvata prosessina, jossa otetaan äänitiedosto, ja muutetaan se kuulostamaan pelimaailmassa olevalta (FMOD White Papers | Spatial Audio n.d.). Spatializer liitännäisellä voidaan määrittää minimi- ja maksimietäisyys äänen kuuntelijan suhteen sekä määrittää etäisyyden muutoksen käytös äänen volyymiin (kuva 14).



KUVA 14. Visualisointi 3D-äänien kantamasta pelinäkymässä, rakennuksien äänikantama keltaisina palloina ja kameran kuuntelijaobjektin kantama vihreänä pallona.

3.2 Instanssien hallinta

Ääniä toistettaessa varataan jokaisella äänelle aina oma ääni-instanssinsa, ja pitkälle kehittyneessä projektissa, jossa äänien määrä kasvaa on tärkeää kiinnittää huomiota niiden määrään ja hallintaan. Yhtäaikaisten ääni-instanssien rajoittaminen jättää varaa parempaan resurssien hallintaan, kuormittaen vähemmän pelaajan laitetta (FMOD Advanced Topics n.d.). FMOD-äänitapahtumia toistettaessa yksi helppo tapa rajoittaa instanssien määrää pelissä, on rajoittaa ja käyttää uudelleen äänentuottajia. Äänipankki-tyyppisellä lähestymistavalla on tarkoitus toistaa järjestelmissä äänitapahtumat yhden äänentuottajan kautta sijoittamalla uusi äänitapahtuma vanhan tilalle.

FMOD-studiossa on myös mahdollista rajoittaa instanssimäärää ”max instances”-kohdalla tapahtumakohtaisesti, sekä erilaiset ”stealing”-optiot mahdollistavat yhtäaikaisten maksimiäänimäärien rajoituksen. FMOD:in rajoitukset vaikuttavat ainoastaan äänien suoraan toistoon, ja instanssit varataan ja toistetaan FMOD:in rajoituksilla virtuaalisena. Virtualisoitu äänitapahtuma on ääni-instanssi, joka toistetaan, mutta ei tuota ääntä (FMOD Advanced Topics n.d.). Tämä tarkoittaa, että äänitapahtuman varataan muistia toistoa varten ja äänitapahtumaa toistetaan kuitenkin hiljaisena. Esimerkiksi kiertävien äänitapahtumien kanssa

voidaan pelaajalle välittää oikeat äänet FMOD:in instanssihallinnalla, mutta kaiken alla voi olla useita hiljaisia instansseja käytössä mikä taas tarkoittaa turhaa resurssien käyttöä. Kiertävät äänitapahtumat ovat äänitapahtumia, joissa äänitapahtuma aloittaa toiston alusta automaattisesti ääniraidan päättyessä.

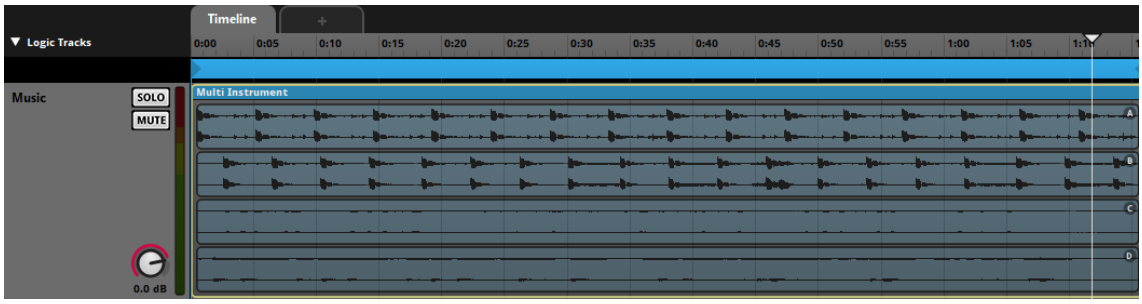
FMOD-studio-ohjelmisto sisältää profiler-toiminnon, jolla voidaan tarkkailla ääniinstansseja reaaliaikaisesti pelinäkymässä. Profilerilla on myös mahdollista tallentaa otanta, jossa pelinäkymässä tallennetulta ajalta voidaan tallennuksen jälkeen tarkastella aktiivisia instansseja.

3.3 Varianssin toteutus

FMOD-äänitapahtumiin voidaan sisällyttää monia ääninäytteitä, ja äänitapahtumia, jotka sisältävät enemmän kuin yhden näytteen kutsutaan ”multi-instrument”-äänitapahtumiksi. Yleisempiä varianssin toteutusmuotoja on äänen sävelkorkeuden satunnaistettu muutos, joka voidaan toteuttaa äänitapahtumakohtaisesti. Helpoiten tämä onnistuu pitämällä windows-käyttöjärjestelmässä alt-näppäintä pohjassa ja raahaamalla äänitapahtuman ”pitch” kenttää ylöspäin. ”Pitch” kentän sisälle ilmestyy vihreä alue, jonka välille toistettavan äänitapahtuman sävelkorkeus sijoittuu satunnaisesti (kuva 15). FMOD sisältää myös ”pitch shifter”-liitännäisen, jolla voidaan luoda vastaavaa varianssia ääniraitakohtaisesti (kuva 16). Esimerkiksi ”multi-instrument”-tyyppiseen äänitapahtumaan, joka sisältää useita äänitiedostoja yhden ääniraidan sisällä (kuva 17).

KUVA 15. Äänitapahtuman sävelkorkeuden varianssi, "pitch" kenttä vasemmalla

KUVA 16. FMOD:in ”pitch shifter”-liitännäinen, sävelkorkeuden varianssi merkattuna vihreänä kenttänä

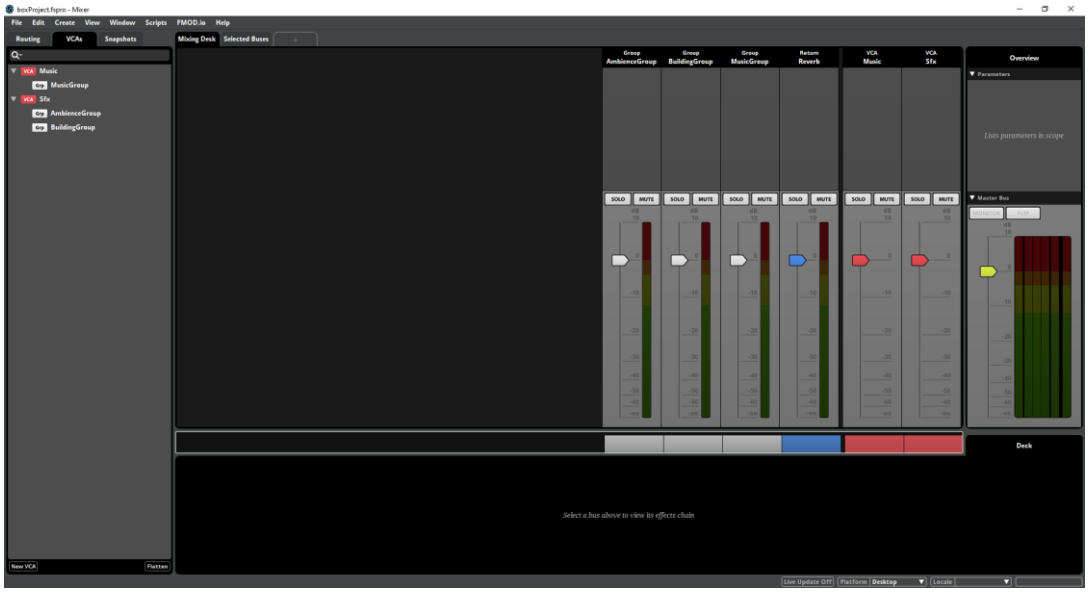


KUVA 17. ”Multi-instrument”-tyyppinen ääniraita

4 Pelin tilojen äänihallinta

4.1 Mikseri ja äänien ryhmittely

FMOD-ohjelmiston mikseri-ikkuna sisältää työkaluja ja toimintoja, jotka mahdollistavat äänitapahtumien reitityksen, efektien lisäämisen signaaliketjuihin ja määrittämisen miten lisätyt efektit käyttäytyvät vastauksena pelin tapahtumille (FMOD Mixing n.d.). ”VCA audio bus” on FMOD:issa käytettävä suurempi signaaliketjujen yhdistäjä. Signaaliketju on esitysmuoto äänisignaalin prosessoinnin tapahtumista, jotka tapahtuvat äänisignaaliin kulkiessa ääniraidan läpi (FMOD Mixing n.d.). VCA-tyyppisiä ryhmiä käytetään muutaman pääasiallisen suurryhmän kontrolloijana. FMOD-projektiin luotiin ”Music” ja ”Sfx” nimiset VCA-ryhmät, joilla niihin sijoitettujen ryhmien äänentasoa säädetään pelissä (kuva 18). VCA-ryhmien lisääminen onnistuu mikseri näkymän ”VCAs”-välilehdellä ollessa työkalupalkista valitsemalla ”create”-osion alta ”New VCA”. Pienempien ryhmien sisällyttäminen VCA-ryhmään onnistuu raahaamalla ne halutun VCA-ryhmän sisälle ”VCAs”-välilehdessä.



KUVA 18. ”VCAs”-välilehti FMOD-studion mikserinäkymässä

Ryhmien lisääminen onnistuu ”Routing”-välilehdessä samoin kuin VCA-ryhmien luonti ”VCAs”-välilehdessä (kuva 19). Ryhmiin saadaan sijoitettua halutut äänita-

pahtumat raahaamalla ne halutun ryhmäkuvakkeen päälle. FMOD-studio-ohjelmistossa luotiin kolme ryhmää, ”AmbienceGroup”, ”BuildingGroup” sekä ”MusicGroup” nimiset ryhmät, joihin sisällytettiin niihin kuuluvat äänitapahtumat.

KUVA 19. FMOD-studio-ohjelmiston ”Routing- välilehti mikserinäkymässä

4.2 Snapshotit

Snapshotit ovat kokoelma tiettyjen ääni-instanssien muutoksista, jotka mahdollistavat niihin lisättyjen instanssien erilaisia hallintatapoja (FMOD Mixing n.d.). Esimerkiksi snapshot voi koostua tietyn pelinäkymän äänitapahtumista, ja snapshotin avulla voidaan lisätä niihin kaikkiin erilaisia efektejä, kontrolloida eri asetuksia tai pysäyttää kokonaan. Snapshotteja voidaan myös hallita yhtäaikaisesti, joka mahdollistaa tarkemman ohjauksen erilaisten ääniryhmien kesken. Unityn oma äänijärjestelmä ja useimmat väliohjelmistot mahdollistavat äänien ryhmittelyn snapshotteihin, joka mahdollistaa äänien hallinnan tilannekohtaisesti.

4.3 Pelinopeus

Pelinopeuden säätö on yleisimpiä mekaniikoita kaupunginrakennuspeleissä. Pelien pääasiallisen tavoitteen keskittyessä toistuviin tehtäviin, on nopeutussäädöstä kasvanut genren standardi pelisession etenemisen helpottamiseksi. Äänijärjestelmissä pelinopeuden vaihtelu esittää haasteita toteutukseen. FMOD-vä-

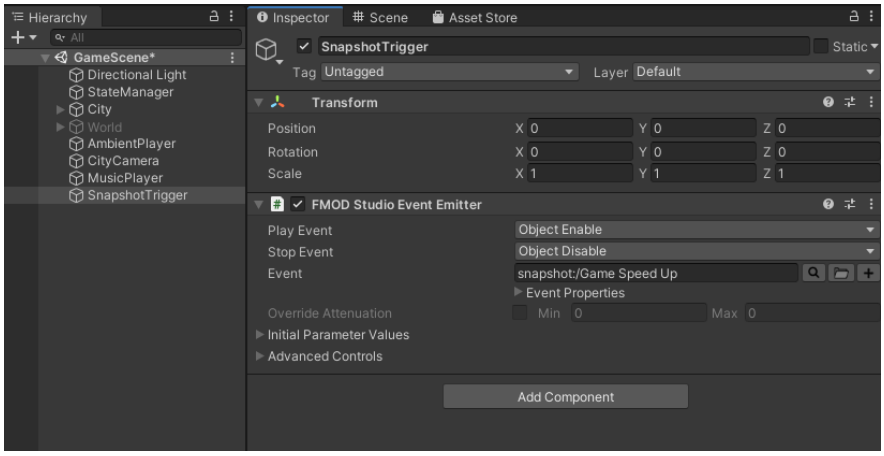
liohjelmiston tai Unityn omat äänityökalut eivät tarjoa suoraa tapaa äänitapahtumien nopeutukseen tai hidastamiseen. Ongelman voi osittain ratkaista käyttämällä useita ”pitch shifter”-liitännäisiä ja äänitapahtumien tempoa säätämällä suhteessa nostettuun sävelkorkeuteen, mutta vastaava käyttö tuo ääniin mukana ei-haluttuja vääristymiä. Lisäksi digitaaliseen signaaliin lisätyt efektit käyttävät tietokoneen resursseja, joten ylimääräistä käyttöä tulisi välttää (FMOD White Papers | Spatial Audio n.d.). Vaihtoehtoinen toteutus äänien nopeuden säätöön toteutuksen kaltaisissa tapauksissa on tiettyjen äänien mykistäminen pelinopeuden kasvaessa. Esimerkiksi alemman prioriteetin tapahtumat, joissa äänet ovat suoraan kytköksissä animaatioon tai vastaavaan ajastettuun tapahtumaan. Jatkuvat äänet, kuten rakennuksien kiertävät toimintaäänet toimivat usein sellaisenaan, jos äänet eivät ole suoraan sidottuja pelin visuaalisiin tapahtumiin.

Pelinopeuden muutoksiin liittyvät säädökset toteutetaan FMOD:in snapshoteilla, joilla ääniä voidaan helposti jakaa eri nopeustiloihin tarpeen vaatiessa. Pelinopeuden muutokselle luotiin ”Game Speed Up” -niminen snapshot, jossa hiljennetään ”LowerPriorityGroup”-niminen äänitapahtumaryhmä (kuva 20).

KUVA 20. ”Game Speed Up” snapshot ja ”LowerPriorityGroup” ääniryhmän hiljentäminen

Snapshotit toistetaan pelissä kuten muutkin äänitapahtumat, eli niissä voidaan käyttää FMOD:in natiiveja toistovaihtoehtoja (kuva 21), tai niitä voidaan ohjata pelin koodista käsin (kuva 22). Äänituottajan käyttämä snapshot voidaan valita

Unityn päänäkymästä käsin vaikka käytettäisiin koodipohjaista toteutusta sen ohjaamiseen.



KUVA 21. "SnapshotTrigger" peliobjekti FMOD:in toistovaihtoehdoilla

```
using UnityEngine;
using FMODUnity;

public class SnapshotTrigger : MonoBehaviour
{
    StudioEventEmitter snapshotEmitter = null;

    void Start()
    {
        snapshotEmitter = GetComponent<StudioEventEmitter>();
    }

    //boolean tyyppinen state parametri lähetetään joko true tai false arvolla,
    //joka määrittää aloitetaanko vai pysäytetäänkö snapshotin toisto.
    public void TriggerSnapshot(bool state)
    {
        if (state)
            snapshotEmitter.Play();
        else
            snapshotEmitter.Stop();
    }
}
```

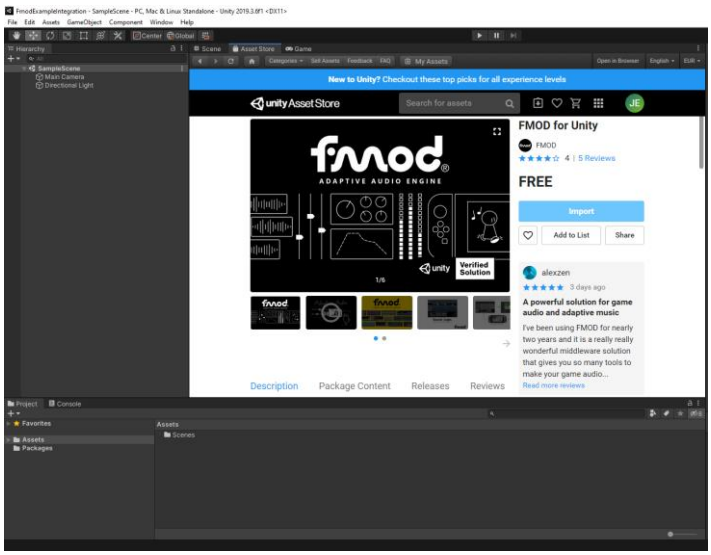
KUVA 22. Yksinkertaistettu koodipohjainen toteutus snapshotin toistoon

5 Äänijärjestelmien käyttöönotto

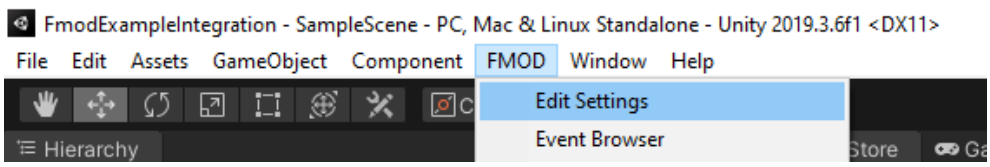
5.1 FMOD Unity integraatio

Työn implementointiosiossa on käytetty työkaluina Unity-pelimoottoria ja FMOD-studio-väliohjelmistoa. FMOD-väliohjelmiston integrointi Unity-pelimoottoriin on ohjelmiston kehittyessä helpottunut huomattavasti. FMOD-väliohjelmistoa käytettäessä on ladattava erillinen FMOD-studio-ohjelmisto, jolla luodaan ja muokataan projektiin tuotavia äänitapahtumia. FMOD-studiolla rakennetaan alustakohtainen äänipankki, johon Unityssä viitataan ääniä haettaessa ja toistaessa.

FMOD:in työkalujen käyttämiseen Unityssä suoritetaan FMOD integraatio projektiin. FMOD-ohjelmisto ladataan Unityn omasta Asset Store-palvelussa, jota voidaan käyttää suoraan Unity editorissa tai nettiselaimella (kuva 23). FMOD:in latauksen jälkeen on projektille asetettava vielä kohdekansio (kuvat 24 ja 25), mihin projektin äänipankki luodaan.



KUVA 23. FMOD integraation lataus unityn asset storesta

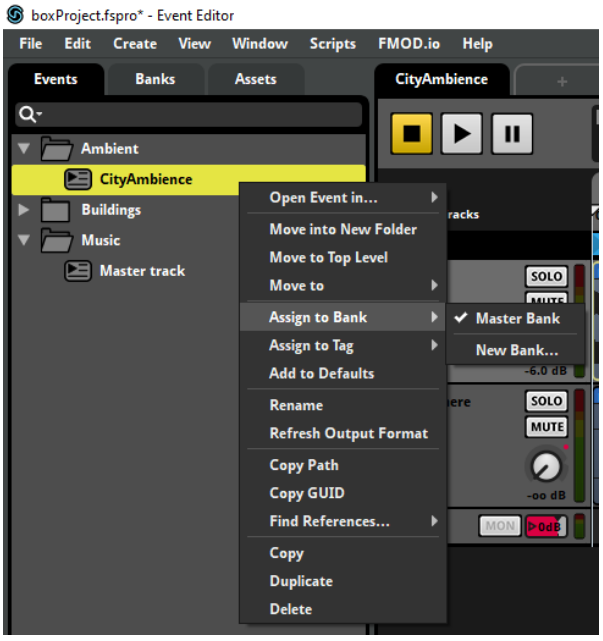


KUVA 24. FMOD-asetuksien valinta Unityn projekti näkymässä

KUVA 25. FMOD asetusnäkymä Unityn inspector ikkunassa

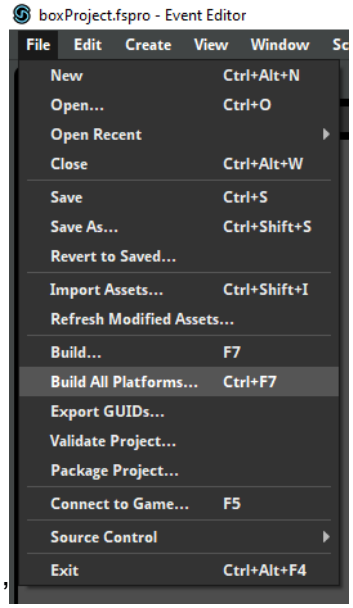
5.2 FMOD-studio-ohjelmisto

FMOD-studio-ohjelmisto toimii projektissa pääasiallisesti äänisuunnittelijan työkaluna, ja toiminnallisuus vastaa projektissa pääasiallisen äänipankin luonnista, äänitapahtumien editoinnista, efektien lisäämisestä ja hallintatapojen ohjaamisesta. Käyttöönotto Unity integraation kanssa studio-ohjelmistolla on täysin itsenäinen, eli FMOD-studio-ohjelmistolla voidaan käyttää yksinään esimerkiksi äänisuunnitteljan toimesta. Ainoastaan studiolla tehty äänipankki täytyy projektiin tuoda äänien toiminnallisuuden käyttöönottoon. Äänitiedostojen tuonti suoritetaan raahaamalla halutut äänitiedostot FMOD-studio-ohjelmiston ”assets”-listaan tai minkä tahansa äänitapahtuman aikajanalle. Äänitapahtuma tulee lisätä FMOD:issa rakennettavaan ”master bank”-äänipankkiin, ennen kuin sitä voidaan käyttää (kuva 26).



KUVA 26. Äänitapahtuman lisääminen master bank äänipankkiin

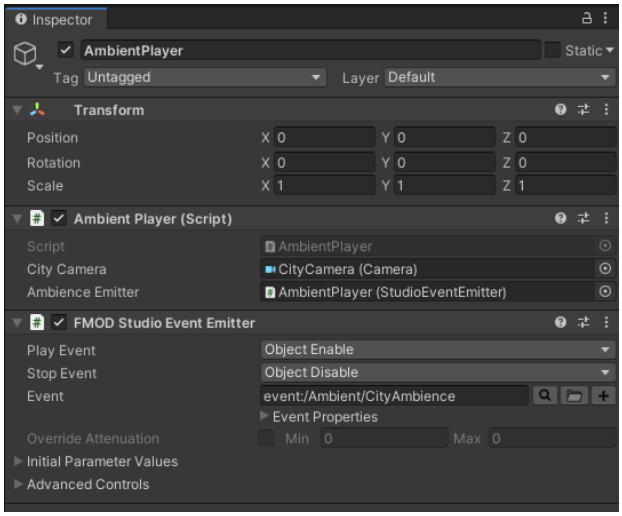
Äänitapahtumien lisäämisen rakennetaan ”master bank”-äänipankki. Tämä onnistuu FMOD-studion yläpalkista valitsemalla ”Build” tai ”Build all platforms” kohta (kuva 27). Mikäli kohdekansio on päivitetty oikein Unity-pelimoottorissa, Unity tunnistaa pankissa tapahtuneen muutoksen ja se on automaattisesti käyttövalmis projektiin.



KUVA 27. FMOD pankin rakennus FMOD-studio-ohjelmistossa

5.3 Unity-pelimoottoriin lisääminen

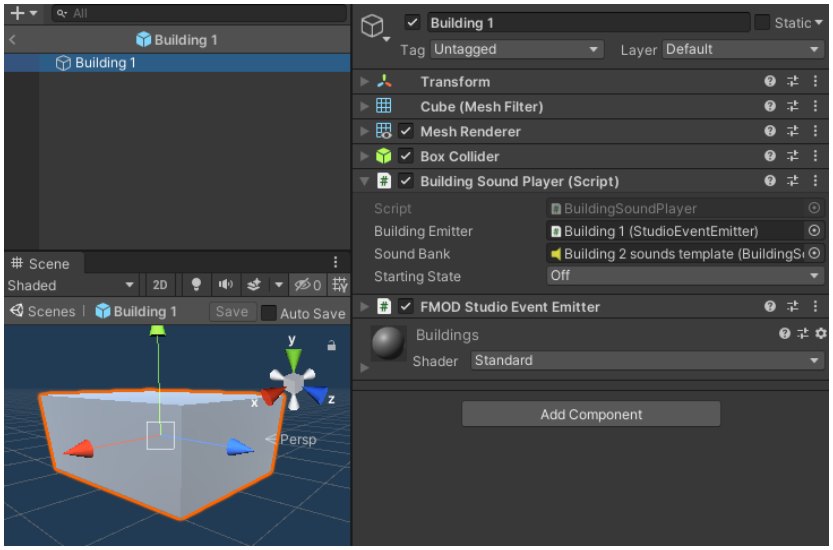
Musiikin- ja ilmapiirintoistojärjestelmät otetaan käyttöön lisäämällä Unityn näkymässä tyhjiin peliobjekteihin FMOD:in äänituottaja komponentit. Äänituottaja komponentteihin osoitetaan musiikkia ja ilmapiiriääntä vastaavat FMOD-äänitapahtumat. Ilmapiiriäänentoistojärjestelmän peliobjektiin lisätään Ambient Player luokka, johon tarvitaan pelikameran ja äänituottajan referenssit (kuva 28).



KUVA 28. AmbientPlayer luokka Unityn inspector-ikkunassa, äänituottaja ja pelin kamera lisättynä niille kuuluviin kenttiin

.

Rakennuksien äänijärjestelmä otetaan käyttöön lisäämällä BuildingSoundPlayer ja äänituottaja komponentit rakennuksen peliobjektiin. Peliobjektiin tulee osoittaa rakennusta vastaava äänipankki sekä äänituottaja BuildingSoundPlayer komponentin kenttiin (kuva 29).



KUVA 29. Rakennuksen äänijärjestelmä lisättynä rakennus peliobjektiin Unity-pelimoottorissa

6 Pohdinta

Äänien ohjelmointi ja suunnittelu ovat yleisesti toisiinsa sidottuja rooleja. Väliohjelmistojen kehitys on mahdollistanut suunnittelijatyön sijoittumisen lähemmäs pelimoottorissa työskentelyä, ja roolien välinen rajapinta on muuttunut vaikeammaksi vetää. Tässä opinnäytetyössä on pääasiallisena tavoitteena ollut suunnitella ja kehittää ratkaisuja implementoinnin osuudesta, kuitenkin työtä tehdessä nousi suuri haaste vetää raja, missä äänisuunnittelijan puoli alkaa ja mihin ohjelmoijan osuus loppuu. Työssä käydään silti osittain suunnittelijan tehtäviin kuuluvia osa-alueita toimivan kokonaisuuden rakentamiseksi.

Rakennuksien äänijärjestelmiä läpikäyvässä osiossa huomattiin haasteena ääniin pohjautuvien tapahtumien visualisointi lukijalle. Äänijärjestelmien toteuttaminen ilman pääasiallista projektia, vaati järjestelmien yleistämisen, jotta niitä on mahdollista kuvata toimivassa kontekstissa. Pelimusiikin toistoon perehtyvän osuuden tulokset vastaavat parhaimmillaan julkaistujen pelien toiminnallisuutta useimmissa tapauksissa. Prosessissa huomattiin, että yksinkertaisella toteutuksella voidaan kehittää kilpailukykyisiä ratkaisuja jo pelkästään työkalujen mahdollisuuksiin perehtymällä.

Äänimaailman rakentamiseen keskittyvässä osiossa kuvailtiin suunnitteluosuudessa kriittiseksi huomattua äänten rajoittamista, painottaen ennakoivaa suunnittelua projektin skaalautuvuuteen. Koin osa-alueen tarkoituksen olevan tärkeä kaupunginrakennuspelien simulaatioraskaassa pelisilmukassa, ja mahdollisten optimointien punnitsemisen jo suunnitteluvaiheessa hyödylliseksi.

Työn loppuosion työkalujen läpikäynnin tarkoituksena oli esitellä lukijalle toteutettujen äänijärjestelmien implementointiin vaadittavat ominaisuudet. Osio ei ollut kaiken kattava, mutta toimii lähtökohtaisesti tarvittavien elementtien esittelyssä työkalujen käyttöönotossa.

Työssä kehitettyjen ratkaisujen on mahdollista toimia pohjana vastaavia järjestelmiä vaativien peliprojektien kehityksessä. Tuotoksia kehittäessä niiden toiminnallisuutta yleistettiin jo valmiiksi niiden kuvaamisen helpottamiseksi, sekä yleispätevyyden parantamiseksi.

LÄHTEET

Bereitschaft, B. 2015. Gods of the City? Reflecting on City Building Games as an Early Introduction to Urban Systems. Luettu 20.05.2020. https://www.tandfonline.com/doi/full/10.1080/00221341.2015.1070366?scroll=top&needAccess=true

Ekman, I. 2005. Meaningful Noise: Understanding Sound Effects in Computer Games. Tutkimus. Hypermedia Laboratory. Tampereen yliopisto. Luettu 9.3.2020. https://www.researchgate.net/profile/Inger_Ekman/publication/224927551_Meaningful_Noise_Understanding_Sound_Effects_in_Computer_Games/links/546208640cf27487b4557a95.pdf

Memon, A. 2013. Advances in Computers. Elsevier Science & Technology.

FMOD Parameters. N.d. Web dokumentaatio. https://www.FMOD.com/resources/documentation-studio?version=2.0&page=parameters.html

FMOD Mixing. N.d. Web dokumentaatio. https://FMOD.com/resources/documentation-studio?version=1.10&page=mixing.html

FMOD Authoring Events N.d. Web dokumentaatio https://fmod.com/resources/documentation-studio?version=1.10&page=authoring-events.html

FMOD White Papers | Spatial Audio. N.d. Web dokumentaatio. https://FMOD.com/resources/documentation-api?version=2.0&page=white-papers-spatial-audio.html

FMOD Advanced Topics. N.d. Web dokumentaatio.
https://FMOD.com/resources/documentation-studio?version=1.10&page=advanced-topics.html

FMOD Glossary. N.d. Web dokumentaatio.
https://FMOD.com/resources/documentation-studio?version=1.10&page=glossary.html

Raybould, D. Stevens, R. 2015. Game Audio Implementation. Massachusetts, USA : Focal Press.

Madhav, Sanjay, 2013. Game Programming Algorithms and Techniques: A Platform-Agnostic Approach. New Jersey, USA : Addison Wesley

Bay, J. N.d. How To Become A Video Game Audio Implementer. Luettu 9.3.2020.
https://www.gameindustrycareerguide.com/how-to-become-a-video-game-audio-implementer/

Brown, Y. N.d. About Audio Middleware. Luettu 9.3.2020.
http://www.yannisbrown.com/about-audio-middleware/

Jolly, K. 2012. SIMCITY THE MUSIC OF SIMCITY. Luettu 12.4.2020.
https://www.ea.com/en-gb/news/simcity-music

Cities: Skylines Wiki. N.d.
https://skylines.paradoxwikis.com/Downloadable_content#Rock_City_Radio

FMOD Game Components. N.d. Web dokumentaatio
https://www.FMOD.com/resources/documentation-unity?version=2.0&page=game-components.html