Leo Virolainen

Adaptiivisen pelimusiikin toteuttaminen Unity-pelimoottorissa

Metropolia Ammattikorkeakoulu

Muotoilija (AMK)

Muotoilun tutkinto-ohjelma

Opinnäytetyö

9.12.2022

Tiivistelmä

Tekijä(t):	Leo Virolainen
Otsikko:	Adaptiivisen pelimusiikin toteuttaminen Unity-pelimoottorissa
Sivumäärä:	35 sivua
Aika:	9.12.2022
Tutkinto:	Muotoilija (AMK)
Tutkinto-ohjelma:	Muotoilun tutkinto-ohjelma
Suuntautumisvaihtoehto:	XR Design
Ohjaaja(t):	Lehtori Markku Luotonen

Opinnäytetyössä käsitellään adaptiivisen eli mukautuvan pelimusiikin historiaa, toimintaperiaatteita, määritelmiä ja kehitysprosessia. Tavoitteena on tarjota laaja yleiskatsaus aiheesta kiinnostuneille ja punnita erilaisia adaptiivisen pelimusiikin toteutustapoja. Työssä esitellään määritelmiä aihetta ympäröivälle terminologialle ja avataan yleisimpiä adaptiivisen musiikin rakennusperiaatteita.

Työ sisältää myös projektiosuuden, jonka lopputuloksena kehitetään indie-pelikehittäjän käyttöön tarkoitettu pienehkö adaptiivinen musiikinhallintajärjestelmä. Järjestelmä sisältää tärkeimmät toiminnallisuudet vertikaalisen ja horisontaalisen adaptiivisuuden toteuttamista varten, mukaan lukien äänenvoimakkuuden häivytyksen ja musiikin tahdin seurannan.

Kehitystyön vaiheita tarkastellaan opinnäytetyössä tarkasti. Alustavan tutkimustyön kautta opitaan, että 1990-luvun peleissä adaptiivisuuden runkona toimii MIDI (Musical Instrument Digital Interface) ja uudemmissa julkaisuissa kolmannen osapuolen äänenhallintaohjelmistot, joiden hyödyllisyyttä myös arvioidaan. Työssä käydään läpi musiikkijärjestelmän toteutuskeinon valintaan johtaneita seikkoja, jonka jälkeen tutustutaan projektiosuuden ohjelmointityön haasteisiin ja ratkaisuihin, etenkin musiikin tahdin seurannan osalta. Ennen loppua käydään läpi adaptiivisen pelimusiikin säveltämiseen liittyviä yksityiskohtia, kuten johtoaihetta (engl. leitmotif) ja pelimusiikin eroja elokuvamusiikkiin. Kehitysvaiheen lopuksi katsotaan yksittäisten funktioiden tarkkuudella projektiosuutta varten kehitetyn järjestelmän integrointia Unity-projektiin, minkä ansiosta löydetään pari tarvittavaa toiminnallisuutta, joista osa vielä lisätään tuotteeseen.

Viimeiseksi arvioidaan tavoitteiden toteutumista, tutkimustyön löydöksiä ja kehitystyön tulosta sekä jatkokehityksen mahdollisuuksia. Havaitaan, että toimivat ja edulliset adaptiivisen musiikin ratkaisut eivät ole tarpeeksi helposti saatavilla indie-kehittäjille ja että projektiosuuden tulos saattaisi jatkokehitettynä muuttaa tilannetta.

Avainsanat: Unity, musiikki, peli, adaptiivinen, säveltäminen

Abstract

Author(s):	Leo Virolainen
Title:	Implementation of adaptive game music in the Unity engine
Number of Pages:	35 pages
Date:	9 December 2022
Degree:	Bachelor of Culture and Arts
Degree Programme:	Design
Specialisation option:	XR Design
Instructor(s):	Markku Luotonen, Senior Lecturer

The thesis examines the history, principles, definitions and development process of adaptive game music. The goal is to offer a broad overview of the subject for those interested and to weigh the different ways of implementing an adaptive game soundtrack. The thesis presents definitions for the terminology surrounding the subject and lays out some of the most usual ways of building an adaptive score.

The work also includes a project portion which concludes with the finishing of a compact adaptive music framework intended for use by indie developers. The framework contains the most important functions for implementing vertical and horizontal adaptivity, including volume fading and the tracking of the tempo and beat of a given song.

The different stages of the music framework's development are scrutinized with great diligence in the thesis. The preliminary research finds that the base of adaptive music technology in the 1990s is MIDI (Musical Instrument Digital Interface) whereas in newer titles it is based more on external, third-party audio control software. The usefulness of said software is also assessed. The thesis considers the factors that led to the choice of means of implementation of the music framework, after which the challenges of the development process are examined in detail, regarding mostly the tracking of a song's progress. The details pertaining to the composition of adaptive music, such as leitmotifs and the differences between game and film music are reviewed and by the end of development, the integration of the developed framework into a Unity project is inspected, while paying close attention to select functions of the associated code. As a result, a need for a few additional features arises with some of the features being implemented into the final product.

Lastly, the completion of objectives, the findings of the research and development work and the possibility for further development are all assessed. As a result, it is found that functional and cost-effective solutions for adaptive music are not available enough but that the product of the project portion could present a change into the situation, if developed further.

Keywords: Unity, music, game, adaptive, composing

Sisällys

1 Johdanto

Miten sävelletään kappale, joka muuttuu joka kuuntelukerralla? Pelimusiikkia tehtäessä on aina hyödynnetty pelin interaktiivisuutta immersiivisemmän äänikokonaisuuden saavuttamiseksi. Nykyään lähestulkoon jokaisen pelin musiikki on jonkinasteisessa vuorovaikutuksessa pelin tapahtumien kanssa joko kappaleenvaihdoksen tai jonkin äänenpiirteen muokkauksen kautta. Pelimusiikin interaktiivisuuden kautta saadaan annettua pelaajalle lisäpalautetta, mutta miten saadaan maksimoitua annetun palautteen arvo?

Adaptiivinen pelimusiikki on sävelletty ja tuotettu alusta asti sillä tavoitteella, että musiikin eri osia voitaisiin kuunnella eri järjestyksissä niin, että kappale on silti ehjä kokonaisuus, jossa on alku, keskikohta ja loppu. Tämän lisäksi adaptiivista pelimusiikkia hallinnoiva koodi varmistaa, että kappaleen osat soivat aina sellaisessa sovituksessa, joka sopii pelaajan ennalta tuntemattomiin tekoihin mahdollisimman hyvin. Adaptiivinen musiikki voi lennosta muuttaa rytmiä, äänenväriä, soitinkokoonpanoa, harmoniaa ja melodiaa merkittävästi erilaisen lopputuloksen aikaansaamiseksi. Parhaimmillaan pelaaja ei edes huomaa pelimusiikin adaptiivisuutta.

Adaptiivinen musiikki jakaantuu kuitenkin moneen eri alalajiin toteutus- ja toimintaperiaatteensa mukaan. Tässä opinnäytetyössä tutkitaan ja kokeillaan, mikä tai mitkä toteutustavat ovat parhaita, tehokkaimpia ja tuottavat onnistuneimmat tulokset. Alussa käydään läpi adaptiivisen musiikin toteutuksia peliteollisuuden historiasta sekä niiden vaikutuksia nykypäivän käytäntöihin. Interaktiiviseen pelimusiikkiin kuuluvalle terminologialle etsitään määritelmät. Tämän jälkeen opinnäytetyössä tarkastellaan adaptiivisen pelimusiikinhallintajärjestelmän kehitystyön eri vaiheita alkaen alustavasta tutkimustyöstä ja prototypointi. Seuraavaksi perehdytään käytännön osuuden ohjelmointivaiheeseen sekä musiikin tuottamiseen adaptiivista järjestelmää varten. Lopuksi käydään läpi kehityksen haasteet, jatkokehityksen tavoitteet ja mahdollisuudet sekä tutkimustyön aikana tehdyt havainnot ja katsotaan, kuinka lähelle alussa linjattuja tavoitteita päästiin.

Opinnäytetyön käytännön osuuden tavoitteena on luoda kevyt, helppokäyttöinen ja tehokas järjestelmä, joka sallisi indie-kehittäjänkin resursseilla luotavan adaptiivista musiikkia Unity-projekteihin.

2 Opinnäytetyön tavoitteet

2.1 Kirjallisen osuuden tavoitteet

Opinnäytetyön kirjallisen osuuden osalta tarkoituksena oli tutkia adaptiivisen musiikin toteutuskeinoja sekä ulkopuolisten lähteiden että itse tehdyn käytännön toteutuksen kautta. Tavoitteena oli selvittää, ovatko vertikaalinen ja horisontaalinen adaptiivisuus yleisellä tasolla parempia adaptiivisen musiikin toteutuskeinoja kuin generatiivinen musiikki tai MIDI-soittimien (Musical Instrument Digital Interface) hyödyntäminen. Tärkeimpänä tutkimuskeinona käytettiin adaptiivisen musiikinhallintajärjestelmän kehittämistä; ajatuksena oli, että uutta järjestelmää kehitettäessä kohdattaisiin todennäköisimmin jokainen ongelma ja yksityiskohta, jota voisi kuvitella adaptiivisen musiikin toteuttamiseen liittyvän. Tällöin saataisiin väkisinkin tietoa myös joistakin ratkaisuista ja keinoista, joiden avulla kehitystyö suoriutuisi tehokkaimmin.

Ensisijaisen tutkimuskysymyksen ohessa tavoitteena oli myös tarkastella pienempiä, ikään kuin sekundäärisiä kysymyksiä. Jotta reaaliajassa mukautuva musiikki sopisi saumattomasti pelin tapahtumiin ja säilyttäisi samalla johdonmukaisen rakenteen, mitä vertikaalisia ja horisontaalisia adaptiivisen musiikin ominaisuuksia ja toiminnallisuuksia vaaditaan? Miten auditiivisia vihjeitä käytetään osana pelimusiikkia, jotta saadaan annettua lisäpalautetta pelaajalle? Mitä ovat adaptiivisen pelimusiikin rakentamisen design- ja tekniset haasteet? Miten ne ratkaistaan?

Tavoitteena oli myös tutustua tarkemmin adaptiivisen musiikin historiaan peleissä (ks. kuva 1). Mitä olivat aikaisimmat adaptiivisen musiikin ratkaisut ja niiden hyöty- ja haittapuolet? Kuinka päästiin nykyään yleisimmin käytettyihin ja

toimivaksi todettuihin toteutus- ja toimintatapoihin? Mikä tekee musiikin adaptiivisuudesta tekemisen arvoista, ja onko se hyödyllistä alun alkujaankaan?

Kuva 1. *Ultima Underworld: The Stygian Abyss* (Origin Systems 1992) oli yksi aikaisimpia adaptiivista musiikkia sisältäneitä pelejä (Ricketts 2022).

Viimeisenä tavoitteena oli vähintäänkin karkeasti määritellä adaptiivinen musiikki ja sitä ympäröivä terminologia. Jo nopealla katsauksella aihetta ympäröivään kirjallisuuteen havaittiin, että aiheeseen liittyvien termien käyttö eri osapuolien toimesta on epäjohdonmukaista: samaan asiaan saatetaan viitata monella eri sanalla. Esimerkiksi algoritmien avulla generoidusta musiikista puhuessaan Ó Nuanain (2019) puhuu proseduraalisesta musiikista (engl. procedural music), kun Kähkönen (2021) puhuu samaan aiheeseen viitatessaan algoritmisesta musiikista (engl. algorithmic music). Tätä kirjoitusta tehtäessä terminologian pitäminen johdonmukaisena ja selkeänä oli etusijalla, mikä vaati termien määrittelyä.

2.2 Käytännön osuuden tavoitteet

Opinnäytetyön käytännön osuutena toimi adaptiivisen musiikinhallintajärjestelmän kehittäminen Unity-pelimoottorille. Sille annettiin nimeksi “Framework for

Adaptive Modular Music” eli lyhennettynä FAMM. Ennen kehitystyön aloittamista laadittiin lista tavoitteista, joihin FAMMia kehitettäessä tulisi päästä, jotta voitaisiin laskea käytännön osuuden onnistuneen. FAMMin tuottaman musiikin oli ensisijaisesti pysyttävä johdonmukaisena: musiikissa olisi aina oltava selkeä alku, keskikohta ja loppu. Musiikki ei saisi alkaa tai loppua rujosti kuin seinään, ja uuteen säkeistöön olisi siirryttävä aina sulavasti. Tämä mielessä pitäen musiikin tapahtumien olisi silti pysyttävä relevantteina pelin tapahtumiin nähden; esimerkiksi suuri crescendo (äänenvoimakkuuden nouseminen) ei saisi ilmetä musiikissa silloin, kun pelaaja seisoo paikallaan toimettomana, vaan esimerkiksi pelaajan selättäessä jonkin esteen tai saavuttaessa jotakin hienoa. Tällä lailla FAMMin tuottama musiikki voisi toimia myös lisäpalautteen antajana pelaajalle, mikä oli myös yksi tavoitteista. Pelaajan päätökset, etenemiset, takaiskut ja läpimurrot olisivat kaikki asioita, joiden tulisi heijastua musiikissa, ja täten musiikki olisi integraalinen osa pelikokemusta. Jos musiikki ei söisi pelin muista ominaisuuksista ja olisi silti välttämätön osa immersion lisäämistä, palautteen antamista ja pelaajan ohjaamista oikeaan suuntaan, FAMM olisi ollut onnistunut ja vaivansa veroinen hanke.

Ensimmäinen askel onnistuneen lopputuloksen saavuttamiseen oli muiden pelien tekemien ratkaisujen ja oivallusten hyödyntäminen FAMMin kehityksessä. Adaptiivista musiikkia on harrastettu pelikehityksessä jo pitkään, ja vanhoistakin peleistä saattaisi löytyä edelleen käyttökelpoisia sekä tehokkaita keinoja adaptiivisiin ratkaisuihin (Ó Nuanáin 2019). Oli vain varottava, ettei lainattujen ratkaisujen implementointi tekisi FAMMista liian monimutkaista tai sekavaa. Koko järjestelmän oli muiltakin osin oltava mahdollisimman selkeästi rakennettu sellaisellekin käyttäjälle, joka ei olisi itse ollut mukana kehitystyössä. Tämä edellytti muun muassa tunnollista koodin kommentointia ja loogisia tiedostojen nimeämiskäytäntöjä. Omien audiotiedostojen lisääminen FAMMin käyttöä varten oli tehtävä intuitiiviseksi, ja muuttujien sijaintien Unityn inspectorissa oli noudatettava yhtenäistä logiikkaa sekä oltava hyvin näkyvissä. Lisäksi testausta varten tehdyssä peliprojektissa haluttiin hyödyntää FAMMia myös valikoissa; Callighanin (2022) mukaan liian harva peli käyttää adaptiivisia äänielementtejä jo

valikoissa, vaikka sillä voikin olla huomattava, jännitystä kohottava vaikutus pelikokemukseen (ks. kuva 2). Yhtenä sivutavoitteena olikin kokeilla läheisempää musiikin adaptiivisuuden hyödyntämistä valikoissakin.

Kuva 2. *Mario Kart Wiin* (Nintendo 2008) alkuvalikon musiikki muuttuu jännittävämmäksi pelaajan tehdessä valintoja ja varsinaisen pelin alun lähestyessä (Callighan 2022).

Ihanteellisessa tapauksessa FAMMin toiminta olisi erittäin pitkälle automaattiseksi ohjelmoitua niin, että yksittäisillä float-muuttujilla voitaisiin suoraan ohjata musiikin luonnetta. Esimerkiksi pelaajan kukistettua viisi vihollista muuttujan “jännite” numeroarvo nousisi ja täten musiikin sisältämä jännite vastaisi muutokseen kasvamalla myös, esimerkiksi uuden perkussiostemman liittymisellä soittoon. Stemmalla tarkoitetaan yhden soittimen tai soitinryhmän esittämää osuutta eli ikään kuin musiikillista kerrosta. (Ó Nuanáin 2019.) Huomionarvoisemmat tapahtumat, kuten tietyn alueen läpäiseminen, suuren vihollisryhmän tuhoaminen tai hankalan esteen ylittäminen, voisivat heijastua musiikissa omina tunnistettavina musiikillisina ideoinaan.

Tavoitteena oli hyödyntää valmiiksi tuotettuja, keskenään yhteensopivia ja päällekkäin ladottavia stemmoja, jotka aktivoitaisiin tai deaktivoitaisiin tarpeen tullen sekä reaaliaikaisesti manipuloitavaa MIDI-soitinta, ensisijaisesti melodioiden soittamista varten. Suurin hyöty tässä olisi ollut melodisten elementtien hyödyntämisen mahdollisuus musiikissa, etenkin niin, että eri melodioiden välillä olisi

voitu vaihdella sulavasti ja huomaamattomasti pienillä transitiomelodioilla ilman tarvetta äänenvoimakkuuden häivytyksille.

3 Adaptiivinen musiikki

Audio-ohjelmoija Andrew Clark (2007) ehdottaa kirjoituksessaan *Defining Adaptive Music* tarkkaa määritelmää adaptiiviselle musiikille sekä virkkeen että luetelman muodossa. Clarkin luetelmamuotoisen määritelmän mukaan adaptiivisessa musiikissa

- on järjestelmä, jolla generoidaan performanssiltaan merkittävästi toisistaan eriäviä versioita kappaleesta.
- Generaatiojärjestelmää ohjaavat ennalta määrätyt inputparametrit.
- Performanssin varsinaisilta tapahtumilta edellytetään merkittävän suurta ennalta määräämättömyyttä.
- Perinteinen musikaalinen johdonmukaisuus ja organisaatio on etusijalla.

(Clark 2007.)

Clarkin itsensäkin mielestä määritelmä on melko ikävystyttävä, pedanttinen ja akateeminen. Selkeyden vuoksi sanottakoon, että tässä kirjoituksessa käytetään sanaa adaptiivinen sellaisesta musiikista, joka on vuorovaikutuksessa pelin tapahtumien kanssa niin, että musiikki itse mukautuu peliin sen tapahtumien perusteella. Muutoksia voi ilmetä musiikin rytmissä, äänenvoimakkuudessa ja soinnillisessa rakenteessa. Adaptiivinen musiikki voi ohjata pelaajaa oikeaan suuntaan ja auttaa tätä ymmärtämään, kuinka hyvin pelaaminen sujuu tai esimerkiksi, kuinka lähellä tämä on voittamista. Ennen kaikkea musiikin adaptiivisuus kohentaa pelaajan immersiota. (Clark 2007.)

3.1 Interaktiivinen musiikki ja terminologia

Koska interaktiivisen pelimusiikin terminologia ei ole täysin juurtunut, samasta asiasta puhuttaessa saatetaan käyttää eri sanoja. Tätä kirjoitusta varten valittiin termit adaptiivinen, reaktiivinen, interaktiivinen ja dynaaminen kuvaamaan opinnäytetyön keskeisimpiä ideoita.

Pelimusiikin osalta termi ”interaktiivinen” toimii tässä kirjoituksessa eräänlaisena kattoterminä ja jakautuu neljään asteeseen: reaktiiviseen, dynaamiseen, adaptiiviseen ja täysin interaktiiviseen musiikkiin. Reaktiivisuudella viitataan tässä kirjoituksessa kaikista yksinkertaisimpaan musiikin interaktiivisuuteen, jossa kappaletta vaihdetaan yhdestä toiseen pelin tilanteen kehittyessä, esimerkiksi kun pelaaja siirtyy pelialueelta toiselle. Kappaleet soivat samanlaisena joka soittokerralla. Dynaamisella musiikilla tarkoitetaan sellaista interaktiivisuutta, jossa musiikin äänenvoimakkuutta säädetään muiden peliäänien ja efektien mukauttamiseksi, kuten musiikin äänenvoimakkuuden laskeminen ääniefektien soittamisen ajaksi. Adaptiivista musiikkia vielä lähemmin pelin tapahtumien kanssa sidoksissa on täysin interaktiivinen musiikki. Tällä viitataan sellaiseen musiikkiin, jossa pelaajan toiminta heijastuu suoraan pelimusiikkiin: tietty toiminto pelissä johtaa aina tietynlaiseen ääneen. Useat musiikkipelit, kuten esimerkiksi *Guitar Hero* (RedOctane 2005), sisältävät täysin interaktiivista musiikkia. Useimmiten tämä toimii niin, että kukin peliohjaimen näppäin vastaa tiettyä nuottia. (Kähkönen 2021.)

Kuva 3. Brittimuusikko Brian Eno oli mukana työstämässä evoluutiota simuloivan *Sporen* (Maxis 2008) generatiivista pelimusiikkia (Ó Nuanáin 2019). Kuvan lähde Kähkönen (2021).

Joissakin peleissä ei ole valmiita kappaleita käytettäväksi taustamusiikkina, vaan musiikki generoidaan, joskus yksittäisiä nuotteja myöten, pelattaessa muutamia ennalta määrättyjä lainalaisuuksia noudattaen. Tällaisesta musiikista

voidaan käyttää termiä generatiivinen, proseduraalinen tai algoritminen. Avaruusolionluontipeli *Spore* (Maxis 2008) (ks. kuva 3) käyttää laajasti muokattua versiota Pure Data -nimisestä musiikkiohjelmistosta ja luo sen avulla lennosta musikaalisia fraaseja, jotka reagoivat tiettyihin pelin tapahtumiin. Tietokoneen luomat fraasit liitetään osaksi musiikkia niin, että lopputuloksena on ehjä ja pelin tapahtumiin sopiva kokonaisuus. *Sporen* musiikki on myös dynaamista, eli sen soittimien tai stemmojen äänenvoimakkuus määräytyy pelissä kullakin hetkellä soivien ääniefektien ehdoilla. (Kähkönen 2021.)

3.2 Horisontaalinen ja vertikaalinen adaptiivisuus

Adaptiivinen musiikki jakautuu toimintalogiikkansa mukaan kahteen eri luokkaan: horisontaaliseen ja vertikaaliseen adaptiivisuuteen. Horisontaalisesta adaptiivisuudesta puhutaan silloin, kun musiikin tai muun peliäänen adaptiivisuus on rakennettu valmiiksi sävelletyistä ja tuotetuista kappaleista niin, että kappale vaihdetaan toiseen pelitilanteen niin vaatiessa. Konamin *Frogger*-peli vuodelta 1981 on aikaisimpia esimerkkejä horisontaalisesta adaptiivisuudesta. *Froggerin* musiikissa tapahtuu huomattava kappaleen vaihdos pelaajan päästessä turvalliseen kohtaan pelissä. (Kähkönen 2021.)

Kappaleen vaihdoksen pehmittämiseksi voidaan yksinkertaisimmillaan hyödyntää äänenvoimakkuuden häivyttämistä, mutta luonnollisempaa lopputulosta tavoiteltaessa voidaan odottaa kappaleen pääsemistä musikaalisen lauseensa loppuun. Jopa lyhyt transitiopätkä, esimerkiksi soolo tai rumpufilli, voidaan lisätä varsinaisten kappaleiden väliin. Jottei kokemus kuulostaisi joka kerralla samalta, voidaan kappaleista tehdä erilaisia variaatioita esimerkiksi melodiaa tai harmoniaa vaihtamalla. (Ó Nuanáin 2019.) Elokuvaohjaaja George Lucasin perustama Lucasfilm Games kehitti vuonna 1991 iMUSE-nimisen (Interactive Music Streaming Engine) työkalun seikkailupelejään varten, jonka tarkoituksena oli auttaa säveltäjää luomaan pelimusiikkia, joka olisi vuorovaikutuksessa pelin tapahtumien kanssa. (Kähkönen 2021.) iMUSE salli transitioiden lisäksi soitettavan pidennettyjä tai lyhennettyjä versioita samasta kappaleesta; jos pelaaja

käyttikin enemmän aikaa kaikkien dialogivaihtoehtojen tutkimiseen tai ohitti dialogin kokonaan, iMUSE pystyi ottamaan mahdollisen viiveen tai aikaistumisen huomioon soittamalla pidennetyn tai leikatun version kullakin hetkellä soivasta kappaleesta. (Silk 2010.) Eripituiset kappalevariaatiot korjasivat sattumalta myös arkisemman ongelman: pelaajan käyttämän koneen laskentatehosta riippuen grafiikkamoottorilla saattoi kestää eri määrä aikaa pelin tapahtumien toteuttamiseen ja piirtämiseen näytölle. iMUSE pystyi ottamaan mahdollisen viiveen huomioon ja soittamaan oikeanpituisen version kappaleesta sovittaakseen musiikin yhteen pelin tapahtumien kanssa. iMUSE kehitettiin *Monkey Island 2: LeChuck's Revenge* -seikkailupeliä (Lucasfilm Games 1991) (ks. kuva 4) varten, mutta sitä käytettiin runsaasti myös Lucasfilm Gamesin myöhemmissä julkaisuissa, kuten *Sam & Max Hit the Road* (1993) tai *Grim Fandango* (1998). (Kähkönen 2021.)

Kuva 4. Kuvakaappaus Peter Silkin (2010) Youtube-videosta. *Monkey Island 2: LeChuck's Revenge* oli ensimmäinen Lucasfilm Gamesin iMUSE-järjestelmää hyödyntäneistä peleistä (Kähkönen 2021).

Vertikaalinen adaptiivisuus käsittää useamman valmiiksi tuotetun ja keskenään yhteensopivan stemman keskenään yhdistelyä tai määrän lisäämistä tai vähentämistä pelitilanteesta riippuen. Tällä tavalla pelin tarinankulun muutosta korostetaan musiikin voimin. Esimerkiksi siirryttäessä huoneesta toiseen voidaan taustamusiikissa siirtyä toiseen säkeistöön tai lisätä uusi soitin kuvastamaan pe-

lin muuttunutta miljöötä tai tunnelmaa. Vertikaalisesti adaptiivinen musiikki soittaa ikään kuin samasta kappaleesta erilaisia sovituksia valiten aina vallitsevaan pelitilanteeseen parhaiten sopivan yhdistelmän soittimia, melodioita, harmonioita jne. Esimerkiksi tieteisstrategia- ja simulaatiopeli *FTL: Faster Than Light* (Subset Games 2014) (ks. kuva 5) sisältää jokaisesta kappaleestaan erikseen "exploration"- ja "battle"-versiot: kun pelaaja joutuu taisteluun toista avaruusalusta vastaan, rauhallisempi musiikki vaihtuu äänenvoimakkuuden häivytyksen kautta jännittävämpään ja rytmisempään taisteluvariaatioonsa. Kummankin variaation harmonia on samanlainen, minkä seurauksena siirtymä kuulostaa erittäin luonnolliselta. (Kähkönen 2021.)

Kuva 5. *Faster Than Lightissa* pelaaja komentaa avaruusalusta ja pakoilee jahtaavia ilkeitä kapinallisia (Wilde 2013, 2).

Lännenelokuviin perustuvan toiminta- ja seikkailupeli *Red Dead Redemptionin* (Rockstar 2010) alkuperäinen soundtrack on täysin adaptiivinen ja hyödyntää vertikaalista adaptiivisuutta erittäin laajasti. Soundtrackilla viitataan tässä kirjoituksessa pelin tai elokuvan musiikkiin; yksittäinen soundtrack koostuu kaikesta pelissä tai elokuvassa käytetystä musiikista. *Red Dead Redemptionissa* liikutaan avoimessa maailmassa ja musiikkiin lisätään tai vähennetään jatkuvasti uusia kerroksia saumattomasti tilanteen perusteella; esimerkiksi ratsaille nous-

tessa kappaleeseen liittyy rytmikäs bassorumpu. *Red Dead Redemptionin* ratkaisussa on kuitenkin huomattava varjopuoli: koska musiikin on soitava tauotta avoimessa maailmassa, jokaisen stemman on oltava yhteensopiva jokaisen muun stemman kanssa. Tämän seurauksena koko pelin soundtrackin tempo on lukittu 130 iskuun sekunnissa ja ainoa käytetty sävellaji on a-molli. (Reddeadwikia 2011.)

4 Musiikkijärjestelmän kehitys

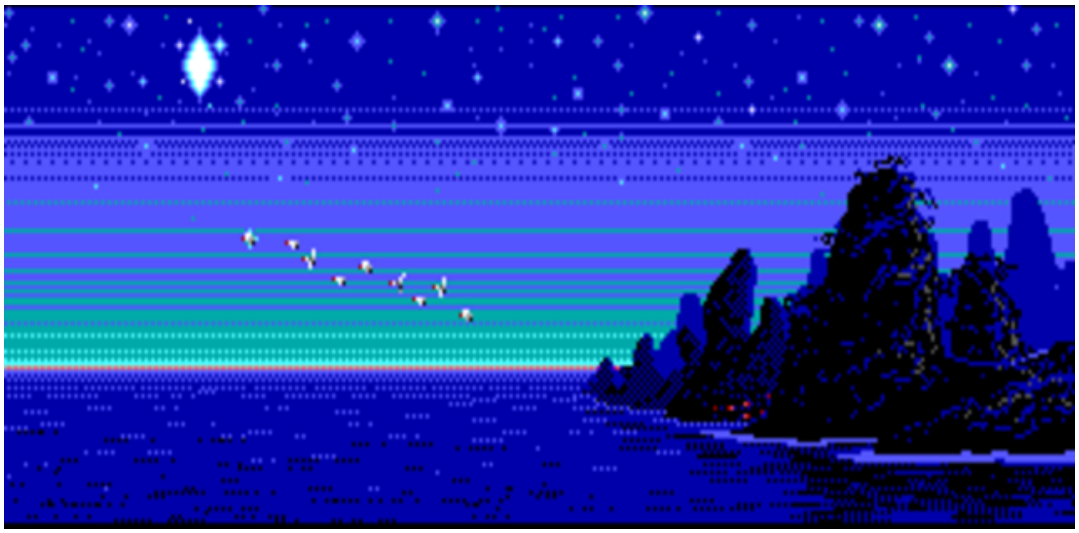
4.1 Alustava tutkimustyö

Ennen FAMMin kehitystyön aloittamista oli selvitettävä, mitkä olisivat parhaita tapoja toteuttaa adaptiivinen pelimusiikki Unity-moottorilla. Päädyttiin siihen tulokseen, että olisi päätettävä, hyödynnetäänkö valmiiksi tuotettujen äänitiedostojen kasaamista vertikaalisesti ja horisontaalisesti vai MIDI-tiedostojen lukemista reaaliajassa vai kenties molempia. Eri toteutustavoista tehtiin nopeat prototyypit Unity-moottorissa ja eri pelien musiikki- ja musiikinhallintajärjestelmiin perehdyttiin tarkemmin, jotta saataisiin lisätietoja kummankin lähestymistavan hyödyistä ja haitoista sekä siitä, miten FAMM olisi parasta toteuttaa.

4.1.1 Adaptiivisen musiikin toteutukset pelialalla

Varsinkin vanhemmissa 1990-luvun peleissä kuten *Loom* (Lucasfilm Games 1990) (ks. kuva 6) tai *Doom* (id Software 1993), on toteutettu musiikki MIDI-tiedostoja hyödyntäen yksinkertaisten, ei-adaptiivistenkin soundtrackien luonnissa. Koska MIDI-tiedostot eivät sisällä niin ikään valmista musiikkia vaan vain tarvittavan tiedon halutunlaisen musiikin tuottamiseen reaaliajassa, MIDI-tiedostot vievät hyvin vähän muistia. (Pidkameny 2002.) 1990-luvulla kuluttajien kotitietokoneiden äänentoistojärjestelmät eivät olleet standardisoituja; osa saattoi käyttää tietokoneensa sisäänrakennettua ääniteknologiaa (PC speaker), ja osalla saattoi olla teollisuuslaatuinen syntetisaattori, kuten Roland mt-32 yksinomaan peliäänien toistoa varten. Tämän vuoksi MIDI oli erittäin otollinen tapa toteuttaa

pelimusiikkia, sillä MIDI-tiedostoja saatettiin lukea minkä tahansa laatuisella äänentoistolaitteella. (Ahoy 2018.)

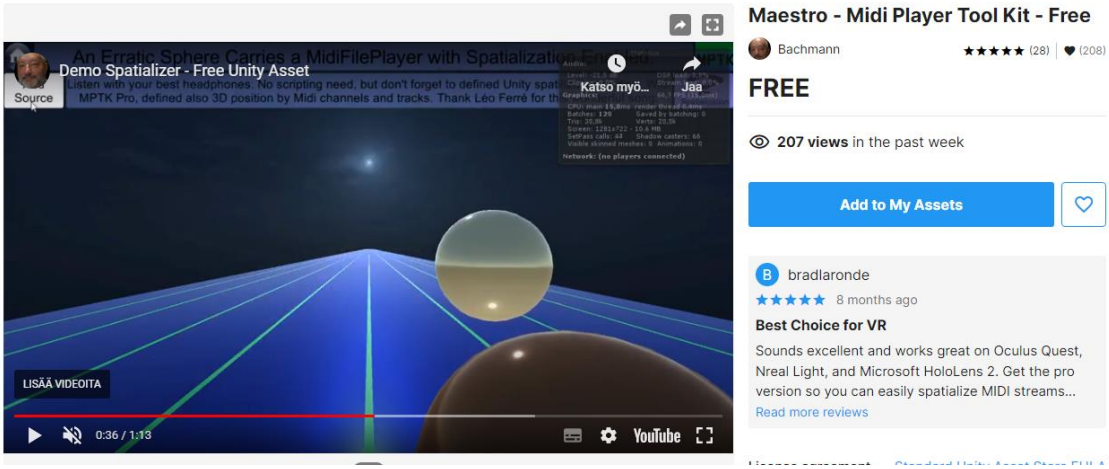


Kuva 6. *Loomin* musiikki on yksi tunnetuimpia aikaisia MIDI:llä toteutettuja pelimusiikkeja. MIDI:llä tehtiin peliin myös musikaalisia arvoituksia (Pidkameny 2022). Kuvan lähde Maher (2017).

Pelinkehittäjille on jo tarjolla audionhallintamoottoreita, kuten Wwise tai FMOD, jotka ovat usein keskeinen komponentti adaptiivisen musiikin toteutuksissa suurissakin peleissä (Ó Nuanáin 2019). FMODia käyttää esim. *World of Tanks* (Wargaming 2010) lukuisten muiden joukossa (Firelight Technologies 2022a), ja Wwisea on käyttänyt mm. *Cyberpunk 2077* (CD Projekt Red 2020) (Audiokinetic 2022). Wwise ja FMOD ovat kuitenkin kaupallisille tuotteille maksullisia, vaikkakin FMOD on ilmainen, jos pelin julkaisevan yhtiön vuositulot ovat alle 200 000 USD ja pelin kehitysbudjetti on alle 500 000 USD (Firelight Technologies 2022b). Siitä huolimatta haluttiin kehittää täysin itse tehty järjestelmä adaptiivisen musiikin hallinnointia varten, jotta saataisiin parempi ymmärrys kaikista adaptiivisiin musiikkijärjestelmiin liittyvistä teknisistä vaatimuksista. Näin saataisiin toivon mukaan paremmin tietotaitoa adaptiivisten musiikkijärjestelmien toiminnasta syvemmällä tasolla. Monelle pienemmälle kehittäjälle FMODin tai Wwisen tapaisen suuren erillisen ohjelmiston käyttäminen voisi tuntua eräänlaiselta ylilyönniltä; kävisi järkeen, ettei Unity-projektin kokoa haluttaisi paisuttaa kaikilla merkityksettömillä lisäominaisuuksilla, joita isokokoinen ulkopuolinen ohjelmisto toisi tullessaan. FAMM tarjoaisi kevyemmän ja helpommin saatavilla olevan vaihtoehdon suurille audionhallintaohjelmistoille.

4.1.2 Prototypointi

Alkuperäiseen visioon FAMMista kuului MIDIn ja modulaaristen musiikkistemmojen yhdistely mahdollisimman monipuolisen lopputuotteen saavuttamiseksi. Modulaarisella musiikilla viitataan tässä kirjoituksessa sellaiseen musiikkiin, joka on sävelletty ja tuotettu adaptiivisesta lähtökohdasta; musiikki on jaettu keskenään yhteensopiviin eri osiin eli moduuleihin, joita voidaan lennosta vaihdella keskenään variaation saavuttamiseksi. MIDI-tiedostot sisältävät informaatiota erillisistä nuoteista eivätkä niinkään lopullisesta äänestä. MIDI-tiedostossa voi olla tallennettuna mm. nuotin sävelkorkeus, ajoitus tai äänenvoimakkuus, mutta myös erilaisia jälkikäsittelyefektejä. Sama MIDI-tiedosto voidaan syöttää mille tahansa virtuaaliselle MIDI-soittimelle, mikä mahdollistaa soittimen vaihtamisen milloin tahansa. (Swift 2012.) MIDIn avulla voisi käyttää modulaarisia, valmiiksi tuotettuja stemmoja taustaäänissä kuten rummuissa, bassolinjassa jne. ja melodian soittamisessa niin, että melodia soitettaisiin aina reaaliajassa MIDI-soittimella, jolloin soitinta voitaisiin myös vaihtaa lennosta. Tällöin melodiastemmalle voisi suoraan kertoa, mitä ääniä ja äänisarjoja soittaa seuraavaksi, jolloin melodia voisi hyvin monipuolisesti mukautua pelin tapahtumiin. Esimerkiksi eri pelihahmoilla voisi olla omat lyhyet melodiat tai johtoaiheet, jotka voitaisiin soittaa aina kullakin hetkellä soivan pohjamelodian väliin sen ollessa aiheellista. MIDI olisi myös sallinut generatiivisen eli proseduraalisilla ohjelmanpätkillä generoidun musiikin tuottamista reaaliajassa, mikä olisi ollut erittäin mielenkiintoista, mutta myöskin varmasti aikaa vievää.



Kuva 7. Maestro - Midi Player Tool Kit on saatavilla ilmaiseksi Unityn asset storesta ja tarjoaa MIDI-tiedostojen lataus- ja soittotoiminnallisuuksia.

Pikainen kokeilu tehtiin myös siitä, mitä erilaisia tapoja MIDIn tuomiseen Unity-moottoriin löytyy. Muutamaa Unityn asset storesta löydettyä valmista MIDI-tiedoston lukijaa (ks. kuva 7) kokeiltua huomattiin kuitenkin, että halutun kaltaisen MIDI-järjestelmän kehittäminen veisi liian paljon aikaa. Täten jouduttiin luopumaan ajatuksesta ainakin siinä määrin, ettei FAMMin ensimmäiseen versioon kehitettäisi MIDI-ominaisuuksia. Päätettiin siis keskittyä entistä enemmän ennalta tuotettujen musiikinpätkien tarjoamiin mahdollisuuksiin eli vertikaaliseen ja horisontaaliseen adaptiivisuuteen.

4.1.3 Toteutuskeinon valinta

Prototypointi ja adaptiivisen musiikin luontiin kohdistuvan tutkimustyön lopuksi saatiin koottua vertikaalisen ja horisontaalisen adaptiivisuuden eroja sekä MIDIn ja generatiivisen musiikin ominaispiirteitä. Saatua tietoa käytettiin apuna lopullisen päätöksen teossa siitä, kuinka ruvettaisiin lähestymään FAMMin kehittämistä.

Vertikaalisen adaptiivisuuden eduista suurin lienee se, että pelaajan kuulema kokonaisuus on teoriassa joka kuuntelukerralla erilainen; stemmat voivat soida eri yhdistelmissä, alkaa eri aikoihin ja täten muodostaa yhdessä erilaisia kappaleita samoista musikaalisista komponenteista. Tämä lisää pelin uudelleenpelattavuutta ja vähentää pelin toistuvuutta esimerkiksi pelaajan juuttuessa ja pelatessa toistuvasti samaa kohtaa pelistä. Vertikaalisen adaptiivisuuden tuottama musiikki muuttuu myös vain vähän kerrallaan, jolloin kappaleen rakenne pysyy orgaanisena ja muutokset hienovaraisina. (Game Maker’s Toolkit 2014.) Näin ollen on pienempi mahdollisuus siihen, että pelaaja havahtuisi muutoksen tapahtuessa musiikissa, mikä rikkoisi pelaajan immersiota. Vertikaalisuus sallii kumminkin myös musikaalisten johtoaiheiden soittamisen horisontaalisesti muiden äänitiedostojen ohella, kun se on ajankohtaista. Tämä lisää huomattavasti musiikin tarinankerronnallista arvoa ja sitoo musiikin entistä paremmin pelin tapahtumiin sekä lisää pelaajalle annettua palautetta. (Noodle 2019.)

Horisontaalisen adaptiivisuuden huomattavin hyöty taas on, ettei jokaisen musiikin osan tarvitse sopia yhteen jokaisen muun musiikin osan kanssa; äänitiedostot ovat valmiita, yksin soivia kokonaisuuksia, mikä mahdollistaa suuremman vapauden musiikkia sävellettäessä. Tämä voi vuorostaan johtaa monipuolisempaan lopputulokseen ja yksinkertaisempaan ja sitä kautta joutuisampaan säveltämis- ja tuottamisprosessiin. (Noodle 2019.) Prototypoinnin kautta havaittiin myös, että horisontaalisten kappaleiden implementointi peliin on siltä kannalta helpompaa kuin vertikaalisten, ettei pelin tarvitse antaa musiikkia hallinnoiville järjestelmille muuta tietoa kuin aika, jolloin vaihtaa kappaletta. Vertikaalisen adaptiivisuuden vaatimat äänenvoimakkuuden häivytykset ja tilanteeseen sopivan stemmojen yhdistelmän ylläpito eivät ole aiheellisia huolia. Ainoa suurempi haaste horisontaalisen järjestelmän implementaatiossa on se, että musiikin kulkua täytyy seurata koodissa yksittäisen iskun tarkkuudella, jotta voidaan varmistaa tarkka ajoitus seuraavan äänitiedoston aloitukselle.

MIDIn käyttäminen adaptiivisen musiikin luomisessa on kiehtova konsepti, joka tarjoaa paljon mahdollisuuksia. Koska tiedosto sisältää informaatiota vain erillisistä nuoteista, lopputulos on erittäin pitkälle muokattavissa erilaisilla jälkikäsittelyefekteillä ja soitinta voidaan vaihtaa jopa lennosta. MIDI-tiedostot ovat myös huomattavan pienikokoisia, minkä vuoksi niitä onkin käytetty runsaasti varsinkin vanhemmissa peleissä, vaikkakin useimmin tavanomaisen, ei-adaptiivisen musiikin soittamiseen. (Swift 2012.) Jos MIDI-tiedostoa voitaisiin vielä muokata reaaliajassa, niin jokainen stemma voisi vaihtaa sävellajia ja tempoa, puhumattakaan erillisten musikaalisten ideoiden mahduttamisesta muun soiton väliin sen ollessa ajankohtaista. Jos esimerkiksi pelin vihollisilla olisi oma teemansa, se voitaisiin soittaa luonnollisena osana musiikkia aina pelaajan kohdatessa vihollisen. Tällaisen MIDI-järjestelmän implementointi olisi kuitenkin huomattava looginen haaste, minkä vuoksi FAMMia ei todennäköisesti olisi saatu valmiiksi ajoissa, jos tällaista toiminnallisuutta olisi ruvettu kehittämään. Aikeena on kuitenkin perehtyä ainakin osittaisen MIDI-järjestelmän lisäämiseen FAMMiin opinnäytetyön valmistuttua.

Generatiivinen eli proseduraalisilla algoritmeilla generoitu adaptiivinen musiikki on vähintäänkin ajatuksia herättävä kandidaatti toteutustavaksi, mutta vaaraksi jää riski liian satunnaiselta kuulostavasta musiikista, jossa ei olisi tarpeeksi toistuvuutta eikä tunnistettavia musikaalisia teemoja. Generatiivisen adaptiivisuuden tuottama musiikki voisi hyvin toteutettuna heijastaa pelin tapahtumia ja tunnelmaa erinomaisellakin menestyksellä, mutta huonossa lopputuloksessa olisi vaarana, ettei kappaleessa olisi mitään muistettavuutta, mikä haittaisi vakavasti musiikin tarinankerronnallista arvoa ja vaikeuttaisi huomattavasti tunteiden herättämistä kuuntelijassa.

Näiden havaintojen perusteella päädyttiin siihen ratkaisuun, että lopullisessa tuotteessa yhdisteltäisiin vertikaalisia ja horisontaalisia adaptiivisuuden keinoja valmiiksi tuotettuja musiikinpätkiä hyödyntäen. Vertikaalisuudella saataisiin musiikkiin sulavuutta ja minimoitaisiin toistuvuutta. Horisontaalisuutta hyödyntäen voitaisiin soittaa melodioita, johtoaiheita ja muita, ikään kuin musikaalisia iskurepliikkejä, tarkoin ajoituksin ja tarkkaan harkituilla hetkillä. MIDI olisi mahdollinen lisäominaisuus jatkokehitystä katsoen ja generatiivinen musiikin tuottaminen jäisi kokonaan pois, vaikka olikin erittäin kiehtova konsepti.

4.2 Funktioiden ohjelmointi

Tutkimustyön lopuksi tiedettiin suurin piirtein, mitä toimintoja FAMMissa olisi oltava, jotta sitä voisi käyttää sellaisen vertikaalisen ja horisontaalisen adaptiivisuuden luontiin, jollaista oli kaavailtu. Vertikaalista toiminnallisuutta ajatellen tärkeimmäksi FAMMin ominaisuudeksi määriteltiin funktiot äänitiedostojen aktivoinnille ja deaktivoinnille äänenvoimakkuutta häivyttämällä. Muita ohjelmoinnillisesti raskaita toiminnallisuuksia vertikaalisuus ei vaatinut, vaan loput työstä koostui melko yksinkertaisesta klikkailusta Unityn inspector- ja hierarkiaikkunoiden puolella. Horisontaalinen adaptiivisuus vaati laajempaa ohjelmointityötä: äänitiedostojen soittamisen aloitus oli voitava tahdistaa yksittäiselle iskulle hyvin täsmällisesti. Tämä vaati iskujen, tahtien ja säkeistöjen tarkkaa seurantaa koodissa, mikä vuorostaan vaati tahdin laskemista täydellisellä tarkkuudella. Toisin sanoen koodin oli selvitettävä, kuinka paljon aikaa kuluu jokaisen iskun välissä,

ja hyödynnettävä tätä tietoa täsmälliseen iskujen, tahtien ja säkeistöjen laskemiseen. Tahdinseurannan sulava toiminta salli funktioiden ajamisen, ensisijaisesti äänitiedostojen soittamisen, minkä tahansa iskun, tahdin tai säkeistön alkaessa.

4.2.1 Äänenvoimakkuuden häivyttäminen

Sekä vertikaalisia että horisontaalisia ominaisuuksia varten tarvittiin ratkaisu äänenvoimakkuuden häivyttämiseksi, jotta stemmoja voitaisiin vaihdella sulavasti. Unityn audio mixer tarjosi tähän valmiin ratkaisun: minkä tahansa Unityn audio sourcen äänilähdön voi reitittää audio mixeriin, jonka kautta voidaan säätää audio sourcen soittaman äänen ominaisuuksia. Yksi mikseri voi hallinnoida monen audio sourcen tai myös toisen mikserin äänentoistoa. Audio mixer -näkymässä voi myös luoda snapshoteja eli erilaisia miksereiden, niiden ominaisuuksien ja efektien yhdistelmiä. Näiden välillä voidaan vaihdella käyttäen Unityn AudioMixerSnapshot.TransitionTo -funktiota, joka siirtyy yhdestä snapshotista toiseen nostamalla uuden ja laskemalla vanhan snapshotin äänenvoimakkuutta funktiolle annetun ajan funktiona. Tämä toteutus ei kuitenkaan ollut toimiva ratkaisu, sillä jotta tätä voitaisiin käyttää yksittäisiä musiikkistemmoja varten, olisi pitänyt luoda erillinen snapshot jokaisen eri stemman aktiivista sekä epäaktiivista tilaa varten jokaisessa eri yhdistelmässä. Tämä olisi ollut erittäin työläs ja tehoton ratkaisu. Snapshotit toimivat silti suurempien musiikillisten muutosten kanssa, joita oli valmiissa tuotteessa paljon vähemmän. Tällaisia ovat esimerkiksi duurin ja mollin välillä vaihtelu tai kappaleen vaihdos esimerkiksi loppuvihollista kohdattaessa.

```
//fade *out* an audio source (stem)
2 references
public IEnumerator FadeOut(AudioSource myAudioSource, float fadeTime) {
    currentTime = 0f;
    float maxVol = myAudioSource.GetComponent<FammAudioSource>().myMaxVol;
    float minVol = myAudioSource.GetComponent<FammAudioSource>().myMinVol;

    while (myAudioSource.volume > minVol) {
        currentTime += Time.deltaTime;
        myAudioSource.volume = Mathf.Lerp(maxVol, minVol, currentTime / fadeTime);
        yield return null;
    }
}
```

Kuva 8. FadeOut-korutiinin koodi FammMaster-scriptissä. FadeIn-korutiini toimii täysin samalla periaattella, mutta käänteisillä arvoilla.

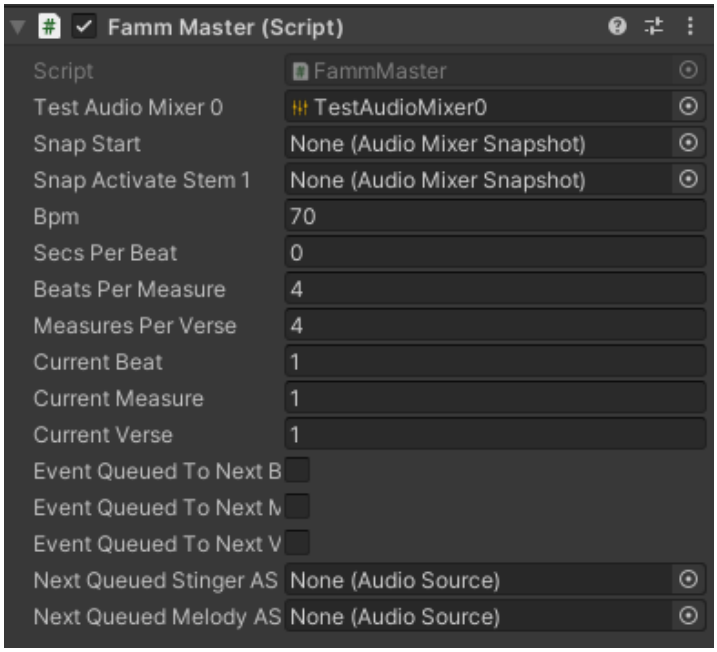
Äänenvoimakkuuden häivytykselle kehitettiin omat korutiinit, FadeIn ja FadeOut (ks. kuva 8), jotka säätävät suoraan yksittäisen audiosourcen äänenvoimakkuutta. Äänenvoimakkuutta nostetaan tai lasketaan annetun ajan funktiona, kunnes audiosourcen volume-ominaisuus on joko audiosourcelle erillisen FammAudioSource-koodin kautta annetun myMaxVol- tai myMinVol-floatin arvo. FadeIn- ja FadeOut-korutiinit voidaan kutsua mistä vain peliobjektista, ja ensisijainen käyttökohde on vertikaalinen adaptiivisuus eli musikaalisten elementtien lisääminen tai poistaminen soivasta kappaleesta.

4.2.2 Tahdin seuranta koodissa

Jotta musiikki voisi mukautua pelin kulkuun reaaliajassa, oli musiikin tahtia pystyttävä seuraamaan. Tällöin musiikin vaihdokset saatiin ajoitettua niin, ettei tahtilaji rikkoutuisi. Jos esimerkiksi uuden musiikkistemman soittaminen aloitettaisiin milloin vain, muodostuisi ongelmaksi pahimmassa tapauksessa se, että kaikki musiikin eri soittimet soittaisivat eri kohtaa kappaleesta, jolloin musiikki menettäisi kaiken merkityksen ja laadun.

Aivan ensiksi oli tiedettävä kappaleen tempo. Musiikin tempoa mitataan iskuilla minuuttia kohden (engl. beats per minute) eli bpm. Toistaiseksi FAMM ei tue tempon vaihtamista kappaleen aikana, joten kappaleet pitää tuottaa niin, että niissä pysyy sama tempo koko ajan. FAMMissa kunkin kappaleen tempo voidaan asettaa manuaalisesti Unityn inspectorissa (ks. kuva 9). Jotta funktioiden ajaminen saadaan tahdistettua musiikin iskuille, jaetaan kappaleen tempo 60:lla, jolloin saadaan musiikin iskujen välinen aikaintervalli sekunneissa. Laskun tuloksena saatava numeroarvo asetetaan secsPerBeat –float-muuttujan arvoksi. FAMMiin rakennettiin ajastin, joka laskee koko ajan nollasta secsPerBeatin arvoa kohti ja saavuttaessaan sen nollautuu taas. Laskimen osuessa nollaan ajetaan funktio ReachedEndOfBeat, jolloin tehdään iskulle tahdistettavat asiat. FAMMille syötetään myös tieto siitä, montako iskua jokainen tahti (engl. measure) sisältää, eli kappaleen tahtilaji. Yleisin tahtilaji musiikissa on 4/4 eli neljä neljäsosanuottia per tahti, siis neljä iskua per tahti. Tahtilaji tietäen voi-

daan yhteenlaskulla pitää lukua siitä, monennenko säkeistön (engl. verse) tahdin isku kulloinkin on meneillään. Säkeistö vaihtuu FAMMissa oletusarvoisesti kerran jokaista 16 iskua eli neljää tahtia kohden. Tätä kautta myös uuden tahdin tai säkeistön alkaessa voidaan toteuttaa toiminnallisuuksia ReachedEndOfMeasure- ja ReachedEndOfVerse-funktioiden avulla.



Kuva 9. FammMaster-scriptin julkiset muuttujat inspector-näkymässä antavat käyttäjälle lisätietoa ja mahdollistavat tempon ja tahtilajin manuaalisen muuttamisen.

Unityn Time.deltaTime -ominaisuuden kautta laskettu aika on liian epätarkka, ja pitemmän päälle pienet ajoitusvirheet kasaantuvat, mikä johtaa siihen, että uusi ääni on jo alkaessaan väärässä tahdissa. Laskemisen tarkkuuden parantamiseksi nextBeat –float-muuttujan, jota käytetään yksittäisten iskujen laskemiseen, nollautuessa lisätään laskuriin aikalaskennan epätarkkuuden aiheuttama yli- tai alijäämäaika, jolloin laskuri ei jätätä todellisesta kappaleen tahdista. Vaikka virheet eivät enää kasaudu, yksittäisten iskujen ajoitukset ovat silti ajoittain vähän myöhässä tai ajoissa sen verran, että virheen huomaa musiikkia kuunneltaessa. Ratkaisu tähän löytyi Unityn Audiosettings.dspTime ominaisuudesta, joka on sekunneissa mitattava arvo, joka perustuu itse Unityn audiojärjestelmän käsittelemien samplejen määrään ja on siksi paljon tarkempi kuin

Time.time -ominaisuuden kautta saatava aika. Näin ollen laskurin yli- tai alijäämäaikaa ei tarvitse enää laskea, sillä laskuri on niin tarkka, ettei virheitä ilmene enää. Jokainen iskulle tahdistettava tapahtuma tapahtuu täsmällisesti eikä liian aikaisin tai myöhässä.

4.3 Peliprojekti

FAMM haluttiin kehittää jonkin olemassa olevan peliprojektin ympärille pelin itsensä ehdoilla, jotta voitaisiin konkreettisesti testata FAMMin integrointia Unity-projektiin. Prototypointia varten valittiin peli, jota oli jo työstetty muutamat kuukaudet ennen opinnäytetyön aloittamista. Koska projektilla ei ollut vielä nimeä, peliprojektiin viitataan jatkossa yksinkertaisesti tankkipelinä.

Tankkipeli (ks. kuva 10) on kolmannen persoonan toiminta- ja seikkailupeli, jossa taistellaan panssarivaunulla fiktiivisessä tulevaisuudessa avaruusolioivalloittajia vastaan. Pelaaja ohjaa tankkia näppäimistöllä ja hiirellä. Pelin visuaalinen ilme saa inspiraatiota vanhoista Playstation 1 -konsolille tehdyistä peleistä, ja pelin tunnelma on humoristinen ja korni. Pelaajan tavoitteena on päästä jokaisen pelin taso alusta loppuun omia joukkoja suojellen ja tuhoten tai väistäen tarvittaessa vihollisia. Tankki saa visuaalisia päivityksiä ja uusia ominaisuuksia pelin edetessä, mikä tuo uusia pelimekaniikkoja. Näin pyritään säilyttämään pelaajan mielenkiinto peliin.

Kuva 10. Tankkipelissä pelaaja ohjaa tulivoimaista panssarivaunua. Peli koostuu noin 10–20 minuutin pituisista tasoista, joista jokaisen taustamusiikkina toimii yhdelle tasolle pyhitetty adaptiivinen kappale.

Tankkipeli oli erittäin otollinen kandidaatti adaptiivista pelimusiikkia varten, sillä peli tarjosi paljon mahdollisuuksia pelaajalle lisäpalautteen antamiseen musiikin kautta. Lisäksi adaptiivinen musiikki voisi suuresti lisätä pelaajan immersiota pelimaailmaan ja saada pelaajan välittämään jokaisen tason tarinan kulusta enemmän. Toisin kuin *Red Dead Redemption* (Rockstar 2010), jonka musiikki soi jatkuvassa avoimessa maailmassa (reddeadwikia 2011), tankkipelin musiikin tempon ja sävellajin rajoitukset koskevat vain yksittäisiä tasoja. Pelin tasoilla voisi vallan hyvin olla omat temponsa sekä sävellajinsa, vaikka käytettäisiinkin samantyylistä ratkaisua. Näin ollen musiikin toistuvuus ei olisi ollenkaan niin suuri huoli kuin jos adaptiivista musiikkia tehtäisiin avoimen maailman peliin.

4.4 Musiikin tuottaminen

Pelejä varten sävellettäessä on otettava huomioon se, että toisin kuin elokuvien kohdalla, pelin tapahtumat eivät ole ennalta tiedossa. Elokuvasäveltäjä voi katsoa elokuvan tai lukea käsikirjoituksen ja näiden pohjalta luoda täysin elokuvaan temaattisesti sopivan ja täydellisesti ajoitetun soundtrackin, mutta pelejä varten musiikkia tehdessä on otettava huomioon pelin interaktiivisuus. Siksi pelimusiikissa on jo vähintään viimeiset 40 vuotta kehitetty erilaisia interaktiivisen

musiikin keinoja. Näillä on pyritty parhaan mukaan ennakoimaan pelaajan tekoja ja päätöksiä, jotta musiikki sopisi yhteen pelin tapahtumien kanssa. (Ó Nuanáin 2019.)

4.4.1 Johtoaihe

“Leitmotif” eli suomeksi johtoaihe on yksi elokuva- ja pelimusiikin ominaisimpia tehokeinoja. Oopperoistaan tunnetun saksalaisen säveltäjän Richard Wagnerin suosituksi tekemällä johtoaiheella tarkoitetaan musiikillista ideaa, jonka musiikin kuuntelija voi rinnastaa hahmoon, esineeseen, paikkaan tai ideaan. Johtoaiheen ei tarvitse olla aina melodia, vaan se voi olla mikä tahansa rytmi, ääni, soitin tai sointu. (Sideways 2016.) Esimerkiksi *Taru Sormusten Herrasta* –elokuvatrilogian (New Line Cinema 2001) musiikissa käytetään johtoaihetta ahkerasti: Tarinan keskipisteellä, sormuksella, on oma melodiansa, joka soi osana taustamusiikkia aina sormuksen esiintyessä. Samaten monella tarinan keskeisimmistä hahmoista ja paikoista on omat musikaaliset tunnusmerkkinsä, jotka ilmenevät johtoaiheena. (Keane 2021.)

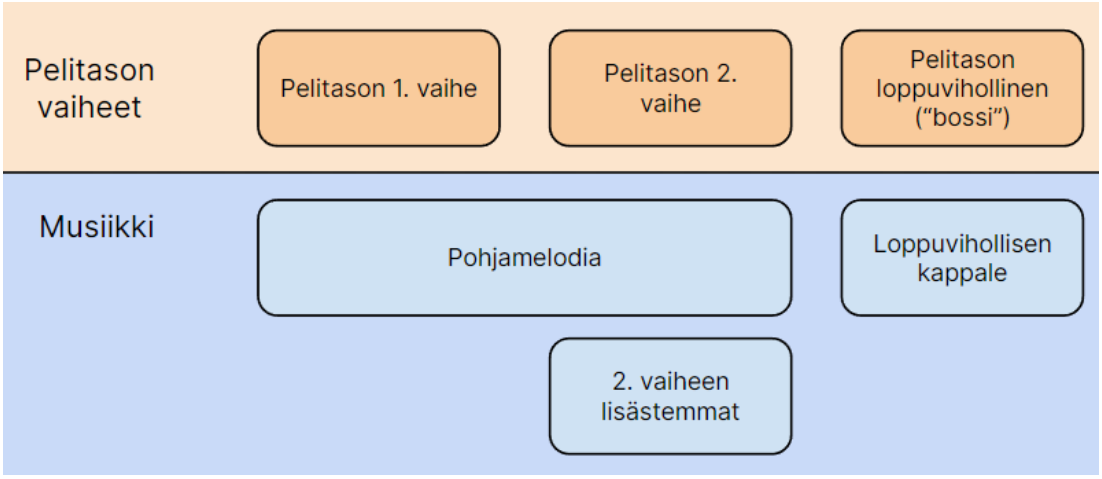
Pelimusiikissa johtoaihetta on nähty enemmän vahvasti tarinaan keskittyvissä peleissä, kuten *Halo: Combat Evolvedissa* (Bungie 2001) ja muissa pelisarjan jatko-osissa, jotka sisältävät paljon adaptiivisen musiikin malliesimerkkejä (Kähkönen 2021). Peleissä ylipäänsä johtoaiheet ovat yleisesti esiintyneet osana suurempia ja laveammin käytettyjä, esimerkiksi koko teokselle tai tarinan päähahmolle kuuluvia musikaalisia pääteemoja, mutta peleissä on harvemmin sidottu johtoaiheita myös yksittäisempiin asioihin esimerkiksi luomalla jokaiselle vihollistyypille oma johtoaiheensa (Cedeno 2018). Tutkimustyön aikana ei kuitenkaan löytynyt esimerkkiä pelistä, joka ottaisi tekniikasta kaiken hyödyn irti soittamalla johtoaihetta adaptiivisesti yhtenä osana jatkuvaa taustamusiikkia esim. aina johtoaihetta vastaavan vihollisen esiintyessä. Tämä on yksi adaptiivisen musiikin keino, jota olisi tarkoitus kokeilla laajasti FAMMin avulla. Kokeilun hypoteesina on, että runsaalla johtoaiheen adaptiivisella käytöllä voisi sekä nostaa pelaajan immersiota että parantaa tämän käyttökokemusta huomattavasti.

4.4.2 Adaptiivisen musiikin säveltäminen

Tankkipelin musiikin sävellystyö kirkasti eroja adaptiivisen ja tavanomaisen pelimusiikin tuottamisen välillä. Suurin ero sävellysprosessiin tulee siitä, että kappaleen jokaisen stemman on useimmiten sovittava yhteen vähintään pohjamelodian kanssa, mieluiten kaikkien stemmojen kanssa. Yleensä yhteensopivuuden edellytyksenä on, että samaan aikaan soivilla stemmoilla on sama tempo ja sävel- sekä tahtilaji. Kesken kappaleen vaihtuva tempo on hyvinkin saavutettavissa oleva ominaisuus, mutta aikarajoitusten vuoksi se jäänee jatkokehityksen puolelle. Jos sävelletään stemmoja, jotka sopivat yhteen vain tiettyjen muiden stemmojen kanssa, on varmistettava, etteivät yhteensopimattomat stemmat voi soida pelissä samanaikaisesti. Yksi tapa varmistaa yhteensopivuus on säveltää koko kappale vain yhtä sävellajia hyödyntäen, mikä voi aiheuttaa liikaa toistuvuutta varsinkin pitempien kappaleiden kohdalla. Toinen keino on säveltää musiikki niin, että sävellajin muutoksia ilmenee, mutta tällöin on jokaiselle musiikkiin lisättävälle stemmalle määrättävä ennalta kohta tai kohdat kappaleessa, johon stemma sopii aiheuttamatta riitasointua. Tämä on melko helppoa, jos kaikki stemmat alkavat samalla hetkellä ja soivat koko ajan taustalla mykistettynä: päälle kytkettäessä mykistys vain poistetaan, ja ajoitus on automaattisesti täydellinen. Lyhyempien, horisontaalisesti musiikin päälle ladottavien stemmojen kuten melodioiden käyttöä varten voidaan tehdä monta eri versiota kustakin stemmasta: yksi jokaista mahdollista stemman aloituskohtaa varten, jolloin stemman harmonia eli soinnut sopivat kappaleen sävellajiin. Tämä on kuitenkin hyvin työläs prosessi. Toinen vaihtoehto on odottaa, että kappale pääsee sellaiseen kohtaan, jossa kulloinkin kyseessä oleva stemma sopii muuhun kappaleeseen. Tämä ei ole kuitenkaan optimaalinen ratkaisu, jos alkavan stemman tarkoituksena on antaa palautetta pelaajalle, sillä myöhässä tuleva palaute saattaisi hämmentää pelaajaa.

Adaptiivista musiikkia sävellettäessä ja tuotettaessa on myös varmistettava, että kappaleen kehitys on jotakuinkin looginen, vaikka kappaleen performanssi onkin joka soittokerralla erilainen. Paras tapa tämän saavuttamiseen ainakin tank-

kipelin osalta on siirtyä kappaleen seuraavaan osaan aina pelin edetessä kunkin tason seuraavaan vaiheeseen (ks. kuva 11). Pelin valikoissa taas vietetään todennäköisesti niin vähän aikaa, että voidaan käyttää lyhyempiä kappaleita, jotka kulkevat itsekseen ilman adaptiivisuutta jokaisen kappaleen osan läpi. Ainoa adaptiivisuus tulee vertikaalisesta stemmojen vaihtelusta.

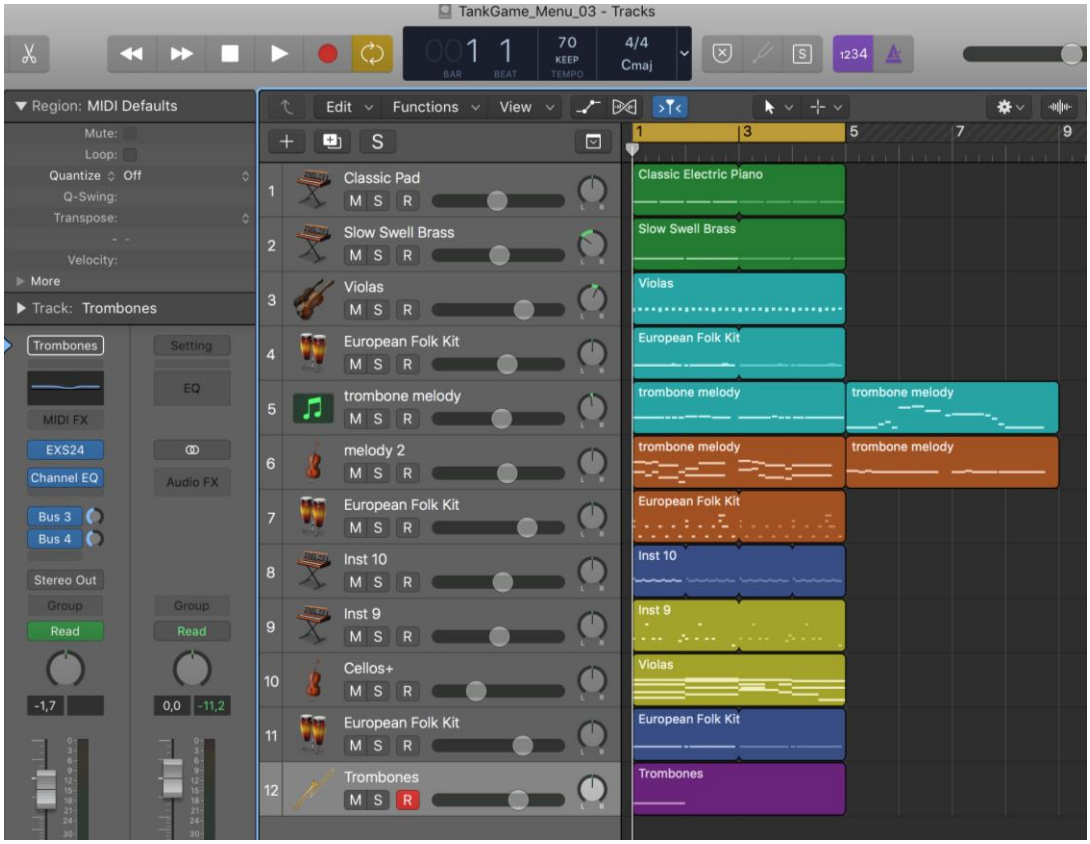


Kuva 11. Esimerkkikaavio logiikasta, jonka perusteella stemmoja ja kappaleita vaihdellaan yhden pelitason sisällä pelin etenemisen mukaan.

Lähtökohtana adaptiivisen musiikin säveltämisessä on aina pelin taso, jota varten kappale sävelletään. Musiikki tehdään pelin ehdoilla; kappaleeseen on sisällytettävä melodioita ja musikaalisia repliikkejä mieluusti jokaista mahdollista pelaajan kannalta merkityksellistä pelitapahtumaa varten, joita voidaan soittaa horisontaalisesti tarpeen vaatiessa. Nämä olisivat suurimmaksi osaksi eräänlaisia johtoaiheita, kuten tankkipelin pääteeman melodia, jota voitaisiin soittaa esimerkiksi pelaajan ollessa voitolla tai tämän läpäistessä pelin. Pelin kehityksen jatkuessa sävellettäisiin lisää johtoaiheita eri tarkoituksia varten. Pelin tarinan riivaajalle tarvittaisiin oma pääteema, kuin myös parille erikoislaatuiselle viholliselle. Heidän johtoaiheitansa voitaisiin soittaa, kun halutaan kertoa pelaajalle, että tämä on tekemisissä jonkin haastavamman vastustajan kanssa. Myös pelaajan omat joukot, pelin eri miljööt sekä pelin voittaminen ja häviäminen saisivat omat johtoaiheensa.

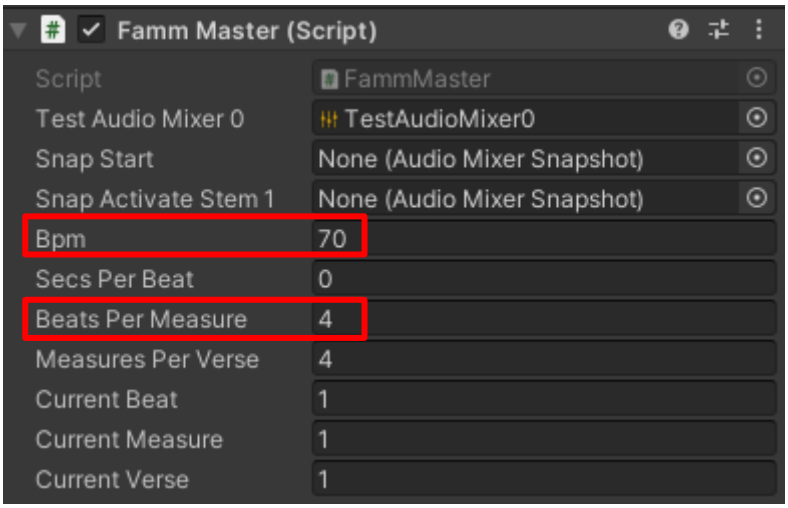
4.5 Kehitystyön lopputulos ja FAMMin implementaatio

FAMMin prototyyppi tuotiin testausta varten tankkipeliin. Testiscenenä toimi tankkipelin päävalikkoscene. Ideana oli kokeilla FAMMin vertikaalista ja horisontaalista adaptiivisuutta niin, että musiikki muuttuu kulloinkin valittuna olevan menunappulan perusteella. Sävelsin ja tuotin kappaleen päävalikkoa varten, jossa on erilaisia vaiheita, yksi kutakin valikon nappulaa kohden (ks. kuva 12).



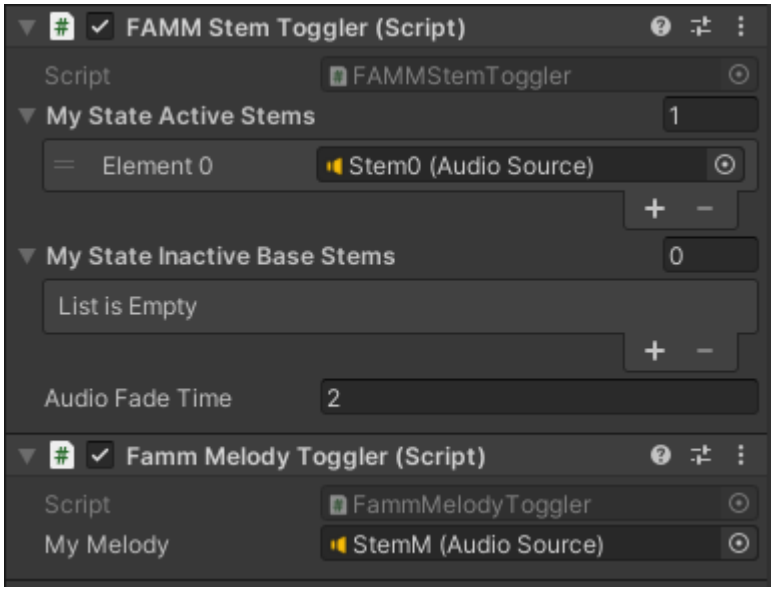
Kuva 12. Näkymä Logic Pro X:stä, jota käytettiin musiikin tuottamiseen. Kuvassa on päävalikon kappaleen raidat (engl. track) värikoodattuna niin, että kukin väri edustaa yhtä stemmaa.

FammMaster –pääscriptin muuttujille annettiin scenessä inspectorin kautta säveltämäni kappaleen tempo eli 70 bpm ja tahtilaji eli 4/4 (ks. kuva 13).



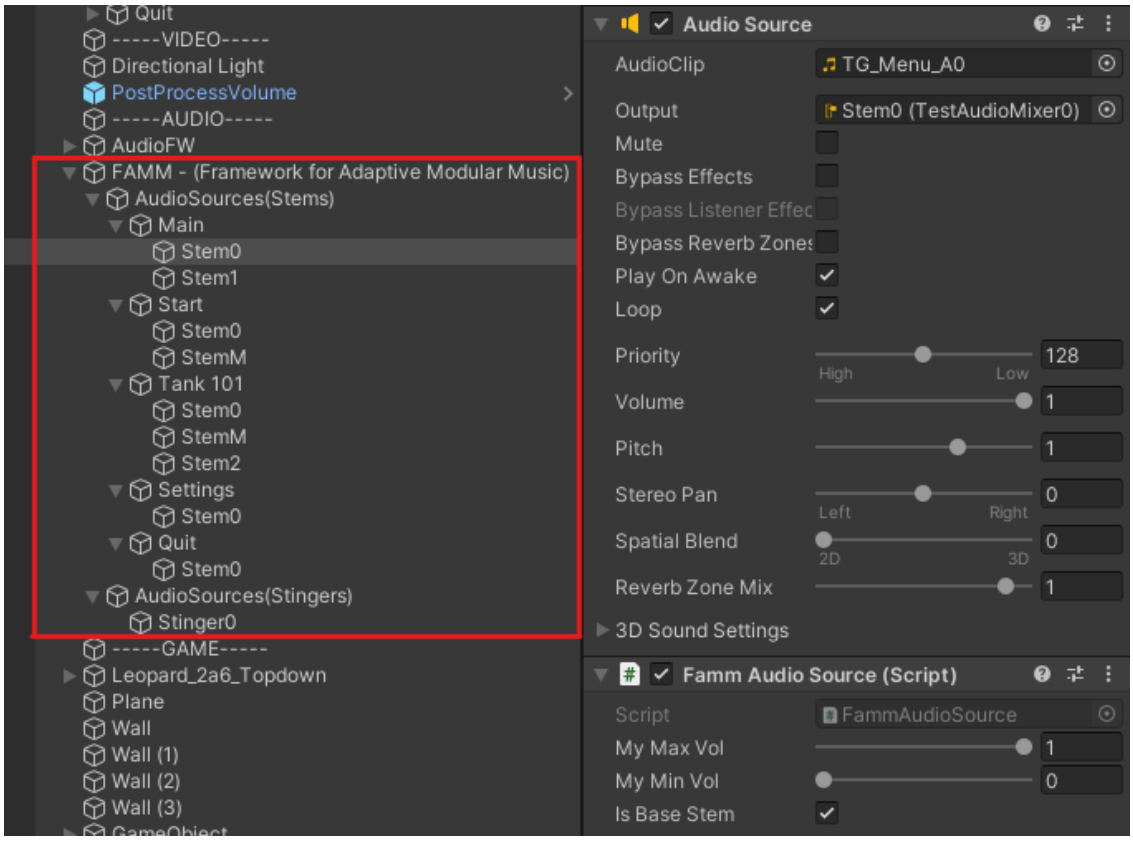
Kuva 13. FammMaster-scriptin bpm- ja beatsPerMeasure-muuttujien (korostettu punaisella) arvot asetetaan FAMMiin manuaalisesti. Korostettujen muuttujien välissä näkyvän secsPerBeat-muuttujan arvo lasketaan automaattisesti bpm:n ja beatsPerMeasuren avulla.

Scenessä soi taustalla oletusarvoisesti kaksi matalataajuista tai bassomaista pohjastemmaa, ja jokainen valikon nappula edustaa tiettyä vaihetta kappaleesta. Jokaisen nappulan peliobjektissa on scripti (ks. kuva 14), jolle on annettu hierarkiasta audio sourceja scriptin muuttujiin. Muuttujille annetut audio sourcet, eli käytännössä stemmat, aktivoituvat nappulan tullessa valituksi. Nappulan scriptille voidaan myös haluttaessa syöttää tieto pohjastemmoista, jotka tulisi hiljentää nappulan ollessa valittuna.



Kuva 14. Päävalikon ”Start” -nappulan scriptit, joita käytetään audio sourcejen manipulointiin.

Kappaleen stemmat sijaitsevat kukin omassa peliobjektissaan itse scenessä (ks. kuva 15). Objekteissa on audio source –komponentit, joita käytetään stemman, melodian tai nk. stingerin, eli lyhyen musikaalisen iskurepliikin, kuten torven töräyksen, äänentoistoon. Stemmojen peliobjektit ovat FAMMin pääpeliobjektin lapsina.



Kuva 15. Unityn hierarkiassa FAMMin parent-peliobjektin alla on sekä stemmat että stingerit peliobjekteina, omiin kategorioihinsa lajiteltuna (punaisella korostettu alue).

Stemmojen audio sourceissa on kytketty “play on awake”-ja “loop”-ominaisuudet päälle, mikä varmistaa stemmojen yhtenäisen ajoituksen. Jokaisen stemman, melodian tai stingerin peliobjektissa on myös FammAudioSource-scripti, johon syötetään kunkin audio sourcen maksimi- ja minimiäänenvoimakkuudet. Näiden arvojen välillä tehdään äänenvoimakkuuden häivytys FadeIn- ja FadeOut-funktioita ajettaessa. Scriptissä on myös isBaseStem-booleani, jonka kautta määritellään, mitkä stemmat ovat oletusarvoisesti äänessä olevia pohjastemmoja (ks. kuva 15).

Stemmojen aktivointia ja hiljentämistä hallinnoidaan kolmella scriptillä, jotka voidaan sijoittaa mihin tahansa peliobjektiin scenessä: FammStemToggler stemmojen hallinnointia varten, FammMelodyToggler melodioiden soittoa varten ja FammStingerTrigger stingereiden soittoa varten. FammStemToggler ja FammMelodyToggler sisältävät On- ja Off-funktiot niille annettujen stemmojen aktivointia ja hiljentämistä varten. FammStemTogglerin StemOn- ja StemOff-funktiot (ks. kuva 16) on tarkoitettu ajettaviksi erillisestä scriptistä käsin, esimerkiksi Unity eventillä. Erillisillä scripteillä voidaan sitten muokata sitä logiikkaa, jonka perusteella Unity eventtejä ajetaan. FammMelodyTogglerin MelodyOn- ja MelodyOff-funktiot taas antavat suoraan FammMaster-scriptille melodian, joka soitetaan kerran aina heti uuden säkeistön alkaessa.

```
public void StemOn() {
    foreach (AudioSource stem in myStateActiveStems) {
        StartCoroutine(masterScript.FadeIn(stem, audioFadeTime));
    }
    foreach (AudioSource stem in myStateInactiveBaseStems) {
        StartCoroutine(masterScript.FadeOut(stem, audioFadeTime));
    }
}
```

Kuva 16. FammStemTogglerin StemOn-funktion koodi, joka käskee FammMasteria ajamaan äänenvoimakkuuden häivytyskorutiinin kyseessä olevan stemman audio sourcella. StemOff-funktio käyttää täsmälleen samaa logiikkaa, mutta kutsuu FammMasterin FadeIn- ja FadeOut-funktioita käänteisessä järjestyksessä.

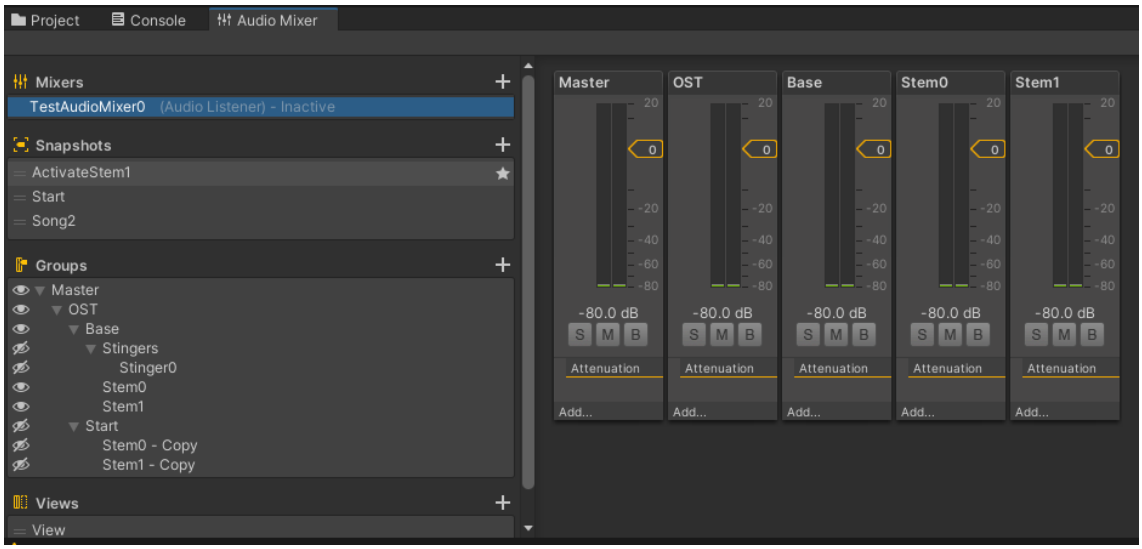
FammStingerTrigger sisältää funktion QueueForNextBeat, joka antaa stingerin suoraan FammMaster-scriptille soitettavaksi seuraavan iskun alkaessa. Toistaiseksi QueueForNextBeat ajetaan testaamista varten jotakin näppäintä painamalla, mutta lopullisessa versiossa funktio voidaan ajaa minkä tahansa logiikan mukaan. FAMMin lopulliseen versioon on myös suunnitteilla lisätä toiminnallisuus äänitiedoston soittamisen aloittamiseksi seuraavan tahdin alkaessa. Tämä olisi hyödyllistä harvinaisempien, alle säkeistön mittaisten äänitiedostojen, kuten lyhyiden johtoaiheiden, soittoa varten.

5 Yhteenveto

5.1 Kehitys ja sen haasteet

Opinnäytetyön käytännön osuuden, FAMMin, kehitystyö oli loppujen lopuksi ohjelmoinnin osalta paljon odotettua ketterämpi prosessi, joskin suunnitteluun, prototypointiin ja tutkimustyöhön kului vuorostaan paljon arvioitua enemmän aikaa. Suurin haaste oli selvittää, mikä olisi paras lähestymistapa adaptiivisen musiikkijärjestelmän toteuttamiseen. Ratkaisua etsittiin muiden pelien toteutuksista, mutta lopulta valinta tehtiin laajalti Unity-prototypoinnin tulosten perusteella. Vertikaalisen ja horisontaalisen adaptiivisuuden yhdisteleminen oli musiikin säveltämisen ja FAMMin kehitystyön kannalta nopein ratkaisu; generatiivinen musiikki ja MIDIn käyttäminen jäivät (ainakin toistaiseksi) taka-alalle.

FAMMia ohjelmoitaessa kaksi kysymystä nousi työvaiheen keskeisiksi haasteiksi: äänenvoimakkuuden häivyttäminen ajan funktiona ja musiikin tahdin seuranta koodissa. Äänenvoimakkuuden häivyttämistä varten hyödynnettiin aluksi Unityn audio mixereiden (ks. kuva 17) snapshot-toiminnallisuutta, mutta tämä todettiin hyvin pian liian työlääksi lähestymistavaksi. Äänenvoimakkuus häivytetään alusta asti ohjelmoiduilla funktioilla, jotka interpoloivat kahden float-muuttujan välillä ja asettavat tämän halutun audio sourcen äänenvoimakkuuden suuruudeksi. Tahdin seurantaa varten jaetaan kappaleen tempo 60:lla ja laskun lopputulos annetaan float-muuttujalle “secsPerBeat”, joka on tahdin seurannan perustana. Tieto kunkin kappaleen temposta ja tahtilajista on syötettävä manuaalisesti FAMMille, mutta loput laskennasta hoituu automaattisesti.



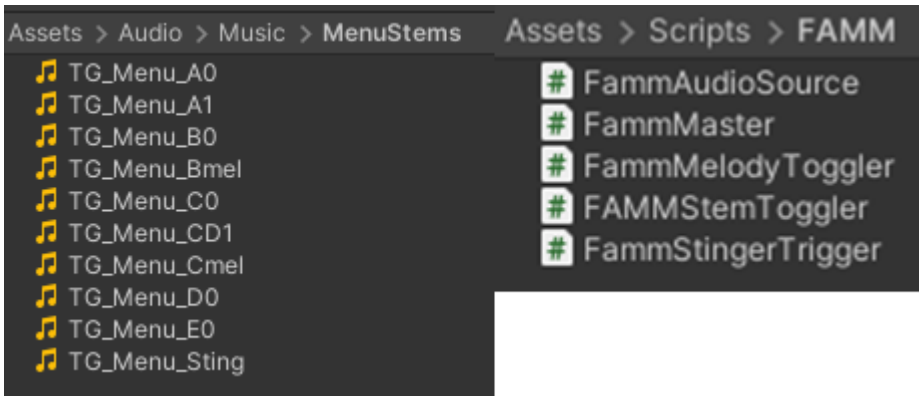
Kuva 17. Unityn audio mixereillä voidaan reitittää audio sourcen äänilähtö mikseriin, jonka avulla voidaan hallita äänentoistoa.

Musiikin tuottaminen adaptiivisesta lähtökohdasta pakotti lähestymään säveltämistä eri tavalla; koska lopullinen performanssi oli joka kerralla erilainen, oli varmistettava, että kappaleen eri osat voisivat soida eri järjestyksessä ja silti käydä järkeen. Tempo ja sävellaji olivat suureksi osaksi lukittuja, mikä rajoitti säveltämistyötä ja pelitaso, jota varten kappale sävellettiin, määräsi pitkälti kappaleelta edellytetyn tunnelman. Jälkeenpäin ajateltuna nämä rajoitteet kuitenkin kenties jopa paransivat säveltäjän luovuutta, koska ne vähensivät mahdollisia lähestymistapoja ja siten selkeyttivät säveltäjän tavoitetta.

Kehitystyöstä suurin osa toteutettiin tankkipelin ympärille. Pelin tarjoama konkreettinen testausalusta oli keskeinen osa FAMMin kehityksen nopeutta ja loogista rakennetta. Peliympäristö pakotti toimivaan ja tehokkaseen lopputulokseen, sillä jokainen kehitystyötä ja FAMMin käyttämistä kehittäjän näkökulmasta hidastava seikka tuli esille moneen kertaan. Esimerkiksi stemmojen äänitiedostot oli nimettävä niin, että jo nimi kertoi, onko tiedosto melodia tai stingeri ja että nimestä näkisi tiedoston järjestyksen osana kappaletta. (kuva 18). Päävalikko toi esille tarpeita uusille ominaisuuksille, joista kaikkia ei ehditty edes toteuttamaan, kuten audion soittamisen ajastaminen seuraavan tahdin alkuun.

5.2 Jatkokehitys

Muutamaa lisäominaisuutta ja pientä siloittelua vaaditaan, jos FAMM koskaan julkaistaan vapaasti käytettäväksi, esimerkiksi Unityn asset storeen. Ensinnäkin FAMMin koodi on FammMaster-scriptiä lukuunottamatta kommenttivapaa. Käyttäjäystävällisyyden nimissä olisi suotavaa lisätä enimmäkseen funktioiden viereen muutamia lyhyitä kommentteja englanniksi, jotka selkeyttäisivät koodin toimintaperiaatteita lukijalle ja nopeuttaisivat uuden käyttäjän perehtymistä FAMMiin. Myös tiedostojen nimiä tulisi hienosäätää; joissakin tiedostoissa FAMM on kirjoitettu kokonaan isoilla kirjaimilla, joissakin vain F-kirjain on iso. (ks. kuva 18).



Kuva 18. FAMMin käyttämät scriptit ja päävalikon musiikin käyttämät stemmat Unityn project-ikkunassa.

FAMMin vielä vaatimista lisäominaisuuksista tärkein ennen FAMMin mahdollista julkaisua olisi uudella tahdilla alkavan äänitiedoston soittaminen, sillä tällä hetkellä äänitiedostoja voidaan ajoittaa vain alkaville iskuille tai säkeistöille. Äärimmäisen mielenkiintoinen jatkokehityksen suunta olisi MIDI-ominaisuuksien tarkastelu: jos FAMMiin saisi liitettyä toiminnallisuuden, joka sallisi MIDI-tiedostojen soittamisen, etenkin millä tahansa MIDI-soittimella, FAMMin soveltamismahdollisuudet moninkertaistuisivat. Tällöin samaa melodiaa voitaisiin haluttaessa soittaa joka soittokerralla eri soittimella, mikä yksin jo lisäisi merkittävästi yksittäisten stemmojen variaatiomahdollisuuksia, puhumattakaan MIDI-tiedostojen mahdollistamasta jälkikäsittelystä. Saman soittimen tuottama ääni olisi melodiasta riippumatta samanlainen, mikä tekisi transitioista teoriassa huomaamattomia.

5.3 Tutkimustyön tulokset

Tutkimustyön lopuksi löytämäni listamuotoinen määritelmä adaptiiviselle musiikille oli hyödyllinen aiheen selittämisessä, mutta käsitteen sisäistämistä varten oli silti katsottava muutamia esimerkkejä adaptiivisen musiikin toteutuksista peleissä (Clark 2007). Tutkimustyötä hidasti ensin se, että aihetta ympäröivää termistöä käytettiin niin epäjohdonmukaisesti, mutta tämä havainto johti onneksi parempaan tarkkaavaisuuteen tutkimustyötä tehdessä. Varsinkin Toni Kähkösen (2021) kirjoitus adaptiivisesta musiikista valaisi aihetta loistavasti. Opinnäytetyön kirjallisen osuuden kohdalla oli loppujen lopuksi osittain itse valittava, mitä interaktiivisen musiikin termiä käytettiin mihinkäkin aiheeseen viittaamiseen, koska niitä käytettiin niin vaihtelevin merkityksin eri kirjoittajien toimesta.

Tutkimustyön perusteella interaktiivista pelimusiikkia on tehty niin kauan kuin itse pelejäkin (yksi vanhimpia esimerkkejä interaktiivisesta pelimusiikista löytyy *Froggerista* (Konami 1981) (Kähkönen 2021), mutta laajasti sekä vertikaalista että horisontaalista adaptiivisuutta hyödyntäviä, edullisia ja tehokkaan kokoisia interaktiivisen musiikin ratkaisuja on indie-kehittäjälle tarjolla huomattavan vähän siihen verrattuna, kuinka paljon tietoa aiheesta on nykyään saatavilla. Käytännössä indie-kehittäjä voi käyttää joko FMODin tai Wwisen kaltaista audionhallintajärjestelmää, mutta useimpia indie-pelejä varten tällainen iso ohjelmisto tarjonnee aivan liikaa kaikkea tarpeetonta halutun toiminnallisuuden lisäksi, ja pahimmillaan käyttäjä joutuu vielä maksamaan ohjelmiston käytöstä. Myöskään esimerkiksi Unityn asset storesta ei löydy kustannustehokkaita ja todella toimivia ratkaisuja adaptiivisen musiikin toteuttamiseen. FAMM tarjoaa nöyrästi muutoksen mahdollisuutta tähän vallitsevaan tilanteeseen.

Lähteet

Ahoy 2018. RetroAhoy: The Secret of Monkey Island. Youtube.com. Katsottavissa osoitteessa <youtube.com/watch?v=9F9ahZQ7oP0> (katsottu 17.10.2022). 1:13:53.

Audiokinetic 2022. Powered by Wwise. <audiokinetic.com/en/discover/wwise-in-games/> (luettu 22.11.2022)

Callighan, Elliot 2022. Adaptive audio in menus: game development's untapped opportunity. Games Industry.biz. <gamesindustry.biz/adaptive-audio-game-developments-untapped-opportunity> (luettu 14.9.2022).

Cedeno, Noah 2018. The Magic and Application of the Leitmotif in Video Game Osts. HubPages. <discover.hubpages.com/entertainment/The-Magic-of-the-Leitmotif-in-Video-Game-OSTs> (luettu 20.11.2022).

Clark, Andrew 2007. Defining Adaptive Music. Game Developer. <gamedeveloper.com/audio/defining-adaptive-music> (luettu 30.8.2022).

Firelight Technologies 2022a. Featured games. <fmod.com/games> (luettu 22.11.2022).

Firelight Technologies 2022b. Licensing. <fmod.com/licensing> (luettu 22.11.2022).

Game Maker's Toolkit 2014. Adaptive Soundtracks in Games. Youtube.com. <youtube.com/watch?v=b0gvM4q2hdI> (katsottu 9.9.2022). 9:24.

Hahn, Michael 2020. What Is MIDI? How To Use the Most Powerful Tool in Music. Landr Blog. <blog.landr.com/what-is-midi/> (luettu 1.9.2022).

Keane, Paul 2021. Howard Shore in The Lord of The Rings: How to use leitmotif technique to create a masterpiece? Taketones.com. <taketones.com/blog/howard-shore-in-the-lord-of-the-rings-how-to-use-leitmotif-technique-to-create-a-masterpiece> (luettu 17.11.2022).

Kähkönen, Toni 2021. A Look Into Adaptive Video Game Music. GDWC. <thegdwc.com/blog/blog.php?blog_id=134> (luettu 27.8.2022).

Maher, Jimmy 2017. Loom (or, how Brian Moriarty Proved That Less is Sometimes More). The Digital Antiquarian. <filfre.net/2017/02/loom-or-how-brian-moriarty-proved-that-less-is-sometimes-more/> (luettu 3.12.2022).

Noodle 2019. Adaptive Music (In Gaming) Is Amazing. Youtube.com. Katsottavissa osoitteessa <youtube.com/watch?v=yLd5wmBNCBM> (katsottu 22.8.2022). 16:48.

Ó Nuanáin, Cárthach 2019. A Brief History of Machine-Assisted Music in Video Games. Medium. <medium.com/the-sound-of-ai/a-brief-history-of-machine-assisted-music-in-video-games-4f249d6b7ef5#> (luettu 26.8.2022).

Pidkameny, Eric 2002. Levels of Sound. <vgmusic.com/information/vgpaper2.html> (luettu 2.11.2022).

reddeadwikia 2011. Red Dead Redemption - Making of the Music [HD]. Youtube.com. Katsottavissa osoitteessa <youtube.com/watch?v=klkWMkrO0pg> (katsottu 25.8.2022). 4:49.

Ricketts, Ed 2022. Ultima Underworld transformed first-person games forever. PC Gamer. <pcgamer.com/reinstall-ultima-underworld-the-stygian-abyss/> (luettu 2.12.2022).

Sideways 2016. Theme vs. Leitmotif. Youtube.com. Katsottavissa osoitteessa <youtube.com/watch?v=qVlsIhbQ2qM> (katsottu 27.10.2022). 5:40.

Silk, Peter 2010. iMUSE Demonstration 1 - Seamless Endings. Youtube.com. Katsottavissa osoitteessa <youtube.com/watch?v=AjtxK_WT784> (katsottu 30.8.2022). 1:43.

Swift, Andrew 2012. An introduction to MIDI. <web.archive.org/web/20120830211425/http://www.doc.ic.ac.uk/~nd/surprise_97/journal/vol1/aps2/> (luettu 20.11.2012).

Wilde, Tyler 2013. The music of FTL: Faster Than Light - an interview with composer Ben Prunty. PC Gamer. <pcgamer.com/ftl-faster-than-light-soundtrack-ben-prunty/> (luettu 3.12.2022).

Aineistona käytetyt pelit ja elokuvat

Cyberpunk 2077 2020. Puola: CD Projekt Red.

Doom 1993. Yhdysvallat: id Software.

Frogger 1981. Japani: Konami.

FTL: Faster Than Light 2012. Yhdysvallat: Subset Games.

Grim Fandango 1998. Yhdysvallat: LucasArts.

Halo: Combat Evolved 2001. Yhdysvallat: Bungie.

Loom 1990. Yhdysvallat: Lucasfilm Games.

Mario Kart Wii 2008. Japani: Nintendo.

Monkey Island 2: LeChuck’s Revenge 1991. Yhdysvallat: LucasArts.

Red Dead Redemption 2010. Yhdysvallat: Rockstar San Diego.

Sam & Max Hit the Road 1993. Yhdysvallat: LucasArts.

Spore 2008. Yhdysvallat: Maxis.

Taru Sormusten Herrasta: Sormuksen ritarit (The Lord of the Rings: The Fellowship of the Ring). 2001. Fran Walsh. Peter Jackson. Uusi-Seelanti: New Line Cinema. 178 min.

Ultima Underworld: The Stygian Abyss 1992. Yhdysvallat: Blue Sky Productions.

World of Tanks 2010. Valko-Venäjä: Wargaming